01.01_计算机基础知识(计算机概述)(了解)

- A:什么是计算机?计算机在生活中的应用举例
  - 计算机（Computer）全称：电子计算机，俗称电脑。是一种能够按照程序运行，自动、高速处理海量数据的现代化智能电子设备。由硬件和软件所组成，没有安装任何软件的计算机称为裸机。常见的形式有台式计算机、笔记本计算机、大型计算机等。
  - 应用举例
    - 1：科学计算
    - 2、数据处理
    - 3、自动控制
    - 4、计算机辅助设计
    - 5、人工智能
    - 6、多媒体应用
    - 7、计算机网络
    - ...
- B:什么是硬件?硬件举例
  - 计算机硬件（Computer Hardware）是指计算机系统中由电子，机械和光电元件等组成的各种物理装置的总称。这些物理装置按系统结构的要求构成一个有机整体为计算机软件运行提供物质基础。
  - 冯.诺依曼体系结构
  - 计算机的硬件分成5大组成部件：运算器、控制器、存储器、输入设备和输出设备。
    - 运算器和控制器是计算机的核心，合称中央处理单元（Central Processing Unit，CPU）或处理器。CPU的内部还有一些高速存储单元，被称为寄存器。其中运算器执行所有的算术和逻辑运算；控制器负责把指令逐条从存储器中取出，经译码后向计算机发出各种控制命令；而寄存器为处理单元提供操作所需要的数据。
    - 存储器是计算机的记忆部分，用来存放程序以及程序中涉及的数据。它分为内部存储器和外部存储器。内部存储器用于存放正在执行的程序和使用的数据，其成本高、容量小，但速度快。外部存储器可用于长期保存大量程序和数据，其成本低、容量大，但速度较慢。
    - 输入设备和输出设备统称为外部设备，简称外设或I/O设备，用来实现人机交互和机间通信。微型机中常用的输入设备有键盘、鼠标等，输出设备有显示器、打印机等。
- C:什么是软件?软件分类及举例
  - 计算机软件(Computer Software)是使用计算机过程中必不可少的东西，计算机软件可以使计算机按照事先预定好的顺序完成特定的功能
  - 计算机软件按照其功能划分为系统软件与应用软件
    - 系统软件： DOS(Disk Operating System), Windows, Linux, Unix, Mac, Android, iOS
    - 应用软件：office QQ聊天 YY语言 扫雷

01.02_计算机基础知识(软件开发和计算机语言概述)(了解)

- A:什么是软件
  - 按照特定顺序组织的计算机数据和指令的集合
- B:什么是开发
  - 软件的制作过程
- C:什么是软件开发
  - 借助开发工具与计算机语言制作软件
- D:什么是计算机语言
  - 人与计算机之间进行信息交流沟通的一种特殊语言
- E:计算机语言的分类
  - 机器语言：
    - 机器语言是直接用二进制代码指令表达的计算机语言，指令是用0和1组成的一串代码，它们有一定的位数，并分成若干段，各段的编码表示不同的含义。
  - 汇编语言：
    - 汇编语言是使用一些特殊的符号来代替机器语言的二进制码，计算机不能直接识别，需要用一种软件将汇编语言翻译成机器语言。
  - 高级语言：
    - 使用普通英语进行编写源代码，通过编译器将源代码翻译成计算机直接识别的机器语言，之后再由计算机执行。
    - 高级语言包括C,C++,C#,JAVA

01.03_计算机基础知识(人机交互)(了解)

- A:人机交互的两种方式
  - a:命令行方式
    - 需要有一个控制台，输入特定的指令，让计算机完成一些操作。较为麻烦，需要记录住一些命令。
  - b:图形化界面方式
    - 这种方式简单直观，使用者易于接受，容易上手操作。

01.04_计算机基础知识(键盘功能键和快捷键)(掌握)

- A:键盘功能键
  - a:Tab
  - b:Shift
  - c:Ctrl
  - d:Alt

    - e:空格
    - f:Enter
    - g:Window
    - h:上下左右键
    - i:PrtSc(PrintScreen)屏幕截图
- B:键盘快捷键
    - a:Ctrl+A 全选
    - b:Ctrl+C 复制
    - c:Ctrl+V 粘贴
    - d:Ctrl+X 剪切
    - e:Ctrl+Z 撤销
    - f:Ctrl+S 保存

01.05_计算机基础知识(如何打开DOS控制台)(掌握)

- A:xp下如何打开DOS控制台？
    - a:开始--程序--附件--命令提示符
    - b:开始--运行--cmd--回车
    - c:win+r--cmd--回车
- B:win7下如何打开DOS控制台？
    - a:开始--所有程序--附件--命令提示符
    - b:开始--搜索程序和文件--cmd--回车
    - c:win+r--cmd--回车
- C:win8下如何打开DOS控制台
    - a:鼠标左击开始--下箭头--命令提示符
    - b:鼠标右击开始--搜索--cmd--回车
    - c:鼠标右击开始--运行--cmd--回车
    - d:win+r--cmd--回车

01.06_计算机基础知识(常见的DOS命令讲解)

- A:d: 回车 盘符切换
- B:dir(directory):列出当前目录下的文件以及文件夹
- C:cd (change directory)改变指定目录(进入指定目录)
- D:cd.. : 退回到上一级目录
- E:cd\: 退回到根目录
- F:cls : (clear screen)清屏
- G:exit : 退出dos命令行(分割线上的需要掌握,下的了解)
- /==
- md (make directory) : 创建目录
- rd (remove directory): 删除目录
- del (delete): 删除文件,删除一堆后缀名一样的文件*.txt
- notepad 创建文件
- 删除带内容的文件夹
    - rd + /s 文件夹名称(询问是否删除)
    - rd + /q + /s 文件夹名称(直接删除)

01.07_Java语言基础(Java语言概述)(了解)

- A:Java语言发展史
    - 詹姆斯·高斯林（James Gosling）1977年获得了加拿大卡尔加里大学计算机科学学士学位，1983年获得了美国卡内基梅隆大学计算机科学博士学位，毕业后到IBM工作，设计IBM第一代工作站NeWS系统，但不受重视。后来转至Sun公司，1990年，与Patrick，Naughton和Mike Sheridan等人合作“绿色计划”，后来发展一套语言叫做“Oak”，后改名为Java。
    - SUN(Stanford University Network，斯坦福大学网络公司)
- B:Java语言版本
    - JDK 1.1.4 Sparkler 宝石 1997-09-12
    - JDK 1.1.5 Pumpkin 南瓜 1997-12-13
    - JDK 1.1.6 Abigail 阿比盖尔--女子名 1998-04-24
    - JDK 1.1.7 Brutus 布鲁图--古罗马政治家和将军 1998-09-28
    - JDK 1.1.8 Chelsea 切尔西--城市名 1999-04-08
    - J2SE 1.2 Playground 运动场 1998-12-04
    - J2SE 1.2.1 none 无 1999-03-30
    - J2SE 1.2.2 Cricket 蟋蟀 1999-07-08
    - J2SE 1.3 Kestrel 美洲红隼(sǔn) 2000-05-08
    - J2SE 1.3.1 Ladybird 瓢虫 2001-05-17
    - J2SE 1.4.0 Merlin 灰背隼 2002-02-13
    - J2SE 1.4.1 grasshopper 蚱蜢 2002-09-16
    - J2SE 1.4.2 Mantis 螳螂 2003-06-26
    - JAVASE 5.0 (1.5.0) Tiger 老虎
    - JAVASE 5.1 (1.5.1) Dragonfly 蜻蜓

    - JAVASE 6.0 (1.6.0) Mustang 野马
    - JAVASE 7.0 (1.7.0) Dolphin 海豚
- C:Java语言平台
    - J2SE(Java 2 Platform Standard Edition)标准版
        - 是为开发普通桌面和商务应用程序提供的解决方案,该技术体系是其他两者的基础，可以完成一些桌面应用程序的开发
    - J2ME(Java 2 Platform Micro Edition)小型版
        - 是为开发电子消费产品和嵌入式设备提供的解决方案
    - J2EE(Java 2 Platform Enterprise Edition)企业版
        - 是为开发企业环境下的应用程序提供的一套解决方案,该技术体系中包含的技术如 Servlet、Jsp等，主要针对于Web应用程序开发
- C:Java语言特点
    - 简单性
    - 解释性
    - 面向对象
    - 高性能
    - 分布式处理
    - 多线程
    - 健壮性
    - 动态
    - 结构中立
    - 安全性
    - 开源
    - 跨平台

01.08_Java语言基础(Java语言跨平台原理)(掌握)

- A:什么是跨平台性
- B:Java语言跨平台原理
    - 只要在需要运行java应用程序的操作系统上，先安装一个Java虚拟机(JVM Java Virtual Machine)即可。由JVM来负责Java程序在该系统中的运行。
- C:Java语言跨平台图解
    - write once ,run anywhere!(一处编译,到处运行)

01.09_Java语言基础(JRE和JDK的概述)(掌握)

- A:什么是JRE
    - 包括Java虚拟机(JVM Java Virtual Machine)和Java程序所需的核心类库等，如果想要运行一个开发好的Java程序，计算机中只需要安装JRE即可。
    - JRE:JVM+类库。
- B:什么是JDK
    - JDK是提供给Java开发人员使用的，其中包含了java的开发工具，也包括了JRE。所以安装了JDK，就不用在单独安装JRE了。
    - 其中的开发工具：编译工具(javac.exe) 打包工具(jar.exe)等
    - JDK:JRE+JAVA的开发工具。
- C:为什么JDK中包含一个JRE
    - 为什么JDK中包含一个JRE呢？
        - 开发完的程序，需要运行一下看看效果。
- D:JDK,JRE,JVM的作用和关系

01.10_Java语言基础(JDK的下载和安装过程图解)(了解)

- A:JDK的下载
    - a:官网 http://www.oracle.com
    - b:演示下载流程
- B:JDK的安装
    - a:傻瓜式安装
        - 双击安装程序，然后一路next即可(但是不建议)
    - b:安装的推荐方式
        - 安装路径不要有中文或者特殊符号如空格等。
        - 所有和开发相关的软件最好安装目录统一。
            - 举例：我的JDK安装路径
                - D:\develop\Java\jdk1.7.0_72
        - 当提示安装JRE时，可以选择不安装。建议还是安装上。
    - c:演示安装流程
        - 可以先在d盘建立一个文件夹develop
        - 然后演示安装过程
- C:验证安装是否成功
    - a:通过DOS命令，切换到JDK安装的bin目录下。
        - D:\develop\Java\jdk1.7.0_72\bin
    - b:然后分别输入javac和java，如果正常显示一些内容，说明安装成功

01.11_Java语言基础(JDK安装路径下的目录解释)(了解)

- a:bin目录：该目录用于存放一些可执行程序。

    - 如javac.exe（java编译器）、java.exe(java运行工具)，jar.exe(打包工具)和* javadoc.exe(文档生成工具)等。
- b:db目录：db目录是一个小型的数据库。
    - 从JDK 6.0开始，Java中引用了一个新的成员JavaDB，这是一个纯Java实现、开源的数据库管理系统。这个数据库不仅轻便，而且支持JDBC 4.0所有的规范，在学习JDBC 时，不再需要额外地安装一个数据库软件，选择直接使用JavaDB即可。
- c:jre目录："jre"是 Java Runtime Environment 的缩写，意为Java程序运行时环境。此目录是Java运行时环境的根目录，它包括Java虚拟机，运行时的类包，Java应用启动器以及一个bin目录，但不包含开发环境中的开发工具。
- d:include目录：由于JDK是通过C和C++实现的，因此在启动时需要引入一些C语言的头文件，该目录就是用于存放这些头文件的。
- e:lib目录：lib是library的缩写，意为 Java 类库或库文件，是开发工具使用的归档包文件。
- f:src.zip文件：src.zip为src文件夹的压缩文件，src中放置的是JDK核心类的源代码，通过该文件可以查看Java基础类的源代码。

01.12_Java语言基础(Java开发工具介绍)(了解)

- A:notepad(微软操作系统自带)
- B:Editplus/Notepad++
- C:Eclipse
- D:MyEclipse
    - 给大家简单的介绍一下这些工具，然后说说我们使用这些工具的顺序。
    - 基础班：先notepad，然后Editplus，再Eclipse。
    - 就业班：MyEclipse和Eclipse都用。

01.13_Java语言基础(HelloWorld案例的编写和运行)(掌握)

- A:定义类
- B:写main方法
- C:写输出语句
- D:Java程序开发运行与工作原理
- E:编译和运行程序

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("HelloWorld");
    }
}
```

01.14_Java语言基础(HelloWorld案例常见问题)(掌握)

- A:找不到文件(都演示一下，让学生看看出现的都是什么问题)
    - a:文件扩展名隐藏导致编译失败
    - b:文件名写错了
- B:单词拼写问题(都演示一下，让学生看看出现的都是什么问题)
    - a:class写成Class
    - b:String写成string
    - c:System写成system
    - d:main写成mian
- C:括号匹配问题(都演示一下，让学生看看出现的都是什么问题)
    - a:把类体的那对大括号弄掉一个
    - b:把方法体的那对大括号弄掉一个
    - c:把输出语句的那对小括号弄掉一个
- D:中英文问题(都演示一下，让学生看看出现的都是什么问题)
    - a:提示信息：错误: 非法字符: \????的格式
    - 注意：java编程中需要的基本上都是英文字符

01.15_Java语言基础(Java语言的书写格式(约定俗成)) (掌握)

- 1,大括号要对齐,并且成对写
- 2,左大括号前面有空格
- 3,遇到左大括号要缩进,Tab
- 4,方法和程序块之间加空行让程序看起来清晰
- 5,并排语句之间加空格,例如for语句
- 6,运算符两侧加空格

01.16_Java语言基础(path环境变量的作用及配置方式1)(掌握)

- A:在JDK的bin目录下开发程序容易产生的问题
    - a:如果文件过多，不方便管理
    - b:删除自己写过的不需要的文件，可能不小心把JDK自带的工具给删除了
- B:如何解决问题呢
    - notepad这样的命令为什么在任何路径下都能够执行,配置path环境变量
- C:配置方式
    - a:xp系统

        - 右键点击桌面计算机→选择属性→选择高级选项卡→点击环境变量→下方系统变量中查找path→双击path→将jdk安装目录下的bin目录添加到最左边并添加分号。
    - b:win7/win8系统
        - 右键点击桌面计算机→选择属性→选择高级系统设置→选择高级选项卡→点击环境变量→下方系统变量中查找path→双击path→将jdk安装目录下的bin目录添加到最左边并添加分号。
- path配置的是可执行的文件.exe,配置后可以在不同的盘符下访问path路径下的可执行文件

01.17_Java语言基础(Path环境变量的配置方式2)(掌握)

- A:先配置JAVA_HOME
- B:再修改path
- C:最后说一下path是有先后顺序关系的

01.18_Java语言基础(classpath环境变量的作用及其配置)(了解)

- A:为什么要配置classpath
- B:classpath配置的原理
- C:如何配置classpath

- path和classpath的区别
    - path配置的是可执行的文件.exe,配置后可以在不同的盘符下访问path路径下的可执行文件
    - classpath配置的java的类文件,就是.class文件

01.19_Java语言基础(Editplus开发程序并编译运行)(了解)

- A:配置快捷键编译运行
- B:去除备份文件

01.20_Java语言基础(注释概述及其分类)(掌握)

- A:什么是注释
- B:注释的分类及讲解
    - 文档注释目前不讲，说后面讲解
- 注释的作用
    - A:解释说明程序
    - B:帮助我们调试错误

01.21_Java语言基础(关键字的概述和使用)(掌握)

- A:什么是关键字
    - 被Java语言赋予特定含义的单词
- B:关键字的特点
    - 组成关键字的字母全部小写
- C:常见关键字
    - public static void class等
- D:关键字的注意事项
    - goto和const作为保留字存在,目前并不使用,类似Editplus这样的高级记事本,针对关键字有特殊的颜色标记，非常直观

01.22_Java语言基础(标识符的概述和组成规则)(掌握)

- A:什么是标识符
    - 就是给类,接口,方法,变量等起名字时使用的字符序列
- B:标识符的组成规则
    - 英文大小写字母
    - 数字字符
    - $和_
- C:标识符注意事项
    - 1,不能使用关键字
    - 2,不能数字开头

01.23_Java语言基础(标识符中常见的命名规则)(了解)

- 见名知意
- A:包
    - 最好是域名倒过来,要求所有的字母小写
- B:类或者接口
    - 如果是一个单词首字母大写
    - 如果是多个单词每个单词首字母大写(驼峰标识)
- C:方法或者变量

    - 如果是一个单词全部小写
    - 如果是多个单词,从第二个单词首字母大写
- D:常量
    - 如果是一个单词,所有字母大写
    - 如果是多个单词,所有的单词大写,用下划线区分每个单词

01.24_day01总结

- 把今天的知识点总结一遍。

02.01_Java语言基础(常量的概述和使用)(掌握)

- A:什么是常量
  - 在程序执行的过程中其值不可以发生改变
- B:Java中常量的分类
  - 字面值常量
  - 自定义常量(面向对象部分讲)
- C:字面值常量的分类
  - 字符串常量 用双引号括起来的内容
  - 整数常量 所有整数
  - 小数常量 所有小数
  - 字符常量 用单引号括起来的内容,里面只能放单个数字,单个字母或单个符号
  - 布尔常量 较为特殊，只有true和false
  - 空常量 null(数组部分讲解)
- D:案例演示
  - 用输出语句输出各种常量。null不演示

02.02_Java语言基础(进制概述和二,八,十六进制图解)(了解)

- A:什么是进制
  - 进制：就是进位制，是人们规定的一种进位方法。 对于任何一种进制--X进制，就表示某一位置上的数运算时是逢X进一位。二进制就是逢二进一，八进制是逢八进一，十进制是逢十进一，十六进制是逢十六进一。
  - 例如一周有七天,七进制,一年有十二个月,十二进制
- B:十进制的由来
  - 十进制的由来是因为人类有十个手指
- C:二进制的由来
  - 其实二进制来源与中国,请看史料记载
  - 18世纪德国数理哲学大师莱布尼兹从他的传教士朋友鲍威特寄给他的拉丁文译本《易经》中，读到了八卦的组成结构，惊奇地发现其基本素数（0）（1），即《易经》的阴爻yao- -和__阳爻，其进位制就是二进制，并认为这是世界上数学进制中最先进的。20世纪被称作第三次科技革命的重要标志之一的计算机的发明与应用，其运算模式正是二进制。它不但证明了莱布尼兹的原理是正确的，同时也证明了《易经》数理学是很了不起的。
- D:八进制的由来
  - 任何数据在计算机中都是以二进制的形式存在的。二进制早期由电信号开关演变而来。一个整数在内存中一样也是二进制的，但是使用一大串的1或者0组成的数值进行使用很麻烦。
  - 所以就想把一大串缩短点，将二进制中的三位用一位表示。这三位可以取到的最大值就是7.超过7就进位了，这就是八进制。
- E:十六进制的由来
  - 但是对于过长的二进制变成八进制还是较长，所以出现的用4个二进制位表示一位的情况，四个二进制位最大是15，这就是十六进制。
- F:不同进制表现同一个数据的形式特点
  - 进制越大，表现形式越短

02.03_Java语言基础(不同进制数据的表现形式)(掌握)

- A:二进制的数据表现形式
  - 由0,1组成。以0b(b可以大写也可以小写)开头(JDK1.7版本可以表示二进制了)
- B:八进制的数据表现形式
  - 由0,1,…7组成。以0开头
- C:十进制的数据表现形式
  - 由0,1,…9组成。整数默认是十进制的
- D:十六进制的数据表现形式
  - 由0,1,…9,a,b,c,d,e,f(大小写均可)。以0x开头
- E:案例演示
  - 输出不同进制表现100的数据。
  - 0b100
  - 0100
  - 100
  - 0x100

02.04_Java语言基础(任意进制到十进制的转换图解)(了解)

- A:任意进制到十进制的转换原理
  - 系数：就是每一位上的数据。
  - 基数：X进制，基数就是X。
  - 权：在右边，从0开始编号，对应位上的编号即为该位的权。
  - 结果：把系数*基数的权次幂相加即可。
- B:画图练习
  - 二进制--十进制
  - 八进制--十进制

- 十六进制--十进制

02.05_Java语言基础(十进制到任意进制的转换图解)(了解)

- A:十进制到任意进制的转换原理
  - 除积倒取余
- B:画图练习
  - 十进制--二进制
  - 十进制--八进制
  - 十进制--十六进制

02.06_Java语言基础(快速的进制转换法)(了解)

- A:8421码及特点
  - 8421码是中国大陆的叫法，8421码是BCD代码中最常用的一种。在这种编码方式中每一位二值代码的1都是代表一个固定数值，把每一位的1代表的十进制数加起来，得到的结果就是它所代表的十进制数码。
- B:通过8421码的方式进行二进制和十进制的相互转换
- C:二进制到八进制的简易方式
- D:二进制到十六进制的简易方式

02.07_Java语言基础(原码反码补码)(了解)

- A:为什么要学习原码反码补码?
  - 后面要学习强制类型转换,如果不知道有原反补会看不懂结果
- B:有符号数据表示法的几种方式
  - 原码
    - 就是二进制定点表示法，即最高位为符号位，“0”表示正，“1”表示负，其余位表示数值的大小。
    - 通过一个字节,也就是8个二进制位表示+7和-7
    - 0(符号位) 0000111
    - 1(符号位) 0000111
  - 反码
    - 正数的反码与其原码相同；负数的反码是对其原码逐位取反，但符号位除外。
  - 补码
    - 正数的补码与其原码相同；负数的补码是在其反码的末位加1。

02.08_Java语言基础(原码反码补码的练习)(了解)

- A:已知原码求补码
  - 0b10110100
- B:已知补码求原码
  - 0b11101110

02.09_Java语言基础(变量的概述及格式)(掌握)

- A:什么是变量
  - 在程序执行的过程中，在某个范围内其值可以发生改变的量
- B:变量的定义格式
  - 数据类型 变量名 = 变量值;
- C:为什么要定义变量
  - 用来不断的存放同一类型的常量，并可以重复使用

02.10_Java语言基础(数据类型的概述和分类)(掌握)

- A:为什么有数据类型
  - Java语言是强类型语言，对于每一种数据都定义了明确的具体数据类型，在内存中分配了不同大小的内存空间
- B:Java中数据类型的分类
  - 基本数据类型
  - 引用数据类型
    - 面向对象部分讲解
- C:基本数据类型分类(4类8种)
  - 整数型
    - byte 占一个字节 -128到127
    - short 占两个字 -2^15~2^15-1
    - int 占四个字节 -2^31~2^31-1
    - long 占八个字节 -2^63~2^63-1
  - 浮点型
    - float 占四个字节 -3.403E38~3.403E38 单精度
    - double 占八个字节-1.798E308~1.798E308 双精度
  - 字符型
    - char 占两个字节 0~65535

- 布尔型
  - boolean
    - boolean理论上是占八分之一个字节,因为一个开关就可以决定是true和false了,但是java中boolean类型没有明确指定他的大小

02.11_Java语言基础(定义不同数据类型的变量)(掌握)

- A:案例演示
  - 定义不同基本数据类型的变量，并输出
  - 赋值时候注意float类型,long类型

02.12_Java语言基础(使用变量的注意事项)(掌握)

- A:案例演示
  - a:作用域问题
    - 同一个区域不能使用相同的变量名
  - b:初始化值问题
    - 局部变量在使用之前必须赋值
  - c:一条语句可以定义几个变量
    - int a,b,c...;

02.13_Java语言基础(数据类型转换之隐式转换)(掌握)

- A:案例演示
  - a:int + int
  - b:byte + int
- B:Java中的默认转换规则
  - 取值范围小的数据类型与取值范围大的数据类型进行运算,会先将小的数据类型提升为大的,再运算
- C:画图解释byte+int类型的问题

02.14_Java语言基础(数据类型转换之强制转换)(掌握)

- A:强制转换问题
  - int a = 10;
  - byte b = 20;
  - b = a + b;
- B:强制转换的格式
  - b = (byte)(a + b);
- C:强制转换的注意事项
  - 如果超出了被赋值的数据类型的取值范围得到的结果会与你期望的结果不同

02.15_Java语言基础(面试题之变量相加和常量相加的区别)(掌握)

- A:案例演示
  - 面试题:看下面的程序是否有问题，如果有问题，请指出并说明理由。
  - byte b1 = 3;
  - byte b2 = 4;
  - byte b3 = b1 + b2;
    - 从两方面去回答这个题
    - b1和b2是两个变量,变量里面存储的值都是变化的,所以在程序运行中JVM是无法判断里面具体的值
    - byte类型的变量在进行运算的时候,会自动类型提升为int类型
  - byte b4 = 3 + 4;
    - 3和4都是常量,java有常量优化机制,就是在编译的的时候直接把3和4的结果赋值给b4了

02.16_Java语言基础(long与float的取值范围谁大谁小)(了解)

- 进行混合运算的时候,byte,short,char不会相互转换,都会自动类型提升为int类型,其他类型进行混合运算的是小的数据类型提升为大的
  - byte,short,char -- int -- long -- float -- double
  - long: 8个字节
  - float：4个字节
  - IEEE754
  - 4个字节是32个二进制位
  - 1位是符号位
  - 8位是指数位
  - 00000000 11111111
  - 0到255
  - 1到254
  - -126到127
  - 23位是尾数位

- 每个指数位减去127
- A:它们底层的存储结构不同。
- B:float表示的数据范围比long的范围要大
  - long：2^63-1
  - float：3.4*10^38 > 210^38 > 2*8^38 = 22^3^38 = 22^114 > 2^63-1

02.17_Java语言基础(字符和字符串参与运算)(掌握)

- A:案例演示
  - System.out.println('a');
  - System.out.println('a'+1);
  - 通过看结果知道'a'的值是多少,由此引出ASCII码表
- B:ASCII码表的概述
  - 记住三个值：
    - '0' 48
    - 'A' 65
    - 'a' 97
- C:案例演示
  - System.out.println("hello"+'a'+1);
  - System.out.println('a'+1+"hello");
- D:+在有字符串参与中被称为字符串连接符
  - System.out.println("5+5="+5+5);
  - System.out.println(5+5+"=5+5");

02.18_Java语言基础(char数据类型)(掌握)

- A:char c = 97; 0到65535
- B:Java语言中的字符char可以存储一个中文汉字吗?为什么呢?
  - 可以。因为Java语言采用的是Unicode编码。Unicode编码中的每个字符占用两个字节。中文也是占的两个字节
  - 所以，Java中的字符可以存储一个中文汉字

02.19_Java语言基础(算术运算符的基本用法)(掌握)

- A:什么是运算符
  - 就是对常量和变量进行操作的符号。
- B:运算符的分类
  - 算术运算符，赋值运算符，比较(关系或条件)运算符，逻辑运算符，位运算符，三目(元)运算符
- C:算数运算符有哪些
  - +,-,*,/,%,++,--
- D:注意事项：
  - a:+号在java中有三种作用,代表正号,做加法运算,字符串的连接符
  - b:整数相除只能得到整数。如果想得到小数，必须把数据变化为浮点数类型
  - c:/获取的是除法操作的商，%获取的是除法操作的余数
  - %运算符
    - 当左边的绝对值小于右边绝对值时,结果是左边
    - 当左边的绝对值等于右边或是右边的倍数时,结果是0
    - 当左边的绝对值大于右边绝对值时,结果是余数
    - %运算符结果的符号只和左边有关系,与右边无关
    - 任何一个正整数%2结果不是0就是1可以用来当作切换条件

02.20_Java语言基础(算术运算符++和--的用法)(掌握)

- A:++,--运算符的作用
  - 自加（++）自减（--）运算
  - ++:自加。对原有的数据进行+1
  - --:自减。对原有的数据进行-1
- B:案例演示
  - a:单独使用：
    - 放在操作数的前面和后面效果一样。(这种用法是我们比较常见的)
  - b:参与运算使用：
    - 放在操作数的前面，先自增或者自减，然后再参与运算。
    - 放在操作数的后面，先参与运算，再自增或者自减。

02.21_Java语言基础(算术运算符++和--的练习)(掌握)

- A:案例演示

- 请分别计算出a,b,c的值?

```
int a = 10;
int b = 10;
int c = 10;

a = b++;
c = --a;
b = ++a;
a = c--;
```

9
10
9

初学JVVA。求详细过程。。int x = 4; int y = (--x)+(x--)+(x*10);求Y

(--x)把x减1再用，3，x变成3
(x--)把x用了再减，3，x变成2
(x*10)此时x为2，
y=3+3+20

- B:案例演示
  - 请分别计算出x,y的值?

```
int x = 4;
int y = (x++)+(++x)+(x*10);
```

70

- C:面试题
  - byte b = 10;
  - b++;
  - b = b + 1;
  - 问哪句会报错,为什么

b = (byte) (b + 1);

02.22_Java语言基础(赋值运算符的基本用法)(掌握)

- A:赋值运算符有哪些
  - a:基本的赋值运算符：=
    - 把=右边的数据赋值给左边。
  - b:扩展的赋值运算符：+=,-=,*=,/=,%=
    - += 把左边和右边做加法，然后赋值给左边。

02.23_Java语言基础(赋值运算符的面试题)(掌握)

- A:案例演示
  - 面试题:看下面的程序是否有问题，如果有问题，请指出并说明理由。
  - short s=1;s = s+1;
  - short s=1;s+=1;

有问题：1为int类型

02.24_Java语言基础(关系运算符的基本用法及其注意事项)(掌握)

- A:关系运算符有哪些(比较运算符,条件运算符)
  - ==,!=,>,>=,<,<=
- 注意事项：
  - 无论你的操作是简单还是复杂，结果是boolean类型。
  - "=="不能写成"="。

02.25_day02总结

- 把今天的知识点总结一遍。

03.01_Java语言基础(逻辑运算符的基本用法)(掌握)

- A:逻辑运算符有哪些
  - &,|,^,!
  - &&,||
- B:案例演示
- 逻辑运算符的基本用法
- 注意事项：
  - a:逻辑运算符一般用于连接boolean类型的表达式或者值。
  - b:表达式：就是用运算符把常量或者变量连接起来的符合java语法的式子。
    - 算术表达式：a + b
    - 比较表达式：a == b(条件表达式)
- C:结论：
- &逻辑与:有false则false。
- |逻辑或:有true则true。
- ^逻辑异或:相同为false，不同为true。
- !逻辑非:非false则true，非true则false。
  - 特点：偶数个不改变本身。

03.02_Java语言基础(逻辑运算符&&和&的区别)(掌握)

- A:案例演示
  - &&和&的区别?
    - a:最终结果一样。
    - b:&&具有短路效果。左边是false，右边不执行。
    - &是无论左边是false还是true,右边都会执行
- B:同理||和|的区别?(学生自学)
- C:开发中常用谁?
  - &&,||,!

03.03_Java语言基础(位运算符的基本用法1)(了解)

- A:位运算符有哪些
  - &,|,^,~ ,>>,>>>,<<
- B:案例演示
  - 位运算符的基本用法1
  - &,|,^,~ 的用法
  - &:有0则0
  - |:有1则1
  - ^:相同则0，不同则1
  - ~:按位取反

03.04_Java语言基础(位异或运算符的特点及面试题)(掌握)

- A:案例演示
  - 位异或运算符的特点
  - ^的特点：一个数据对另一个数据位异或两次，该数本身不变。
- B:面试题：
  - 请自己实现两个整数变量的交换
  - 注意：以后讲课的过程中，我没有明确指定数据的类型，默认int类型。

03.05_Java语言基础(位运算符的基本用法2及面试题)(了解)

- A:案例演示 >>,>>>,<<的用法:
  - <<:左移 左边最高位丢弃，右边补齐0
  - >>:右移 最高位是0，左边补齐0;最高为是1，左边补齐1
  - >>>:无符号右移 无论最高位是0还是1，左边补齐0
  - 最有效率的算出2 * 8的结果 $2*(2<<3)$

03.06_Java语言基础(三元运算符的基本用法)(掌握)

- A:三元运算符的格式
- (关系表达式) ? 表达式1 : 表达式2;

- B:三元运算符的执行流程
- C:案例演示
  - 获取两个数中的最大值

03.07_Java语言基础(三元运算符的练习)(掌握)

- A:案例演示
  - 比较两个整数是否相同
- B:案例演示
  - 获取三个整数中的最大值

03.08_Java语言基础(键盘录入的基本格式讲解)(掌握)

- A:为什么要使用键盘录入数据
  - a:为了让程序的数据更符合开发的数据
  - b:让程序更灵活一下
- B:如何实现键盘录入呢?
  - 先照格式来。
  - a:导包
    - 格式：
      - import java.util.Scanner;
    - 位置：
      - 在class上面。
  - b:创建键盘录入对象
    - 格式：
      - Scanner sc = new Scanner(System.in);
  - c:通过对象获取数据
    - 格式：
      - int x = sc.nextInt();
- C:案例演示
  - 键盘录入1个整数，并输出到控制台。
  - 键盘录入2个整数，并输出到控制台。

03.09_Java语言基础(键盘录入的练习1)(掌握)

- A:案例演示
  - 键盘录入练习：键盘录入两个数据，并对这两个数据求和，输出其结果
- B:案例演示
  - 键盘录入练习：键盘录入两个数据，获取这两个数据中的最大值

03.10_Java语言基础(键盘录入的练习2)(掌握)

- A:案例演示
  - 键盘录入练习：键盘录入两个数据，比较这两个数据是否相等
- B:案例演示
  - 键盘录入练习：键盘录入三个数据，获取这三个数据中的最大值

03.11_Java语言基础(顺序结构语句)(了解)

- A:什么是流程控制语句
  - 流程控制语句：可以控制程序的执行流程。
- B:流程控制语句的分类
  - 顺序结构
  - 选择结构
  - 循环结构
- C:执行流程：
  - 从上往下，依次执行。
- D:案例演示
  - 输出几句话看效果即可

03.12_Java语言基础(选择结构if语句格式1及其使用)(掌握)

- A:选择结构的分类
  - if语句
  - switch语句
- B:if语句有几种格式
  - 格式1
  - 格式2
  - 格式3
- C:if语句的格式1

```
if(比较表达式) {
    语句体;
}
```

- D:执行流程：
  - 先计算比较表达式的值，看其返回值是true还是false。
  - 如果是true，就执行语句体；
  - 如果是false，就不执行语句体；

03.13_Java语言基础(选择结构if语句注意事项)(掌握)

- A:案例演示
  - a:比较表达式无论简单还是复杂，结果必须是boolean类型
  - b:if语句控制的语句体如果是一条语句，大括号可以省略；
    - 如果是多条语句，就不能省略。建议永远不要省略。
  - c:一般来说：有左大括号就没有分号，有分号就没有左大括号

03.14_Java语言基础(选择结构if语句格式2及其使用)(掌握)

- A:if语句的格式2

```
if(比较表达式) {
    语句体1;
}else {
    语句体2;
}
```

- B:执行流程：
  - 首先计算比较表达式的值，看其返回值是true还是false。
  - 如果是true，就执行语句体1；
  - 如果是false，就执行语句体2；
- C:案例演示
  - a:获取两个数据中较大的值
  - b:判断一个数据是奇数还是偶数,并输出是奇数还是偶数
  - 注意事项：else后面是没有比较表达式的，只有if后面有。

03.15_Java语言基础(if语句的格式2和三元的相互转换问题)(掌握)

- A:案例演示
  - if语句和三元运算符完成同一个效果
- B:案例演示
  - if语句和三元运算符的区别
  - 三元运算符实现的，都可以采用if语句实现。反之不成立。
  - 什么时候if语句实现不能用三元改进呢?
    - 当if语句控制的操作是一个输出语句的时候就不能。
    - 为什么呢?因为三元运算符是一个运算符，运算符操作完毕就应该有一个结果，而不是一个输出。

03.16_Java语言基础(选择结构if语句格式3及其使用)(掌握)

- A:if语句的格式3：

```
if(比较表达式1) {
    语句体1;
}else if(比较表达式2) {
    语句体2;
}else if(比较表达式3) {
    语句体3;
}
...
else {
    语句体n+1;
}
```

- B:执行流程：
  - 首先计算比较表达式1看其返回值是true还是false，
  - 如果是true，就执行语句体1，if语句结束。
  - 如果是false，接着计算比较表达式2看其返回值是true还是false，
  - 如果是true，就执行语句体2，if语句结束。

- 如果是false，接着计算比较表达式3看其返回值是true还是false，
- 如果都是false，就执行语句体n+1。

- C:注意事项:最后一个else可以省略,但是建议不要省略,可以对范围外的错误值提示

03.17_Java语言基础(选择结构if语句格式3练习)(掌握)

- A:练习1

```
需求：键盘录入一个成绩，判断并输出成绩的等级。
90-100 优
80-89  良
70-79  中
60-69  及
0-59   差
```

- B:练习2
  - 需求：
    - 键盘录入x的值，计算出y的并输出。
    - x>=3 y = 2 * x + 1;
    - -1<x<3 y = 2 * x;
    - x<=-1 y = 2 * x - 1;

03.18_Java语言基础(选择结构if语句的嵌套使用)(掌握)

- A:案例演示
  - 需求：获取三个数据中的最大值
  - if语句的嵌套使用。

03.19_Java语言基础(选择结构switch语句的格式及其解释)(掌握)

- A:switch语句的格式

```
switch(表达式) {
      case 值1:
        语句体1;
        break;
        case 值2:
        语句体2;
        break;
        …
        default:
        语句体n+1;
        break;
}
```

- B:switch语句的格式解释

- C:面试题
  - byte可以作为switch的表达式吗? 可以
  - long可以作为switch的表达式吗? 可以
  - String可以作为switch的表达式吗? 1.7后可以
- C:执行流程
  - 先计算表达式的值
  - 然后和case后面的匹配，如果有就执行对应的语句，否则执行default控制的语句

03.20_Java语言基础(选择结构switch语句的练习)(掌握)

- A:整数(给定一个值,输出对应星期几)

03.21_Java语言基础(选择结构switch语句的注意事项)(掌握)

- A:案例演示
  - a:case后面只能是常量，不能是变量，而且，多个case后面的值不能出现相同的
  - b:default可以省略吗?
    - 可以省略，但是不建议，因为它的作用是对不正确的情况给出提示。
    - 特殊情况：
      - case就可以把值固定。
      - A,B,C,D
  - c:break可以省略吗?

        - 最后一个可以省略,其他最好不要省略
        - 会出现一个现象：case穿透。
        - 最终我们建议不要省略
    - d:default一定要在最后吗?
        - 不是，可以在任意位置。但是建议在最后。
    - e:switch语句的结束条件
        - a:遇到break就结束了
        - b:执行到switch的右大括号就结束了

03.22_Java语言基础(选择结构switch语句练习)(掌握)

- A:看程序写结果：

```
int x = 2;
int y = 3;
switch(x){
    default:
        y++;
        break;
    case 3:
        y++;
    case 4:
        y++;
}
System.out.println("y="+y);
```

- B:看程序写结果：

```
int x = 2;
int y = 3;
switch(x){
    default:
        y++;
    case 3:
        y++;
    case 4:
        y++;
}
System.out.println("y="+y);
```

03.23_Java语言基础(选择结构if语句和switch语句的区别)(掌握)

- A:总结switch语句和if语句的各自使用场景
- switch建议判断固定值的时候用
- if建议判断区间或范围的时候用
- B:案例演示
    - 分别用switch语句和if语句实现下列需求：
        - 键盘录入月份，输出对应的季节

03.24_day03总结

把今天的知识点总结一遍。

04.01_Java语言基础(循环结构概述和for语句的格式及其使用)

- A:循环结构的分类
  - for,while,do...while
- B:循环结构for语句的格式：

```
for(初始化表达式;条件表达式;循环后的操作表达式) {
    循环体;
}
```

- C执行流程：
  - a:执行初始化语句
  - b:执行判断条件语句,看其返回值是true还是false
    - 如果是true，就继续执行
    - 如果是false，就结束循环
  - c:执行循环体语句;
  - d:执行循环后的操作表达式
  - e:回到B继续。
- D:案例演示
  - 在控制台输出10次"helloworld"

04.02_Java语言基础(循环结构for语句的练习之获取数据)

- A:案例演示
  - 需求：请在控制台输出数据1-10
  - 需求：请在控制台输出数据10-1
- B:注意事项
  - a:判断条件语句无论简单还是复杂结果是boolean类型。
  - b:循环体语句如果是一条语句，大括号可以省略；如果是多条语句，大括号不能省略。建议永远不要省略。
  - c:一般来说：有左大括号就没有分号，有分号就没有左大括号

04.03_Java语言基础(循环结构for语句的练习之求和思想)

- A:案例演示
  - 需求：求出1-10之间数据之和
- B:学生练习
  - 需求：求出1-100之间偶数和
  - 需求：求出1-100之间奇数和

04.04_Java语言基础(循环结构for语句的练习之水仙花)

- A:案例演示
  - 需求：在控制台输出所有的”水仙花数”
  - 所谓的水仙花数是指一个三位数，其各位数字的立方和等于该数本身。
  - 举例：153就是一个水仙花数。
  - 153 = 1*1*1 + 5*5*5 + 3*3*3 = 1 + 125 + 27 = 153

04.05_Java语言基础(循环结构for语句的练习之统计思想)

- A:案例演示
  - 需求：统计”水仙花数”共有多少个

04.06_Java语言基础(循环结构while语句的格式和基本使用)

- A:循环结构while语句的格式：

```
while循环的基本格式：
while(判断条件语句) {
    循环体语句;
}

完整格式：

初始化语句;
while(判断条件语句) {
     循环体语句;
     控制条件语句;
}
```

- B:执行流程：

    - a:执行初始化语句
    - b:执行判断条件语句,看其返回值是true还是false
        - 如果是true，就继续执行
        - 如果是false，就结束循环
    - c:执行循环体语句;
    - d:执行控制条件语句
    - e:回到B继续。
- C:案例演示
    - 需求：请在控制台输出数据1-10

04.07_Java语言基础(循环结构while语句的练习)

- A:求和思想
    - 求1-100之和
- B:统计思想
    - 统计”水仙花数”共有多少个

04.08_Java语言基础(循环结构do...while语句的格式和基本使用)

- A:循环结构do...while语句的格式：

```
do {
    循环体语句;
}while(判断条件语句);

完整格式:
初始化语句;
do {
    循环体语句;
    控制条件语句;
}while(判断条件语句);
```

- B:执行流程：
    - a:执行初始化语句
    - b:执行循环体语句;
    - c:执行控制条件语句
    - d:执行判断条件语句,看其返回值是true还是false
        - 如果是true，就继续执行
        - 如果是false，就结束循环
    - e:回到b继续。
- C:案例演示
    - 需求：请在控制台输出数据1-10

04.09_Java语言基础(循环结构三种循环语句的区别)

- A:案例演示
    - 三种循环语句的区别:
    - do...while循环至少执行一次循环体。
    - 而for,while循环必须先判断条件是否成立，然后决定是否执行循环体语句。
- B:案例演示
    - for循环和while循环的区别：
        - A:如果你想在循环结束后，继续使用控制条件的那个变量，用while循环，否则用for循环。不知道用谁就用for循环。因为变量及早的从内存中消失，可以提高内存的使用效率。

04.10_Java语言基础(循环结构注意事项之死循环)

- A:一定要注意控制条件语句控制的那个变量的问题，不要弄丢了，否则就容易死循环。
- B:两种最简单的死循环格式
    - while(true){...}
    - for(;;){...}

04.11_Java语言基础(循环结构循环嵌套输出4行5列的星星)

- A:案例演示
    - 需求：请输出一个4行5列的星星(*)图案。

```
如图：
    *****
    *****
    *****
    *****

注意：
```

```
    System.out.println("*");和System.out.print("*");的区别
```

- B:结论：
  - 外循环控制行数，内循环控制列数

04.12_Java语言基础(循环结构循环嵌套输出正三角形)

- A:案例演示

```
需求：请输出下列的形状
*
**
***
****
*****
```

04.13_Java语言基础(循环结构九九乘法表)

- A:案例演示
  - 需求：在控制台输出九九乘法表。
- B:代码优化

```
注意：
'\x' x表示任意，\是转义符号,这种做法叫转移字符。

'\t'    tab键的位置
'\r'    回车
'\n'    换行
'\"'
'\''
```

04.14_Java语言基础(控制跳转语句break语句)

- A:break的使用场景
  - 只能在switch和循环中

04.15_Java语言基础(控制跳转语句continue语句)

- A:continue的使用场景
  - 只能在循环中

04.16_Java语言基础(控制跳转语句标号)

- 标号:标记某个循环对其控制
- 标号组成规则:其实就是合法的标识符

04.17_Java语言基础(控制调整语句练习)

- A:练习题

```
for(int x=1; x<=10; x++) {
    if(x%3==0) {
        //在此处填写代码
    }
    System.out.println(“Java基础班”);
}

我想在控制台输出2次:“Java基础班“
我想在控制台输出7次:“Java基础班“
我想在控制台输出13次:“Java基础班“
```

04.18_Java语言基础(控制跳转语句return语句)

- A:return的作用
  - 返回
  - 其实它的作用不是结束循环的，而是结束方法的。
- B:案例演示
  - return和break以及continue的区别?
  - return是结束方法
  - break是跳出循环
  - continue是终止本次循环继续下次循环

04.19_Java语言基础(方法概述和格式说明)

- A:为什么要有方法

    - 提高代码的复用性
- B:什么是方法
    - 完成特定功能的代码块。
- C:方法的格式

```
修饰符 返回值类型 方法名(参数类型 参数名1,参数类型 参数名2...) {
    方法体语句;
    return 返回值;
}
```

- D:方法的格式说明
    - 修饰符：目前就用 public static。后面我们再详细的讲解其他的修饰符。
    - 返回值类型：就是功能结果的数据类型。
    - 方法名：符合命名规则即可。方便我们的调用。
    - 参数：
        - 实际参数：就是实际参与运算的。
        - 形式参数：就是方法定义上的，用于接收实际参数的。
    - 参数类型：就是参数的数据类型
    - 参数名：就是变量名
    - 方法体语句：就是完成功能的代码。
    - return：结束方法的。
    - 返回值：就是功能的结果，由return带给调用者。

04.20_Java语言基础(方法之求和案例及其调用)

- A:如何写一个方法
    - 1,明确返回值类型
    - 2,明确参数列表
- B:案例演示
    - 需求：求两个数据之和的案例
- C:方法调用图解

04.21_Java语言基础(方法的注意事项)

- A:方法调用(有具体返回值)
    - a:单独调用,一般来说没有意义，所以不推荐。
    - b:输出调用,但是不够好。因为我们可能需要针对结果进行进一步的操作。
    - c:赋值调用,推荐方案。
- B:案例演示
    - a:方法不调用不执行
    - b:方法与方法是平级关系，不能嵌套定义
    - c:方法定义的时候参数之间用逗号隔开
    - d:方法调用的时候不用在传递数据类型
    - e:如果方法有明确的返回值，一定要有return带回一个值

04.22_Java语言基础(方法的练习)

- A:案例演示
    - 需求：键盘录入两个数据，返回两个数中的较大值
- B:案例演示
    - 需求：键盘录入两个数据，比较两个数是否相等

04.23_Java语言基础(方法之输出星形及其调用)

- A:案例演示
    - 需求：根据键盘录入的行数和列数，在控制台输出星形
- B:方法调用：(无返回值,void)
    - 单独调用
    - 输出调用(错误)
    - 赋值调用(错误)

04.24_Java语言基础(方法的练习)

- A:案例演示
    - 需求：根据键盘录入的数据输出对应的乘法表

04.25_Java语言基础(方法重载概述和基本使用)

- A:方法重载概述
    - 求和案例
        - 2个整数

        - 3个整数
        - 4个整数
- B:方法重载：
    - 在同一个类中，方法名相同，参数列表不同。与返回值类型无关。
    - 参数列表不同：
        - A:参数个数不同
        - B:参数类型不同
        - C:参数的顺序不同(算重载,但是在开发中不用)

04.26_Java语言基础(方法重载练习比较数据是否相等)

- A:案例演示
    - 需求：比较两个数据是否相等。
    - 参数类型分别为两个int类型，两个double类型，并在main方法中进行测试

04.27_day04总结

把今天的知识点总结一遍。

05.01_Java语言基础(数组概述和定义格式说明)(了解)

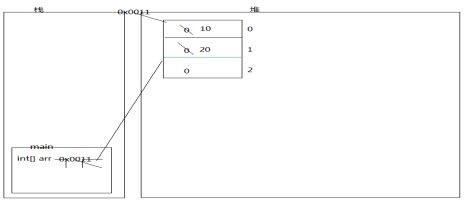
- A:为什么要有数组(容器)
  - 为了存储同种数据类型的多个值
- B:数组概念
  - 数组是存储同一种数据类型多个元素的集合。也可以看成是一个容器。
  - 数组既可以存储基本数据类型，也可以存储引用数据类型。
- C:数组定义格式 数据类型[] 数组名 = new 数据类型[数组的长度];

05.02_Java语言基础(数组的初始化动态初始化)(掌握)

- A:什么是数组的初始化
  - 就是为数组开辟连续的内存空间，并为每个数组元素赋予值
- B:如何对数组进行初始化
  - a:动态初始化 只指定长度，由系统给出初始化值
    - int[] arr = new int[5];
  - b:静态初始化 给出初始化值，由系统决定长度
- C:动态初始化的格式：
  - 数据类型[] 数组名 = new 数据类型[数组长度];
- D:案例演示
  - 输出数组名称和数组元素

元素初始化为零

```
class Demo3_Array {
	public static void main(String[] args) {
		int[] arr = new int[3];
		System.out.println(arr);
		arr[0] = 10;
		arr[1] = 20;

		System.out.println(arr[0]);
		System.out.println(arr[1]);
	}
}
```



05.03_Java语言基础(Java中的内存分配以及栈和堆的区别)

- A:栈(掌握)
  - 存储局部变量
- B:堆(掌握)
  - 存储new出来的数组或对象
- C:方法区
  - 面向对象部分讲解
- D:本地方法区
  - 和系统相关
- E:寄存器
  - 给CPU使用

05.04_Java语言基础(数组的内存图解1一个数组)(掌握)

- A:画图演示
  - 一个数组

05.05_Java语言基础(数组的内存图解2二个数组)(了解)

- A:画图演示
  - 二个不同的数组

05.06_Java语言基础(数组的内存图解3三个引用两个数组)(了解)

- A:画图演示
  - 三个引用，有两个数组的引用指向同一个地址

05.07_Java语言基础(数组的初始化静态初始化及内存图)(掌握)

- A:静态初始化的格式：
  - 格式：数据类型[] 数组名 = new 数据类型[]{元素1,元素2,…};
  - 简化格式：
    - 数据类型[] 数组名 = {元素1,元素2,…};
- B:案例演示
  - 对数组的解释
  - 输出数组名称和数组元素
- C:画图演示
  - 一个数组

05.08_Java语言基础(数组操作的两个常见小问题越界和空指针)(掌握)

- A:案例演示
  - a:ArrayIndexOutOfBoundsException:数组索引越界异常
    - 原因：你访问了不存在的索引。
  - b:NullPointerException:空指针异常
    - 原因：数组已经不在指向堆内存了。而你还用数组名去访问元素。

- int[] arr = {1,2,3};
- arr = null;
- System.out.println(arr[0]);

05.09_Java语言基础(数组的操作1遍历)(掌握)

- A:案例演示
  - 数组遍历：就是依次输出数组中的每一个元素。
  - 数组的属性:arr.length数组的长度
  - 数组的最大索引:arr.length - 1;

```
public static void print(int[] arr) {
    for (int i = 0;i < arr.length ;i++ ) {
        System.out.print(arr[i] + " ");
    }
}
```

05.10_Java语言基础(数组的操作2获取最值)(掌握)

- A:案例演示
  - 数组获取最值(获取数组中的最大值最小值)

```
public static int getMax(int[] arr) {
    int max = arr[0];
    for (int i = 1;i < arr.length ;i++ ) {          //从数组的第二个元素开始遍历
        if (max < arr[i]) {                         //如果max记录的值小于的数组中的元素
            max = arr[i];                           //max记录住较大的
        }
    }

    return max;
}
```

05.11_Java语言基础(数组的操作3反转)(掌握)

- A:案例演示
  - 数组元素反转(就是把元素对调)

```
public static void reverseArray(int[] arr) {
    for (int i = 0;i < arr.length / 2 ; i++) {
        //arr[0]和arr[arr.length-1-0]交换
        //arr[1]和arr[arr.length-1-1]交换
        //arr[2]和arr[arr.lentth-1-2]
        //...

        int temp = arr[i];
        arr[i] = arr[arr.length-1-i];
        arr[arr.length-1-i] = temp;
    }
}
```

05.12_Java语言基础(数组的操作4查表法)(掌握)

- A:案例演示
  - 数组查表法(根据键盘录入索引,查找对应星期)

```
public static char getWeek(int week) {
    char[] arr = {' ','一','二','三','四','五','六','日'};     //定义了一张星期表
    return arr[week];                                          //通过索引获取表中的元素
}
```

05.13_Java语言基础(数组的操作5基本查找)(掌握)

- A:案例演示
  - 数组元素查找(查找指定元素第一次在数组中出现的索引)

```
public static int getIndex(int[] arr,int value) {
    for (int i = 0;i < arr.length ;i++ ) {              //数组的遍历
        if (arr[i] == value) {                          //如果数组中的元素与查找的元素匹配
            return i;
```

```
        }
    }
    return -1;
}
```

05.14_Java语言基础(二维数组概述和格式1的讲解)(了解)

- A:二维数组概述
- B:二维数组格式1
  - int[][] arr = new int[3][2];
- C:二维数组格式1的解释
- D:注意事项
  - a:以下格式也可以表示二维数组
    - 1:数据类型 数组名[][] = new 数据类型[m][n];
    - 2:数据类型[] 数组名[] = new 数据类型[m][n];
  - B:注意下面定义的区别

```
int x;
int y;
int x,y;

int[] x;
int[] y[];

int[] x,y[];    x是一维数组,y是二维数组
```

- E:案例演示
  - 定义二维数组，输出二维数组名称，一维数组名称，一个元素

05.15_Java语言基础(二维数组格式1的内存图解)(了解)

- A:画图演示
  - 画图讲解上面的二维数组名称，一维数组名称，一个元素的值的问题

05.16_Java语言基础(二维数组格式2的讲解及其内存图解)(了解)

- A:二维数组格式2
  - int[][] arr = new int[3][];
- B:二维数组格式2的解释
- C:案例演示
  - 讲解格式，输出数据，并画内存图

05.17_Java语言基础(二维数组格式3的讲解及其内存图解)(了解)

- A:二维数组格式3
  - int[][] arr = {{1,2,3},{4,5},{6,7,8,9}};
- B:二维数组格式3的解释
- C:案例演示
  - 讲解格式，输出数据，并画内存图

05.18_Java语言基础(二维数组练习1遍历)(掌握)

- A:案例演示
  - 需求：二维数组遍历
  - 外循环控制的是二维数组的长度，其实就是一维数组的个数。
  - 内循环控制的是一维数组的长度。

```
int[][] arr = {{1,2,3},{4,5},{6,7,8,9}};

for (int i = 0;i < arr.length ;i++ ) {          //获取到每个二维数组中的一维数组
    for (int j = 0;j < arr[i].length ;j++ ) {   //获取每个一维数组中的元素
        System.out.print(arr[i][j] + " ");
    }

    System.out.println();
}
```

05.19_Java语言基础(二维数组练习2求和)(掌握)

- A:案例演示

```
需求：公司年销售额求和
某公司按照季度和月份统计的数据如下：单位(万元)
第一季度：22,66,44
第二季度：77,33,88
第三季度：25,45,65
第四季度：11,66,99

int[][] arr = {{22,66,44},{77,33,88},{25,45,65},{11,66,99}};

int sum = 0;                                    //定义变量,记录每次相加的结果
for (int i = 0;i < arr.length ;i++ ) {          //获取每一个一维数组
    for (int j = 0;j < arr[i].length ;j++ ) {   //获取每一个一维数组中的元素
        sum = sum + arr[i][j];                  //累加
    }
}

System.out.println(sum);
```

05.20_Java语言基础(思考题Java中的参数传递问题及图解)(掌握)

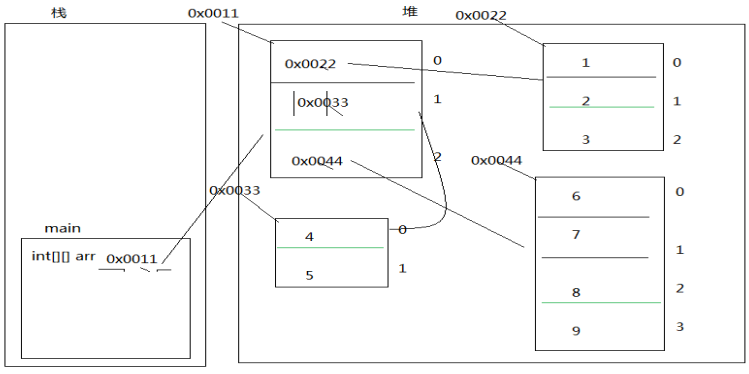
- A:案例演示

```
看程序写结果，并画内存图解释
public static void main(String[] args) {
    int a = 10;
    int b = 20;
    System.out.println("a:"+a+",b:"+b);
    change(a,b);
    System.out.println("a:"+a+",b:"+b);

    int[] arr = {1,2,3,4,5};
    change(arr);
    System.out.println(arr[1]);
}

public static void change(int a,int b) {
    System.out.println("a:"+a+",b:"+b);
    a = b;
    b = a + b;
    System.out.println("a:"+a+",b:"+b);
}

public static void change(int[] arr) {
    for(int x=0; x<arr.length; x++) {
        if(arr[x]%2==0) {
            arr[x]*=2;
        }
    }
}
```

05.21_day05总结

- 把今天的知识点总结一遍。

```
class Demo4_Array {
    public static void main(String[] args) {
        int[][] arr = {{1,2,3},{4,5},{6,7,8,9}};
        System.out.println(arr);
        System.out.println(arr[0]);
        System.out.println(arr[0][0]);
    }
}
```



06.01_面向对象(面向对象思想概述)(了解)

- A:面向过程思想概述
  - 第一步
  - 第二步
- B:面向对象思想概述
  - 找对象(第一步,第二步)
- C:举例
  - 买煎饼果子
  - 洗衣服
- D:面向对象思想特点
  - a:是一种更符合我们思想习惯的思想
  - b:可以将复杂的事情简单化
  - c:将我们从执行者变成了指挥者
    - 角色发生了转换
- E:面向对象开发
  - 就是不断的创建对象，使用对象，指挥对象做事情。
- F:面向对象设计
  - 其实就是在管理和维护对象之间的关系。
- G:面向对象特征
  - 封装(encapsulation)
  - 继承(inheritance)
  - 多态(polymorphism)

06.02_面向对象(类与对象概述)(掌握)

- A:我们学习编程是为了什么
  - 为了把我们日常生活中实物用学习语言描述出来
- B:我们如何描述现实世界事物
  - 属性 就是该事物的描述信息(事物身上的名词)
  - 行为 就是该事物能够做什么(事物身上的动词)
- C:Java中最基本的单位是类,Java中用class描述事物也是如此
  - 成员变量 就是事物的属性
  - 成员方法 就是事物的行为
- D:定义类其实就是定义类的成员(成员变量和成员方法)
  - a:成员变量 和以前定义变量是一样的，只不过位置发生了改变。在类中，方法外。
  - b:成员方法 和以前定义方法是一样的，只不过把static去掉，后面在详细讲解static的作用。
- E:类和对象的概念
  - a:类：是一组相关的属性和行为的集合
  - b:对象：是该类事物的具体体现
  - c:举例：
    - 类 学生
    - 对象 具体的某个学生就是一个对象

06.03_面向对象(学生类的定义)(掌握)

- A:学生事物
- B:学生类
- C:案例演示
  - 属性:姓名,年龄,性别
  - 行为:学习,睡觉

06.04_面向对象(手机类的定义)(掌握)

- 模仿学生类，让学生自己完成
  - 属性:品牌(brand)价格(price)
  - 行为:打电话(call),发信息(sendMessage)玩游戏(playGame)

06.05_面向对象(学生类的使用)(掌握)

- A:文件名问题
  - 在一个java文件中写两个类：一个基本的类，一个测试类。
  - 建议：文件名称和测试类名称一致。
- B:如何使用对象?
  - 创建对象并使用
  - 格式：类名 对象名 = new 类名();
- D:如何使用成员变量呢?
  - 对象名.变量名
- E:如何使用成员方法呢?

- 对象名.方法名(...)

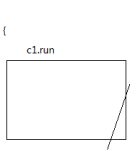
06.06_面向对象(手机类的使用)(掌握)

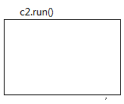
- A:学生自己完成
  - 模仿学生类，让学生自己完成

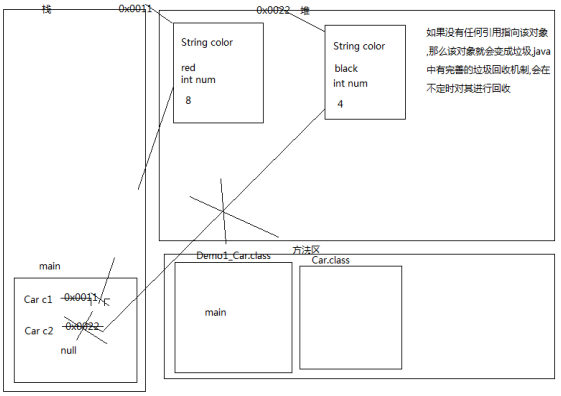
06.07_面向对象(一个对象的内存图)(掌握)

- A:画图演示
  - 一个对象

```
class Demo1_Car {
	public static void main(String[] args) {
		Car c1 = new Car();
		c1.color = "red";
		c1.num = 8;
		c1.run();
		Car c2 = new Car();
		c2.color = "black";
		c2.num = 4;
		c2.run();
		c2 = null;
		c2.run();
	}
}
class Car {
	String color;
	int num;
	public void run() {
		System.out.println(color + "..." + num);
	}
}
```







06.08_面向对象(二个对象的内存图)(了解)

- A:画图演示
  - 二个不同的对象

06.09_面向对象(三个引用两个对象的内存图

- A:画图演示
  - 三个引用，有两个对象的引用指向同一个地址

06.10_面向对象(成员变量和局部变量的区别)(掌握)

- A:在类中的位置不同
  - 成员变量：在类中方法外
  - 局部变量：在方法定义中或者方法声明上
- B:在内存中的位置不同
  - 成员变量：在堆内存(成员变量属于对象,对象进堆内存)
  - 局部变量：在栈内存(局部变量属于方法,方法进栈内存)
- C:生命周期不同
  - 成员变量：随着对象的创建而存在，随着对象的消失而消失
  - 局部变量：随着方法的调用而存在，随着方法的调用完毕而消失
- D:初始化值不同
  - 成员变量：有默认初始化值
  - 局部变量：没有默认初始化值，必须定义，赋值，然后才能使用。
- 注意事项：
  - 局部变量名称可以和成员变量名称一样，在方法中使用的时候，采用的是就近原则。
  - 基本数据类型变量包括哪些:byte,short,int,long,float,double,boolean,char
  - 引用数据类型变量包括哪些:数组,类,接口,枚举

06.11_面向对象(方法的形式参数是类名的时候如何调用)(掌握)

- A:方法的参数是类名public void print(Student s){}//print(new Student());
  - 如果你看到了一个方法的形式参数是一个类类型(引用类型)，这里其实需要的是该类的对象。

06.12_面向对象(匿名对象的概述和应用)(掌握)

- A:什么是匿名对象
  - 没有名字的对象
- B:匿名对象应用场景
  - a:调用方法，仅仅只调用一次的时候。
    - 那么，这种匿名调用有什么好处吗?
      - 节省代码
    - 注意：调用多次的时候，不适合。匿名对象调用完毕就是垃圾。可以被垃圾回收器回收。
  - b:匿名对象可以作为实际参数传递
- C:案例演示
  匿名对象应用场景

06.13_面向对象(封装的概述)(掌握)

- A:封装概述
  - 是指隐藏对象的属性和实现细节，仅对外提供公共访问方式。
- B:封装好处
  - 隐藏实现细节，提供公共的访问方式
  - 提高了代码的复用性
  - 提高安全性。
- C:封装原则

- 将不需要对外提供的内容都隐藏起来。
- 把属性隐藏，提供公共方法对其访问。

06.14_面向对象(private关键字的概述和特点)(掌握)

- A:人类赋值年龄的问题
- B:private关键字特点
  - a:是一个权限修饰符
  - b:可以修饰成员变量和成员方法
  - c:被其修饰的成员只能在本类中被访问
- C:案例演示
  - 封装和private的应用：
  - A:把成员变量用private修饰
  - B:提供对应的getXxx()和setXxx()方法
  - private仅仅是封装的一种体现形式,不能说封装就是私有

06.15_面向对象(this关键字的概述和应用)(掌握)

- A:this关键字特点
  - 代表当前对象的引用
- B:案例演示
  - this的应用场景
  - 用来区分成员变量和局部变量重名

06.16_面向对象(手机类代码及其测试)(掌握)

- A:学生练习
  - 请把手机类写成一个标准类，然后创建对象测试功能。

```
class Demo2_Phone {
    public static void main(String[] args) {
        Phone p1 = new Phone();
        p1.setBrand("三星");
        p1.setPrice(5288);

        System.out.println(p1.getBrand() + "..." + p1.getPrice());
        p1.call();
        p1.sendMessage();
        p1.playGame();
    }
}
/*
手机类
    属性:品牌brand,价格price
    行为:打电话call,发短信sendMessage,玩游戏,playGame
*/
class Phone {                               //java bean
    private String brand;                   //品牌
    private int price;                      //价格

    public void setBrand(String brand) {    //设置品牌
        this.brand = brand;
    }

    public String getBrand() {              //获取品牌
        return this.brand;                  //this.可以省略,你不加系统会默认给你加
    }

    public void setPrice(int price) {       //设置价格
        this.price = price;
    }

    public int getPrice() {                 //获取价格
        return price;
    }

    public void call() {                    //打电话
        System.out.println("打电话");
    }

    public void sendMessage() {             //发短信
        System.out.println("发短信");
    }

    public void playGame() {                //玩游戏
```

```
        System.out.println("玩游戏");
    }
}
```

06.17_day06总结

- 把今天的知识点总结一遍。

07.01_面向对象(构造方法Constructor概述和格式)(掌握)

- A:构造方法概述和作用
  - 给对象的数据(属性)进行初始化
- B:构造方法格式特点
  - a:方法名与类名相同(大小也要与类名一致)
  - b:没有返回值类型，连void都没有
  - c:没有具体的返回值return;

07.02_面向对象(构造方法的重载及注意事项)(掌握)

- A:案例演示
  - 构造方法的重载
  - 重载:方法名相同,与返回值类型无关(构造方法没有返回值),只看参数列表
- B:构造方法注意事项
  - a:如果我们没有给出构造方法，系统将自动提供一个无参构造方法。
  - b:如果我们给出了构造方法，系统将不再提供默认的无参构造方法。
    - 注意：这个时候，如果我们还想使用无参构造方法，就必须自己给出。建议永远自己给出无参构造方法

07.03_面向对象(给成员变量赋值的两种方式的区别)

- A:setXxx()方法
  - 修改属性值
- B:构造方法
  - 给对象中属性进行初始化

07.04_面向对象(学生类的代码及测试)(掌握)

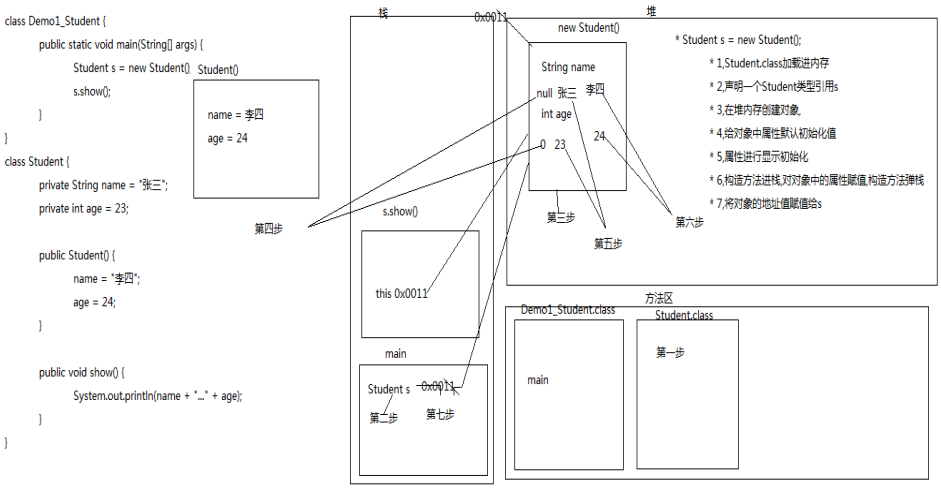
- A:案例演示
  - 学生类：
    - 成员变量：
      - name，age
    - 构造方法：
      - 无参，带两个参
    - 成员方法：
      - getXxx()/setXxx()
      - show()：输出该类的所有成员变量值
- B:给成员变量赋值：
  - a:setXxx()方法
  - b:构造方法
- C:输出成员变量值的方式：
  - a:通过getXxx()分别获取然后拼接
  - b:通过调用show()方法搞定

07.05_面向对象(手机类的代码及测试)(掌握)

- A:案例演示
  - 模仿学生类，完成手机类代码

07.06_面向对象(创建一个对象的步骤)(掌握)

- A:画图演示
  - 画图说明一个对象的创建过程做了哪些事情?
  - Student s = new Student();
  - 1,Student.class加载进内存
  - 2,声明一个Student类型引用s
  - 3,在堆内存创建对象,
  - 4,给对象中属性默认初始化值
  - 5,属性进行显示初始化
  - 6,构造方法进栈,对对象中的属性赋值,构造方法弹栈
  - 7,将对象的地址值赋值给s



07.07_面向对象(长方形案例练习)(掌握)

- A:案例演示
  - 需求：
    - 定义一个长方形类,定义 求周长和面积的方法，
    - 然后定义一个测试类进行测试。

07.08_面向对象(员工类案例练习)(掌握)

- A:案例演示
  - 需求：定义一个员工类Employee
  - 自己分析出几个成员，然后给出成员变量
    - 姓名name,工号id,工资salary
  - 构造方法，
    - 空参和有参的
  - getXxx()setXxx()方法，
  - 以及一个显示所有成员信息的方法。并测试。
    - work

07.09_面向对象(static关键字及内存图)(了解)

- A:案例演示
  - 通过一个案例引入static关键字。
  - 人类：Person。每个人都有国籍，中国。
- B:画图演示
  - 带有static的内存图

07.10_面向对象(static关键字的特点)(掌握)

- A:static关键字的特点
  - a:随着类的加载而加载
  - b:优先于对象存在
  - c:被类的所有对象共享
    - 举例：咱们班级的学生应该共用同一个班级编号。
    - 其实这个特点也是在告诉我们什么时候使用静态?
      - 如果某个成员变量是被所有对象共享的，那么它就应该定义为静态的。
    - 举例：
      - 饮水机(用静态修饰)
      - 水杯(不能用静态修饰)
      - 共性用静态,特性用非静态
  - d:可以通过类名调用
    - 其实它本身也可以通过对象名调用。
    - 推荐使用类名调用。
    - 静态修饰的内容一般我们称其为：与类相关的，类成员
- B:案例演示
  - static关键字的特点

07.11_面向对象(static的注意事项)(掌握)

- A:static的注意事项
  - a:在静态方法中是没有this关键字的
    - 如何理解呢?
      - 静态是随着类的加载而加载，this是随着对象的创建而存在。
      - 静态比对象先存在。
  - b:静态方法只能访问静态的成员变量和静态的成员方法
    - 静态方法：
      - 成员变量：只能访问静态变量
      - 成员方法：只能访问静态成员方法
    - 非静态方法：
      - 成员变量：可以是静态的，也可以是非静态的
      - 成员方法：可是是静态的成员方法，也可以是非静态的成员方法。
    - 简单记：
      - 静态只能访问静态。
- B:案例演示
  - static的注意事项

07.12_面向对象(静态变量和成员变量的区别)(掌握)

- 静态变量也叫类变量 成员变量也叫对象变量
- A:所属不同
  - 静态变量属于类，所以也称为为类变量
  - 成员变量属于对象，所以也称为实例变量(对象变量)
- B:内存中位置不同
  - 静态变量存储于方法区的静态区
  - 成员变量存储于堆内存

- C:内存出现时间不同
  - 静态变量随着类的加载而加载，随着类的消失而消失
  - 成员变量随着对象的创建而存在，随着对象的消失而消失
- D:调用不同
  - 静态变量可以通过类名调用，也可以通过对象调用
  - 成员变量只能通过对 象名调用

07.13_面向对象(main方法的格式详细解释)(了解)

- A:格式
  - public static void main(String[] args) {}
- B:针对格式的解释
  - public 被jvm调用，访问权限足够大。
  - static 被jvm调用，不用创建对象，直接类名访问
  - void被jvm调用，不需要给jvm返回值
  - main 一个通用的名称，虽然不是关键字，但是被jvm识别
  - String[] args 以前用于接收键盘录入的
- C:演示案例
  - 通过args接收键盘例如数据

07.14_面向对象(工具类中使用静态)(了解)

- A:制作一个工具类
  - ArrayTool
  - 1,获取最大值
  - 2,数组的遍历
  - 3,数组的反转

07.15_面向对象(说明书的制作过程)(了解)

- A:对工具类加入文档注释
- B:通过javadoc命令生成说明书
  - @author(提取作者内容)
  - @version(提取版本内容)
  - javadoc -d 指定的文件目录 -author -version ArrayTool.java
  - @param 参数名称//形式参数的变量名称@return 函数运行完返回的数据

07.16_面向对象(如何使用JDK提供的帮助文档)(了解)

- A:找到文档，打开文档
- B:点击显示，找到索引，出现输入框
- C:你应该知道你找谁?举例：Scanner
- D:看这个类的结构(需不需要导包)
  - 成员变量 字段
  - 构造方法 构造方法
  - 成员方法 方法

07.17_面向对象(学习Math类的随机数功能)(了解)

- 打开JDK提供的帮助文档学习
- A:Math类概述
  - 类包含用于执行基本数学运算的方法
- B:Math类特点
  - 由于Math类在java.lang包下，所以不需要导包。
  - 因为它的成员全部是静态的,所以私有了构造方法
- C:获取随机数的方法
  - public static double random():返回带正号的 double 值，该值大于等于 0.0 且小于 1.0。
- D:我要获取一个1-100之间的随机数，肿么办?
  - int number = (int)(Math.random()*100)+1;

07.18_面向对象(猜数字小游戏案例)(了解)

- A:案例演示
  - 需求：猜数字小游戏(数据在1-100之间)

07.19_day07总结

把今天的知识点总结一遍。

08.01_面向对象(代码块的概述和分类)(了解)(面试的时候会问,开发不用或者很少用)

- A:代码块概述
  - 在Java中，使用{}括起来的代码被称为代码块。
- B:代码块分类
  - 根据其位置和声明的不同，可以分为局部代码块，构造代码块，静态代码块，同步代码块(多线程讲解)。
- C:常见代码块的应用
  - a:局部代码块
    - 在方法中出现；限定变量生命周期，及早释放，提高内存利用率
  - b:构造代码块 (初始化块)
    - 在类中方法外出现；多个构造方法方法中相同的代码存放到一起，每次调用构造都执行，并且在构造方法前执行
  - c:静态代码块
    - 在类中方法外出现，并加上static修饰；用于给类进行初始化，在加载的时候就执行，并且只执行一次。
    - 一般用于加载驱动

08.02_面向对象(代码块的面试题)(掌握)

- A:看程序写结果

```
class Student {
    static {
        System.out.println("Student 静态代码块");
    }

    {
        System.out.println("Student 构造代码块");
    }

    public Student() {
        System.out.println("Student 构造方法");
    }
}

class Demo2_Student {
    static {
        System.out.println("Demo2_Student静态代码块");
    }

    public static void main(String[] args) {
        System.out.println("我是main方法");

        Student s1 = new Student();
        Student s2 = new Student();
    }
}
```

```
---------- 运行Java ----------
Demo2_Student静态代码块
我是main方法
Student 静态代码块
Student 构造代码块
Student 构造方法
Student 构造代码块
Student 构造方法

输出完成 (耗时 2 秒) - 正常终止
```

08.03_面向对象(继承案例演示)(掌握)

- A:继承(extends)
  - 让类与类之间产生关系,子父类关系
- B:继承案例演示：
  - 动物类,猫类,狗类
  - 定义两个属性(颜色,腿的个数)两个功能(吃饭，睡觉)
- C:案例演示
  - 使用继承前
- D:案例演示
  - 使用继承后

08.04_面向对象(继承的好处和弊端)(掌握)

- A:继承的好处
  - a:提高了代码的复用性
  - b:提高了代码的维护性
  - c:让类与类之间产生了关系，是多态的前提
- B:继承的弊端
  - 类的耦合性增强了。
  - 开发的原则：高内聚，低耦合。
  - 耦合：类与类的关系
  - 内聚：就是自己完成某件事情的能力

08.05_面向对象(Java中类的继承特点)(掌握)

- A:Java中类的继承特点
  - a:Java只支持单继承，不支持多继承。(一个儿子只能有一个爹)
    - 有些语言是支持多继承，格式：extends 类1,类2,...
  - b:Java支持多层继承(继承体系)
- B:案例演示
  - Java中类的继承特点
    - 如果想用这个体系的所有功能用最底层的类创建对象
    - 如果想看这个体系的共性功能,看最顶层的类

08.06_面向对象(继承的注意事项和什么时候使用继承)(掌握)

- A:继承的注意事项
  - a:子类只能继承父类所有非私有的成员(成员方法和成员变量)
  - b:子类不能继承父类的构造方法，但是可以通过super(马上讲)关键字去访问父类构造方法。
  - c:不要为了部分功能而去继承
  - 项目经理 姓名 工号 工资 奖金
  - 程序员 姓名 工号 工资
- B:什么时候使用继承

  - 继承其实体现的是一种关系："is a"。 Person Student Teacher 水果 苹果 香蕉 橘子

  采用假设法。 如果有两个类A,B。只有他们符合A是B的一种，或者B是A的一种，就可以考虑使用继承。

08.07_面向对象(继承中成员变量的关系)(掌握)

- A:案例演示
  - a:不同名的变量
  - b:同名的变量

08.08_面向对象(this和super的区别和应用)(掌握)

- A:this和super都代表什么
  - this:代表当前对象的引用,谁来调用我,我就代表谁
  - super:代表当前对象父类的引用
- B:this和super的使用区别
  - a:调用成员变量
    - this.成员变量 调用本类的成员变量,也可以调用父类的成员变量
    - super.成员变量 调用父类的成员变量
  - b:调用构造方法
    - this(...) 调用本类的构造方法
    - super(...) 调用父类的构造方法
  - c:调用成员方法
    - this.成员方法 调用本类的成员方法,也可以调用父类的方法
    - super.成员方法 调用父类的成员方法

08.09_面向对象(继承中构造方法的关系)(掌握)

- A:案例演示
  - 子类中所有的构造方法默认都会访问父类中空参数的构造方法
- B:为什么呢?

  - 因为子类会继承父类中的数据，可能还会使用父类的数据。
  - 所以，子类初始化之前，一定要先完成父类数据的初始化。

  - 其实：

    - 每一个构造方法的第一条语句默认都是：super() Object类最顶层的父类。

08.10_面向对象(继承中构造方法的注意事项)(掌握)

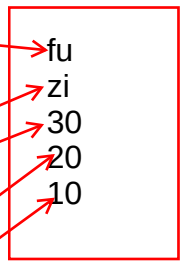
- A:案例演示
  - 父类没有无参构造方法,子类怎么办?
  - super解决
  - this解决
- B:注意事项
  - super(…)或者this(….)必须出现在构造方法的第一条语句上

08.11_面向对象(继承中的面试题)(掌握)

- A:案例演示

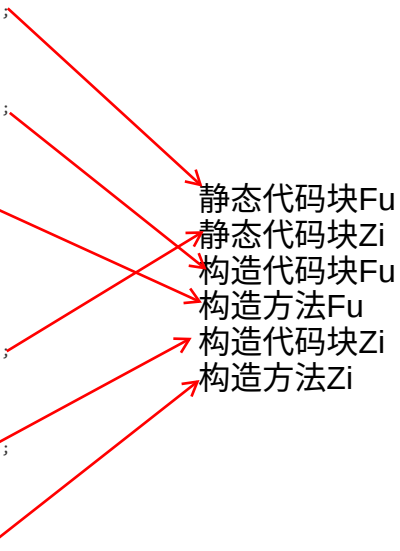
看程序写结果1

```
class Fu{
    public int num = 10;
    public Fu(){
        System.out.println("fu");
    }
}
class Zi extends Fu{
    public int num = 20;
    public Zi(){
        System.out.println("zi");
    }
    public void show(){
        int num = 30;
        System.out.println(num);
        System.out.println(this.num);
        System.out.println(super.num);
    }
}
class Test1_Extends {
    public static void main(String[] args) {
        Zi z = new Zi();
        z.show();
    }
}
```



看程序写结果2

```
class Fu {
    static {
        System.out.println("静态代码块Fu");
    }

    {
        System.out.println("构造代码块Fu");
    }

    public Fu() {
        System.out.println("构造方法Fu");
    }
}

class Zi extends Fu {
    static {
        System.out.println("静态代码块Zi");
    }

    {
        System.out.println("构造代码块Zi");
    }

    public Zi() {
        System.out.println("构造方法Zi");
    }
}

Zi z = new Zi(); 请执行结果。
```



08.12_面向对象(继承中成员方法关系)(掌握)

- A:案例演示
  - a:不同名的方法
  - b:同名的方法

08.13_面向对象(方法重写概述及其应用)(掌握)

- A:什么是方法重写
  - 重写:子父类出现了一模一样的方法(注意:返回值类型可以是子父类,这个我们学完面向对象讲)
- B:方法重写的应用:
  - 当子类需要父类的功能，而功能主体子类有自己特有内容时，可以重写父类中的方法。这样，即沿袭了父类的功能，又定义了子类特有的内容。
- C:案例演示
  - a:定义一个手机类。

08.14_面向对象(方法重写的注意事项)(掌握)

- A:方法重写注意事项
  - a:父类中私有方法不能被重写

        - 因为父类私有方法子类根本就无法继承
    - b:子类重写父类方法时，访问权限不能更低
        - 最好就一致
    - c:父类静态方法，子类也必须通过静态方法进行重写
        - 其实这个算不上方法重写，但是现象确实如此，至于为什么算不上方法重写，多态中我会讲解(静态只能覆盖静态)
    - 子类重写父类方法的时候，最好声明一模一样。

- B:案例演示
    - 方法重写注意事项

08.15_面向对象(方法重写的面试题)(掌握)

- A:方法重写的面试题
    - Override和Overload的区别?Overload能改变返回值类型吗?
    - overload可以改变返回值类型,只看参数列表
    - 方法重写：子类中出现了和父类中方法声明一模一样的方法。与返回值类型有关,返回值是一致(或者是子父类)的
    - 方法重载：本类中出现的方法名一样，参数列表不同的方法。与返回值类型无关。
    - 子类对象调用方法的时候：
        - 先找子类本身，再找父类。

08.16_面向对象(使用继承前的学生和老师案例)(掌握)

- A:案例演示
    - 使用继承前的学生和老师案例
    - 属性:姓名,年龄
    - 行为:吃饭
    - 老师有特有的方法:讲课
    - 学生有特有的方法:学习

08.17_面向对象(使用继承后的学生和老师案例)(掌握)

- A:案例演示
    - 使用继承后的学生和老师案例

08.18_面向对象(猫狗案例分析,实现及测试)(掌握)

- A:猫狗案例分析
- B:案例演示
    - 猫狗案例继承版
    - 属性:毛的颜色,腿的个数
    - 行为:吃饭
    - 猫特有行为:抓老鼠catchMouse
    - 狗特有行为:看家lookHome

08.19_面向对象(final关键字修饰类,方法以及变量的特点)(掌握)

- A:final概述
- B:final修饰特点
    - 修饰类，类不能被继承
    - 修饰变量，变量就变成了常量，只能被赋值一次
    - 修饰方法，方法不能被重写
- C:案例演示
    - final修饰特点

08.20_面向对象(final关键字修饰局部变量)(掌握)

- A:案例演示
    - 方法内部或者方法声明上都演示一下(了解)
    - 基本类型，是值不能被改变
    - 引用类型，是地址值不能被改变,对象中的属性可以改变

08.21_面向对象(final修饰变量的初始化时机)(掌握)

- A:final修饰变量的初始化时机
    - 显示初始化

    - 在对象构造完毕前即可

08.22_day08总结

- 把今天的知识点总结一遍。

10.01_面向对象(package关键字的概述及作用)(了解)

- A:为什么要有包
  - 将字节码(.class)进行分类存放
  - 包其实就是文件夹
- B:包的概述
- 举例： 学生：增加，删除，修改，查询 老师：增加，删除，修改，查询 ...

```
方案1：按照功能分
    com.heima.add
        AddStudent
        AddTeacher
    com.heima.delete
        DeleteStudent
        DeleteTeacher
    com.heima.update
        UpdateStudent
        UpdateTeacher
    com.heima.find
        FindStudent
        FindTeacher

方案2：按照模块分
    com.heima.teacher
        AddTeacher
        DeleteTeacher
        UpdateTeacher
        FindTeacher
    com.heima.student
        AddStudent
        DeleteStudent
        UpdateStudent
        FindStudent
```

10.02_面向对象(包的定义及注意事项)(掌握)

- A:定义包的格式
  - package 包名;
  - 多级包用.分开即可
- B:定义包的注意事项
  - A:package语句必须是程序的第一条可执行的代码
  - B:package语句在一个java文件中只能有一个
  - C:如果没有package，默认表示无包名
- C:案例演示
  - 包的定义及注意事项

10.03_面向对象(带包的类编译和运行)(掌握)

- A:如何编译运行带包的类
  - a:javac编译的时候带上-d即可
    - javac -d . HelloWorld.java
  - b:通过java命令执行。
    - java 包名.HellWord

10.04_面向对象(不同包下类之间的访问)(掌握)

- A:案例演示
  - 不同包下类之间的访问

10.05_面向对象(import关键字的概述和使用)(掌握)

- A:案例演示
  - 为什么要有import
    - 其实就是让有包的类对调用者可见,不用写全类名了
- B:导包格式
  - import 包名;
  - 注意：
  - 这种方式导入是到类的名称。
  - 虽然可以最后写*，但是不建议。
- C:package,import,class有没有顺序关系(面试题)

10.06_面向对象(四种权限修饰符的测试)(掌握)

- A:案例演示
  - 四种权限修饰符
- B:结论

```
            本类    同一个包下(子类和无关类)  不同包下(子类)    不同包下(无关类)
private     Y
默认          Y       Y
protected   Y       Y                          Y
public      Y       Y                          Y               Y
```

10.07_面向对象(类及其组成所使用的常见修饰符)(掌握)

- A:修饰符：
  - 权限修饰符：private，默认的，protected，public
  - 状态修饰符：static，final
  - 抽象修饰符：abstract
- B:类：
  - 权限修饰符：默认修饰符，public
  - 状态修饰符：final
  - 抽象修饰符：abstract
  - 用的最多的就是：public
- C:成员变量：
  - 权限修饰符：private，默认的，protected，public
  - 状态修饰符：static，final
  - 用的最多的就是：private
- D:构造方法：
  - 权限修饰符：private，默认的，protected，public
  - 用的最多的就是：public
- E:成员方法：
  - 权限修饰符：private，默认的，protected，public
  - 状态修饰符：static，final
  - 抽象修饰符：abstract
  - 用的最多的就是：public
- F:除此以外的组合规则：
  - 成员变量：public static final
  - 成员方法：
    - public static
    - public abstract
    - public final

10.08_面向对象(内部类概述和访问特点)(了解)

- A:内部类概述
- B:内部类访问特点
  - a:内部类可以直接访问外部类的成员，包括私有。
  - b:外部类要访问内部类的成员，必须创建对象。
  - 外部类名.内部类名 对象名 = 外部类对象.内部类对象;
- C:案例演示
  - 内部类极其访问特点

10.09_面向对象(成员内部类私有使用)(了解)

- private

10.10_面向对象(静态成员内部类)(了解)

- static
- B:成员内部类被静态修饰后的访问方式是:
  - 外部类名.内部类名 对象名 = 外部类名.内部类对象;

10.11_面向对象(成员内部类的面试题)(掌握)

- A:面试题

```
要求：使用已知的变量，在控制台输出30，20，10。

class Outer {
    public int num = 10;
    class Inner {
        public int num = 20;
        public void show() {
            int num = 30;
            System.out.println(?);
            System.out.println(??);
            System.out.println(???);
        }
    }
}
class InnerClassTest {
    public static void main(String[] args) {
        Outer.Inner oi = new Outer().new Inner();
        oi.show();
    }
}
```

成员内部类

```
class Test1_InnerClass {
 public static void main(String[] args) {
  Outer.Inner oi = new Outer().new Inner();
  oi.show();
 }
}
//要求：使用已知的变量，在控制台输出30，20，10。
//内部类之所以能获取到外部类的成员,是因为他能获取到外部类的引用外部类名.this
class Outer {
 public int num = 10;
 class Inner {
  public int num = 20;
  public void show() {
   int num = 30;
   System.out.println(num);
   System.out.println(this.num);
   System.out.println(Outer.this.num);
  }
 }
}
```

10.12_面向对象(局部内部类访问局部变量的问题)(掌握)

- A:案例演示
  - 局部内部类访问局部变量必须用final修饰
  - 局部内部类在访问他所在方法中的局部变量必须用final修饰,为什么? 因为当调用这个方法时,局部变量如果没有用final修饰,他的生命周期和方法的生命周期是一样的,当方法弹栈,这个局部变量也会消失,那么如果局部内部类对象还没有马上消失想用这个局部变量,就没有了,如果用final修饰会在类加载的时候进入常量池,即使方法弹栈,常量池的常量还在,也可以继续使用

    但是jdk1.8取消了这个事情,所以我认为这是个bug

10.13_面向对象(匿名内部类的格式和理解)

- A:匿名内部类
  - 就是内部类的简化写法。
- B:前提：存在一个类或者接口
  - 这里的类可以是具体类也可以是抽象类。
- C:格式：

```
new 类名或者接口名(){
    重写方法;
}
```

- D:本质是什么呢?
  - 是一个继承了该类或者实现了该接口的子类匿名对象。
- E:案例演示
  - 按照要求来一个匿名内部类

10.14_面向对象(匿名内部类重写多个方法调用)

- A:案例演示
  - 匿名内部类的方法调用

10.15_面向对象(匿名内部类在开发中的应用)

- A:代码如下

```
//这里写抽象类，接口都行
abstract class Person {
    public abstract void show();
}

class PersonDemo {
    public void method(Person p) {
        p.show();
    }
}

class PersonTest {
    public static void main(String[] args) {
        //如何调用PersonDemo中的method方法呢?
        PersonDemo pd = new PersonDemo ();

    }
```

```
pd.method(new Person() {
   public void show() {
    System.out.println("show");
   }
  });
```

```
}
```

10.16_面向对象(匿名内部类的面试题)

- A:面试题

```
按照要求，补齐代码
interface Inter { void show(); }
class Outer { //补齐代码 }
class OuterDemo {
    public static void main(String[] args) {
          Outer.method().show();
      }
}
要求在控制台输出”HelloWorld”
```

分析：Outer.method()表示
这是一个静态方法（类名调用的方法）
.show()表示method（）方法表示其返回值
是对象。

```
//补齐代码
 public static Inter method() {
  return new Inter() {
   public void show() {
    System.out.println("show");
   }
  };
 }
```

10.17_day10总结

- 把今天的知识点总结一遍。

11.01_Java开发工具(常见开发工具介绍)(了解)

- A:操作系统自带的记事本软件
- B:高级记事本软件
- C:集成开发环境 IDE
  - (Integrated Development Environment)
- D:Eclipse和MyEclipse的区别
  - a:Eclipse是一种可扩展的开放源代码的IDE。
  - b:Eclipse的特点描述
    - 免费
    - 纯Java语言编写
    - 免安装
    - 扩展性强
  - c:MyEclipse
    - 在Eclipse基础上追加的功能性插件，对插件收费
    - 在WEB开发中提供强大的系统架构平台
- E:下载 http://eclipse.org/
  - org是非盈利团体
- F:安装
  - 绿色版 解压就可以使用(Eclipse)
  - 安装版 双击运行,一路next即可(JDK)
- G:卸载
  - 绿色版 直接删除文件夹即可
  - 安装版 专业卸载软件或者控制面板添加删除程序

11.02_Java开发工具(Eclipse中HelloWorld案例以及汉化)(了解)

- A:选择工作空间
  - 工作空间 其实就是我们写的源代码所在的目录
- B:用Eclipse来完成一个HelloWorld案例
  - 代码以项目为基本单位
  - 创建项目
  - 创建包
  - 创建类
  - 编写代码
- C:编译和运行
- D:Eclipse的汉化
  - 从Eclipse3.5开始，安装目录下就多了一个dropins目录,只要将插件解压后放到到该目录即可。
  - 同理，这种方式卸载插件也是特别的方便，推荐这种方式
- E:语法检查提示
  - 红色波浪线
    - 必须处理,否则编译通不过
  - 黄色波浪线
    - 可以不搭理他

11.03_Java开发工具(Eclipse的视窗和视图概述)(了解)

- A:视窗 每一个基本的窗体被称为视窗
  - PackageExplorer 显示项目结构，包，类，及资源
  - Outline 显示类的结构，方便查找，识别，修改
  - Console 程序运行的结果在该窗口显示
  - Hierarchy 显示Java继承层次结构，选中类后F4
- B:视图 是由某些视窗的组合而成的
  - Java视图
  - Debug视图

11.04_Java开发工具(Eclipse工作空间的基本配置)(掌握)

- A:程序的编译和运行的环境配置(一般不改)
  - window -- Preferences -- Java
  - 编译环境：Compiler 默认选中的就是最高版本。
  - 运行环境：Installed JREs 默认会找你安装的那个JDK。建议配置了Java的环境变量。
  - 问题：
    - 低编译，高运行。可以。
    - 高编译，低运行。不可以。
    - 建议，编译和运行的版本一致。
- B:如何去掉默认注释?
  - window -- Preferences -- Java -- Code Style -- Code Templates
  - 选择你不想要的内容，通过右边Edit编辑。

    - 注意：请只删除注释部分，不是注释部分的不要删除。
- C:行号的显示和隐藏
    - 显示：在代码区域的最左边的空白区域，右键 -- Show Line Numbers即可。
    - 隐藏：把上面的动作再做一次。
- D:字体大小及颜色
    - a:Java代码区域的字体大小和颜色：
        - window -- Preferences -- General -- Appearance -- Colors And Fonts --Java修改 -- Java Edit Text Font
    - b:控制台
        - window -- Preferences -- General -- Appearance -- Colors And Fonts -- Debug -- Console font
    - c:其他文件
        - window -- Preferences -- General -- Appearance -- Colors And Fonts -- Basic -- Text Font
- E:窗体给弄乱了，怎么办?
    - window -- Reset Perspective
- F:控制台找不到了，怎么办?
    - Window--Show View—Console
- G:取消悬浮提示
    - window -- Preferences -- Java--Editor--Hovers。右边将Combined Hover勾去掉。
    - 这样代码的悬浮框就不会自动出现了。如果想看提示，将光标悬浮在代码上，按F2即可。

11.05_Java开发工具(Eclipse中内容辅助键的使用)(掌握)

- A:Alt+/ 起提示作用
- B:main+alt+/,syso+alt+/,给出其他提示
- C:补充输出语句,选中需要输出的部分,alt+/选择最后一项即可
- C:定义自己的alt + /
    - windows--perference-Java-Editor-Templates--New

11.06_Java开发工具(Eclipse中快捷键的使用)(掌握)

- A:新建 ctrl + n
- B:格式化 ctrl+shift+f
- C:导入包 ctrl+shift+o
- D:注释 ctrl+/,ctrl+shift+/,ctrl+shift+\
- E:代码上下移动 选中代码alt+上/下箭头
- F:查看源码 选中类名(F3或者Ctrl+鼠标点击)
- G:查找具体的类 ctrl + shift + t
- H:查找具体类的具体方法 ctrl + o
- I:给建议 ctrl+1,根据右边生成左边的数据类型,生成方法
- J:删除代码 ctrl + d
- K:抽取方法alt + shift + m
- L:改名alt + shift + r
-

11.07_Java开发工具(Eclipse中如何提高开发效率)(掌握)

- alt + shift + s
- A:自动生成构造方法
- B:自动生成get/set方法

11.08_Java开发工具(Eclipse中一个标准学生类及其测试)(掌握)

- A:案例演示
    - 用Eclipse实现标准学生类及其测试

11.09_Java开发工具(Eclipse中接口抽象类具体类代码体现)(掌握)

- A:案例演示
    - 用Eclipse实现接口抽象类具体类代码

11.10_Java开发工具(Eclipse中如何生成jar包并导入到项目中)(了解)

- A:jar是什么?
    - jar是多个class文件的压缩包。
- B:jar有什么用?
    - 用别人写好的东西
- C:打jar包
    - 选中项目--右键--Export--Java--Jar--自己指定一个路径和一个名称--Finish
- D:导入jar包
    - 复制到项目路径下并添加至构建路径。

11.11_Java开发工具(Eclipse中如何删除项目和导入项目)(掌握)

- A:删除项目
  - 选中项目 – 右键 – 删除
    - 从项目区域中删除
    - 从硬盘上删除
- B:导入项目
  - 在项目区域右键找到import
  - 找到General，展开，并找到
  - Existing Projects into Workspace
  - 点击next,然后选择你要导入的项目
  - 注意：这里选择的是项目名称

11.12_Java开发工具(Eclipse中断点调试的基本使用)(了解)

- A:Debug的作用
  - 调试程序
  - 查看程序执行流程
- B:如何查看程序执行流程
  - 什么是断点：
    - 就是一个标记，从哪里开始。
  - 如何设置断点：
    - 你想看哪里的程序，你就在那个有效程序的左边双击即可。
  - 在哪里设置断点：
    - 哪里不会点哪里。
    - 目前：我们就在每个方法的第一条有效语句上都加。
  - 如何运行设置断点后的程序：
    - 右键 -- Debug as -- Java Application
  - 看哪些地方：
    - Debug：断点测试的地方
      - 在这个地方，记住F6，或者点击也可以。一次看一行的执行过程。
    - Variables：查看程序的变量变化
    - ForDemo：被查看的源文件
    - Console：控制台
  - 如何去断点：
    - a:再次双击即可
    - b:找到Debug视图，Variables界面，找到Breakpoints，并点击，然后看到所有的断点，最后点击那个双叉。

11.13_Java开发工具(Eclipse查看Java中参数传递问题)(了解)

- A:断点演示
  - 断点查看Java中参数传递问题

11.14_常见对象(API概述)(了解)

- A:API(Application Programming Interface)
  - 应用程序编程接口
- B:Java API
  - 就是Java提供给我们使用的类，这些类将底层的实现封装了起来，
  - 我们不需要关心这些类是如何实现的，只需要学习这些类如何使用。

11.15_常见对象(Object类的概述)(了解)

- A:Object类概述
  - 类层次结构的根类
  - 所有类都直接或者间接的继承自该类
- B:构造方法
  - public Object()
  - 回想面向对象中为什么说：
  - 子类的构造方法默认访问的是父类的无参构造方法

11.16_常见对象(Object类的hashCode()方法)(了解)

- A:案例演示
  - public int hashCode()

- a:返回该对象的哈希码值。默认情况下，该方法会根据对象的地址来计算。
- b:不同对象的，hashCode()一般来说不会相同。但是，同一个对象的hashCode()值肯定相同。

11.17_常见对象(Object类的getClass()方法)(在反射的时候掌握)

- A:案例演示
  - public final Class getClass()
  - a:返回此 Object 的运行时类。
  - b:可以通过Class类中的一个方法，获取对象的真实类的全名称。
    - public String getName()

11.18_常见对象(Object类的toString()方法)(掌握)

- A:案例演示
  - public String toString()
  - a:返回该对象的字符串表示。

```
public Stirng toString() {
    return name + "," + age;
}
```

  - b:它的值等于：
    - getClass().getName() + "@" + Integer.toHexString(hashCode())
  - c:由于默认情况下的数据对我们来说没有意义，一般建议重写该方法。
- B:最终版
  - 自动生成

11.19_常见对象(Object类的equals()方法)(掌握)

- A:案例演示
  - a:指示其他某个对象是否与此对象“相等”。
  - b:默认情况下比较的是对象的引用是否相同。
  - c:由于比较对象的引用没有意义，一般建议重写该方法。

11.20_常见对象(==号和equals方法的区别)(掌握)

- ==是一个比较运算符号,既可以比较基本数据类型,也可以比较引用数据类型,基本数据类型比较的是值,引用数据类型比较的是地址值
- equals方法是一个方法,只能比较引用数据类型,所有的对象都会继承Object类中的方法,如果没有重写Object类中的equals方法,equals方法和==号比较引用数据类型无区别,重写后的equals方法比较的是对象中的属性

11.21_day11总结

- 把今天的知识点总结一遍。

13.01_常见对象(StringBuffer类的概述)

- A:StringBuffer类概述
  - 通过JDK提供的API，查看StringBuffer类的说明
  - 线程安全的可变字符序列
- B:StringBuffer和String的区别
  - String是一个不可变的字符序列
  - StringBuffer是一个可变的字符序列

13.02_常见对象(StringBuffer类的构造方法)

- A:StringBuffer的构造方法：
  - public StringBuffer():无参构造方法
  - public StringBuffer(int capacity):指定容量的字符串缓冲区对象
  - public StringBuffer(String str):指定字符串内容的字符串缓冲区对象
- B:StringBuffer的方法：
  - public int capacity()：返回当前容量。 理论值(不掌握)
  - public int length():返回长度（字符数）。 实际值
- C:案例演示
  - 构造方法和长度方法的使用

13.03_常见对象(StringBuffer的添加功能)

- A:StringBuffer的添加功能
  - public StringBuffer append(String str):
    - 可以把任意类型数据添加到字符串缓冲区里面,并返回字符串缓冲区本身
  - public StringBuffer insert(int offset,String str):
    - 在指定位置把任意类型的数据插入到字符串缓冲区里面,并返回字符串缓冲区本身

13.04_常见对象(StringBuffer的删除功能)

- A:StringBuffer的删除功能
  - public StringBuffer deleteCharAt(int index):
    - 删除指定位置的字符，并返回本身
  - public StringBuffer delete(int start,int end):
    - 删除从指定位置开始指定位置结束的内容，并返回本身

13.05_常见对象(StringBuffer的替换和反转功能)

- A:StringBuffer的替换功能
  - public StringBuffer replace(int start,int end,String str):
    - 从start开始到end用str替换
- B:StringBuffer的反转功能
  - public StringBuffer reverse():
    - 字符串反转

13.06_常见对象(StringBuffer的截取功能及注意事项)

- A:StringBuffer的截取功能
  - public String substring(int start):
    - 从指定位置截取到末尾
  - public String substring(int start,int end):
    - 截取从指定位置开始到结束位置，包括开始位置，不包括结束位置
- B:注意事项
  - 注意:返回值类型不再是StringBuffer本身

13.07_常见对象(StringBuffer和String的相互转换)

- A:String -- StringBuffer
  - a:通过构造方法
  - b:通过append()方法
- B:StringBuffer -- String
  - a:通过构造方法
  - b:通过toString()方法
  - c:通过subString(0,length);

13.08_常见对象(把数组转成字符串)

- A:案例演示

- 需求：把数组中的数据按照指定个格式拼接成一个字符串

```
举例：
    int[] arr = {1,2,3};
输出结果：
    "[1, 2, 3]"

用StringBuffer的功能实现
```

13.09_常见对象(字符串反转)

- A:案例演示

```
需求：把字符串反转
    举例：键盘录入"abc"
    输出结果："cba"

用StringBuffer的功能实现
```

13.10_常见对象(StringBuffer和StringBuilder的区别)

- A:StringBuilder的概述
  - 通过查看API了解一下StringBuilder类
- B:面试题
  - String,StringBuffer,StringBuilder的区别
  - StringBuffer和StringBuilder的区别
  - StringBuffer是jdk1.0版本的,是线程安全的,效率低
  - StringBuilder是jdk1.5版本的,是线程不安全的,效率高
  - String和StringBuffer,StringBuilder的区别
  - String是一个不可变的字符序列
  - StringBuffer,StringBuilder是可变的字符序列

13.11_常见对象(String和StringBuffer分别作为参数传递)

- A:形式参数问题
  - String作为参数传递
  - StringBuffer作为参数传递
- B:案例演示
  - String和StringBuffer分别作为参数传递问题

13.12_常见对象(数组高级冒泡排序原理图解)

- A:画图演示

```
需求：
    数组元素：{24, 69, 80, 57, 13}
    请对数组元素进行排序。

    冒泡排序
        相邻元素两两比较，大的往后放，第一次完毕，最大值出现在了最大索引处
```

13.13_常见对象(数组高级冒泡排序代码实现)

- A:案例演示
  - 数组高级冒泡排序代码

13.14_常见对象(数组高级选择排序原理图解)

- A:画图演示
  - 需求：
    - 数组元素：{24, 69, 80, 57, 13}
    - 请对数组元素进行排序。
    - 选择排序
      - 从0索引开始，依次和后面元素比较，小的往前放，第一次完毕，最小值出现在了最小索引处

13.15_常见对象(数组高级选择排序代码实现)

- A:案例演示

- 数组高级选择排序代码

13.16_常见对象(数组高级二分查找原理图)

- A:画图演示
  - 二分查找
  - 前提：数组元素有序

13.17_常见对象(数组高级二分查找代码实现及注意事项)

- A:案例演示
  - 数组高级二分查找代码
- B:注意事项
  - 如果数组无序，就不能使用二分查找。
    - 因为如果你排序了，但是你排序的时候已经改变了我最原始的元素索引。

13.18_常见对象(Arrays类的概述和方法使用)

- A:Arrays类概述
  - 针对数组进行操作的工具类。
  - 提供了排序，查找等功能。
- B:成员方法
  - public static String toString(int[] a)
  - public static void sort(int[] a)
  - public static int binarySearch(int[] a,int key)

13.19_常见对象(基本类型包装类的概述)

- A:为什么会有基本类型包装类
  - 将基本数据类型封装成对象的好处在于可以在对象中定义更多的功能方法操作该数据。
- B:常用操作
  - 常用的操作之一：用于基本数据类型与字符串之间的转换。
- C:基本类型和包装类的对应

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

13.20_常见对象(Integer类的概述和构造方法)

- A:Integer类概述
  - 通过JDK提供的API，查看Integer类的说明
  - Integer 类在对象中包装了一个基本类型 int 的值,
  - 该类提供了多个方法，能在 int 类型和 String 类型之间互相转换，
  - 还提供了处理 int 类型时非常有用的其他一些常量和方法
- B:构造方法
  - public Integer(int value)
  - public Integer(String s)
- C:案例演示
  - 使用构造方法创建对象

13.21_常见对象(String和int类型的相互转换)

- A:int -- String
  - a:和""进行拼接
  - b:public static String valueOf(int i)
  - c:int -- Integer -- String(Integer类的toString方法())
  - d:public static String toString(int i)(Integer类的静态方法)
- B:String -- int
  - a:String -- Integer -- int
  - public static int parseInt(String s)

13.22_常见对象(JDK5的新特性自动装箱和拆箱)

- A:JDK5的新特性

    - 自动装箱：把基本类型转换为包装类类型
    - 自动拆箱：把包装类类型转换为基本类型
- B:案例演示
    - JDK5的新特性自动装箱和拆箱
    - Integer ii = 100;
    - ii += 200;
- C:注意事项
    - 在使用时，Integer x = null;代码就会出现NullPointerException。
    - 建议先判断是否为null，然后再使用。

13.23_常见对象(Integer的面试题)

- A:Integer的面试题

```
看程序写结果

Integer i1 = new Integer(97);
Integer i2 = new Integer(97);
System.out.println(i1 == i2);
System.out.println(i1.equals(i2));
System.out.println("-----------");

Integer i3 = new Integer(197);
Integer i4 = new Integer(197);
System.out.println(i3 == i4);
System.out.println(i3.equals(i4));
System.out.println("-----------");

Integer i5 = 97;
Integer i6 = 97;
System.out.println(i5 == i6);
System.out.println(i5.equals(i6));
System.out.println("-----------");

Integer i7 = 197;
Integer i8 = 197;
System.out.println(i7 == i8);
System.out.println(i7.equals(i8));
```

```
false
true
-----------
false
true
-----------
true
true
-----------
false
true
```

-128到127是byte的取值范围,如果在这个取值范围内,自动装箱就不会新创建对象,而是从常量池中获取

* 如果超过了byte 取值范围就会再新创建对象

自动封箱

13.24_day13总结

- 把今天的知识点总结一遍。

14.01_常见对象(正则表达式的概述和简单使用)

- A:正则表达式
  - 是指一个用来描述或者匹配一系列符合某个语法规则的字符串的单个字符串。其实就是一种规则。有自己特殊的应用。
  - 作用:比如注册邮箱,邮箱有用户名和密码,一般会对其限制长度,这个限制长度的事情就是正则表达式做的
- B:案例演示
  - 需求：校验qq号码.
    - 1:要求必须是5-15位数字
    - 2:0不能开头
    - 3:必须都是数字
  - a:非正则表达式实现
  - b:正则表达式实现

14.02_常见对象(字符类演示)

- A:字符类
  - [abc] a、b 或 c（简单类）
  - [^abc] 任何字符，除了 a、b 或 c（否定）
  - [a-zA-Z] a到 z 或 A到 Z，两头的字母包括在内（范围）
  - [0-9] 0到9的字符都包括

14.03_常见对象(预定义字符类演示)

- A:预定义字符类
  - . 任何字符。
  - \d 数字：[0-9]
  - \w 单词字符：[a-zA-Z_0-9]

14.04_常见对象(数量词)

- A:Greedy 数量词
  - X? X，一次或一次也没有
  - X* X，零次或多次
  - X+ X，一次或多次
  - X{n} X，恰好 n 次
  - X{n,} X，至少 n 次
  - X{n,m} X，至少 n 次，但是不超过 m 次

14.05_常见对象(正则表达式的分割功能)

- A:正则表达式的分割功能
  - String类的功能：public String[] split(String regex)
- B:案例演示
  - 正则表达式的分割功能

14.06_常见对象(把给定字符串中的数字排序)

- A:案例演示
  - 需求：我有如下一个字符串:”91 27 46 38 50”，请写代码实现最终输出结果是：”27 38 46 50 91”

14.07_常见对象(正则表达式的替换功能)

- A:正则表达式的替换功能
  - String类的功能：public String replaceAll(String regex,String replacement)
- B:案例演示
  - 正则表达式的替换功能

14.08_常见对象(正则表达式的分组功能)

- A:正则表达式的分组功能
  - 捕获组可以通过从左到右计算其开括号来编号。例如，在表达式 ((A)(B(C))) 中，存在四个这样的组：

```
1     ((A)(B(C)))
2     (A
3     (B(C))
4     (C)

组零始终代表整个表达式。
```

B:案例演示 a:切割 需求：请按照叠词切割： "sdqqfgkkkhjppppkl"; b:替换 需求：我我....我...我.要...要要...要学....学学..学.编..编编.编.程.程.程..程 将字符串还原成:“我要学编程”。

14.09_常见对象(Pattern和Matcher的概述)

- A:Pattern和Matcher的概述
- B:模式和匹配器的典型调用顺序
  - 通过JDK提供的API，查看Pattern类的说明
  - 典型的调用顺序是
  - Pattern p = Pattern.compile("a*b");
  - Matcher m = p.matcher("aaaaab");
  - boolean b = m.matches();

14.10_常见对象(正则表达式的获取功能)

- A:正则表达式的获取功能
  - Pattern和Matcher的结合使用
- B:案例演示
  - 需求：把一个字符串中的手机号码获取出来

14.11_常见对象(Math类概述和方法使用)

- A:Math类概述
  - Math 类包含用于执行基本数学运算的方法，如初等指数、对数、平方根和三角函数。
- B:成员方法
  - public static int abs(int a)
  - public static double ceil(double a)
  - public static double floor(double a)
  - public static int max(int a,int b) min自学
  - public static double pow(double a,double b)
  - public static double random()
  - public static int round(float a) 参数为double的自学
  - public static double sqrt(double a)

14.12_常见对象(Random类的概述和方法使用)

- A:Random类的概述
  - 此类用于产生随机数如果用相同的种子创建两个 Random 实例，
  - 则对每个实例进行相同的方法调用序列，它们将生成并返回相同的数字序列。
- B:构造方法
  - public Random()
  - public Random(long seed)
- C:成员方法
  - public int nextInt()
  - public int nextInt(int n)(重点掌握)

14.13_常见对象(System类的概述和方法使用)

- A:System类的概述
  - System 类包含一些有用的类字段和方法。它不能被实例化。
- B:成员方法
  - public static void gc()
  - public static void exit(int status)
  - public static long currentTimeMillis()
  - pubiic static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)
- C:案例演示
  - System类的成员方法使用

14.14_常见对象(BigInteger类的概述和方法使用)

- A:BigInteger的概述
  - 可以让超过Integer范围内的数据进行运算
- B:构造方法
  - public BigInteger(String val)
- C:成员方法
  - public BigInteger add(BigInteger val)
  - public BigInteger subtract(BigInteger val)
  - public BigInteger multiply(BigInteger val)

- public BigInteger divide(BigInteger val)
- public BigInteger[] divideAndRemainder(BigInteger val)

14.15_常见对象(BigDecimal类的概述和方法使用)

- A:BigDecimal的概述
  - 由于在运算的时候，float类型和double很容易丢失精度，演示案例。
  - 所以，为了能精确的表示、计算浮点数，Java提供了BigDecimal
  - 不可变的、任意精度的有符号十进制数。
- B:构造方法
  - public BigDecimal(String val)
- C:成员方法
  - public BigDecimal add(BigDecimal augend)
  - public BigDecimal subtract(BigDecimal subtrahend)
  - public BigDecimal multiply(BigDecimal multiplicand)
  - public BigDecimal divide(BigDecimal divisor)
- D:案例演示
  - BigDecimal类的构造方法和成员方法使用

14.16_常见对象(Date类的概述和方法使用)(掌握)

- A:Date类的概述
  - 类 Date 表示特定的瞬间，精确到毫秒。
- B:构造方法
  - public Date()
  - public Date(long date)
- C:成员方法
  - public long getTime()
  - public void setTime(long time)

14.17_常见对象(SimpleDateFormat类实现日期和字符串的相互转换)(掌握)

- A:DateFormat类的概述
  - DateFormat 是日期/时间格式化子类的抽象类，它以与语言无关的方式格式化并解析日期或时间。是抽象类，所以使用其子类SimpleDateFormat
- B:SimpleDateFormat构造方法
  - public SimpleDateFormat()
  - public SimpleDateFormat(String pattern)
- C:成员方法
  - public final String format(Date date)
  - public Date parse(String source)

"yyyy.MM.dd G 'at' HH:mm:ss z" 2001.07.04 AD at 12:08:56 PDT

SimpleDateFormat不是线程安全的

14.18_常见对象(你来到这个世界多少天案例)(掌握)

- A:案例演示
  - 需求：算一下你来到这个世界多少天?

14.19_常见对象(Calendar类的概述和获取日期的方法)(掌握)

- A:Calendar类的概述
  - Calendar 类是一个抽象类，它为特定瞬间与一组诸如 YEAR、MONTH、DAY*OF*MONTH、HOUR 等日历字段之间的转换提供了一些方法，并为操作日历字段（例如获得下星期的日期）提供了一些方法。
- B:成员方法
  - public static Calendar getInstance()
  - public int get(int field)

14.20_常见对象(Calendar类的add()和set()方法)(掌握)

- A:成员方法
  - public void add(int field,int amount)
  - public final void set(int year,int month,int date)
- B:案例演示
  - Calendar类的成员方法使用

14.21_常见对象(如何获取任意年份是平年还是闰年)(掌握)

- A:案例演示
  - 需求：键盘录入任意一个年份，判断该年是闰年还是平年

14.22_day14总结

- 把今天的知识点总结一遍。

15.01_集合框架(对象数组的概述和使用)

- A:案例演示
  - 需求：我有5个学生，请把这个5个学生的信息存储到数组中，并遍历数组，获取得到每一个学生信息。

```
Student[] arr = new Student[5];                 //存储学生对象
arr[0] = new Student("张三", 23);
arr[1] = new Student("李四", 24);
arr[2] = new Student("王五", 25);
arr[3] = new Student("赵六", 26);
arr[4] = new Student("马哥", 20);

for (int i = 0; i < arr.length; i++) {
    System.out.println(arr[i]);
}
```

- B:画图演示
  - 把学生数组的案例画图讲解
  - 数组和集合存储引用数据类型,存的都是地址值

15.02_集合框架(集合的由来及集合继承体系图)

- A:集合的由来
  - 数组长度是固定,当添加的元素超过了数组的长度时需要对数组重新定义,太麻烦,java内部给我们提供了集合类,能存储任意对象,长度是可以改变的,随着元素的增加而增加,随着元素的减少而减少
- B:数组和集合的区别
  - 区别1：
    - 数组既可以存储基本数据类型,又可以存储引用数据类型,基本数据类型存储的是值,引用数据类型存储的是地址值
    - 集合只能存储引用数据类型(对象)集合中也可以存储基本数据类型,但是在存储的时候会自动装箱变成对象
  - 区别2:
    - 数组长度是固定的,不能自动增长
    - 集合的长度的是可变的,可以根据元素的增加而增长
- C:数组和集合什么时候用 * 1,如果元素个数是固定的推荐用数组 * 2,如果元素个数不是固定的推荐用集合
- D:集合继承体系图

15.03_集合框架(Collection集合的基本功能测试)

- A:案例演示

```
基本功能演示

boolean add(E e)
boolean remove(Object o)
void clear()
boolean contains(Object o)
boolean isEmpty()
int size()
```

- B:注意:

```
collectionXxx.java使用了未经检查或不安全的操作.
注意:要了解详细信息,请使用 -Xlint:unchecked重新编译.
java编译器认为该程序存在安全隐患
温馨提示:这不是编译失败,所以先不用理会,等学了泛型你就知道了
```

15.04_集合框架(集合的遍历之集合转数组遍历)

- A:集合的遍历
  - 其实就是依次获取集合中的每一个元素。
- B:案例演示
  - 把集合转成数组，可以实现集合的遍历
  - toArray() *

```
Collection coll = new ArrayList();
coll.add(new Student("张三",23));     //Object obj = new Student("张三",23);
coll.add(new Student("李四",24));
coll.add(new Student("王五",25));
coll.add(new Student("赵六",26));

Object[] arr = coll.toArray();              //将集合转换成数组
for (int i = 0; i < arr.length; i++) {
    Student s = (Student)arr[i];            //强转成Student
```

```
    System.out.println(s.getName() + "," + s.getAge());
}
```

15.05_集合框架(Collection集合的带All功能测试)

- A:案例演示

```
带All的功能演示

boolean addAll(Collection c)
boolean removeAll(Collection c)
boolean containsAll(Collection c)
boolean retainAll(Collection c)
```

15.06_集合框架(集合的遍历之迭代器遍历)

- A:迭代器概述
  - 集合是用来存储元素,存储的元素需要查看,那么就需要迭代(遍历)
- B:案例演示
  - 迭代器的使用

```
Collection c = new ArrayList();
c.add("a");
c.add("b");
c.add("c");
c.add("d");

Iterator it = c.iterator();                    //获取迭代器的引用
while(it.hasNext()) {                          //集合中的迭代方法(遍历)
    System.out.println(it.next());
}
```

15.07_集合框架(Collection存储自定义对象并遍历)

- A:案例演示
  - Collection存储自定义对象并用迭代器遍历

```
Collection c = new ArrayList();

c.add(new Student("张三",23));
c.add(new Student("李四",24));
c.add(new Student("王五",25));
c.add(new Student("赵六",26));
c.add(new Student("赵六",26));

for(Iterator it = c.iterator();it.hasNext();) {
    Student s = (Student)it.next();                    //向下转型
    System.out.println(s.getName() + "," + s.getAge()); //获取对象中的姓名和年龄
}
System.out.println("------------------------------");
Iterator it = c.iterator();                            //获取迭代器
while(it.hasNext()) {                                  //判断集合中是否有元素
    //System.out.println(((Student)(it.next())).getName() + "," + ((Student)(it.next())).getAge());
    Student s = (Student)it.next();                    //向下转型
    System.out.println(s.getName() + "," + s.getAge()); //获取对象中的姓名和年龄
}
```

15.08_集合框架(迭代器的原理及源码解析)(了解)

- A:迭代器原理
  - 迭代器原理:迭代器是对集合进行遍历,而每一个集合内部的存储结构都是不同的,所以每一个集合存和取都是不一样,那么就需要在每一个类中定义hasNext()和next()方法,这样做是可以的,但是会让整个集合体系过于臃肿,迭代器是将这样的方法向上抽取出接口,然后在每个类的内部,定义自己迭代方式,这样做的好处有二,第一规定了整个集合体系的遍历方式都是hasNext()和next()方法,第二,代码有底层内部实现,使用者不用管怎么实现的,会用即可
- B:迭代器源码解析
  - 1,在eclipse中ctrl + shift + t找到ArrayList类
  - 2,ctrl+o查找iterator()方法
  - 3,查看返回值类型是new Itr(),说明Itr这个类实现Iterator接口
  - 4,查找Itr这个内部类,发现重写了Iterator中的所有抽象方法

15.09_集合框架(List集合的特有功能概述和测试)

- A:List集合的特有功能概述
  - void add(int index,E element)

- E remove(int index)
- E get(int index)
- E set(int index,E element)

15.10_集合框架(List集合存储学生对象并遍历)

- A:案例演示
  - 通过size()和get()方法结合使用遍历。

```
List list = new ArrayList();
list.add(new Student("张三", 18));
list.add(new Student("李四", 18));
list.add(new Student("王五", 18));
list.add(new Student("赵六", 18));

for(int i = 0; i < list.size(); i++) {
    Student s = (Student)list.get(i);
    System.out.println(s.getName() + "," + s.getAge());
}
```

15.11_集合框架(并发修改异常产生的原因及解决方案)

- A:案例演示
  - 需求：我有一个集合，请问，我想判断里面有没有"world"这个元素，如果有，我就添加一个"javaee"元素，请写代码实现。

```
List list = new ArrayList();
list.add("a");
list.add("b");
list.add("world");
list.add("d");
list.add("e");

/*Iterator it = list.iterator();
while(it.hasNext()) {
    String str = (String)it.next();
    if(str.equals("world")) {
        list.add("javaee");         //这里会抛出ConcurrentModificationException并发修改异常
    }
}*/
```

- B:ConcurrentModificationException出现
  - 迭代器遍历，集合修改集合
- C:解决方案
  - a:迭代器迭代元素，迭代器修改元素(ListIterator的特有功能add)
  - b:集合遍历元素，集合修改元素

```
ListIterator lit = list.listIterator();     //如果想在遍历的过程中添加元素,可以用ListIterator中的add方法
while(lit.hasNext()) {
    String str = (String)lit.next();
    if(str.equals("world")) {
        lit.add("javaee");
        //list.add("javaee");
    }
}
```

15.12_集合框架(ListIterator)(了解)

- boolean hasNext()是否有下一个
- boolean hasPrevious()是否有前一个
- Object next()返回下一个元素
- Object previous();返回上一个元素

15.13_集合框架(Vector的特有功能)

- A:Vector类概述
- B:Vector类特有功能
  - public void addElement(E obj)
  - public E elementAt(int index)
  - public Enumeration elements()
- C:案例演示

- Vector的迭代

```
Vector v = new Vector();                //创建集合对象,List的子类
v.addElement("a");
v.addElement("b");
v.addElement("c");
v.addElement("d");

//Vector迭代
Enumeration en = v.elements();          //获取枚举
while(en.hasMoreElements()) {           //判断集合中是否有元素
    System.out.println(en.nextElement());//获取集合中的元素
}
```

15.14_集合框架(数据结构之数组和链表)

- A:数组
  - 查询快修改也快
  - 增删慢
- B:链表
  - 查询慢,修改也慢
  - 增删快

15.15_集合框架(List的三个子类的特点)

- A:List的三个子类的特点

```
ArrayList:
    底层数据结构是数组，查询快，增删慢。
    线程不安全，效率高。
Vector:
    底层数据结构是数组，查询快，增删慢。
    线程安全，效率低。
Vector相对ArrayList查询慢(线程安全的)
Vector相对LinkedList增删慢(数组结构)
LinkedList:
    底层数据结构是链表，查询慢，增删快。
    线程不安全，效率高。

Vector和ArrayList的区别
    Vector是线程安全的,效率低
    ArrayList是线程不安全的,效率高
共同点:都是数组实现的
ArrayList和LinkedList的区别
    ArrayList底层是数组结果,查询和修改快
    LinkedList底层是链表结构的,增和删比较快,查询和修改比较慢
共同点:都是线程不安全的
```

- B:List有三个儿子，我们到底使用谁呢? 查询多用ArrayList 增删多用LinkedList 如果都多ArrayList

15.16_day15总结

把今天的知识点总结一遍。

16.01_集合框架(去除ArrayList中重复字符串元素方式)(掌握)

- A:案例演示
  - 需求：ArrayList去除集合中字符串的重复值(字符串的内容相同)
  - 思路：创建新集合方式

```
/**
 *  A:案例演示
 * 需求：ArrayList去除集合中字符串的重复值(字符串的内容相同)
 * 思路：创建新集合方式
 */
public static void main(String[] args) {
    ArrayList list = new ArrayList();
    list.add("a");
    list.add("a");
    list.add("b");
    list.add("b");
    list.add("b");
    list.add("c");
    list.add("c");
    list.add("c");
    list.add("c");

    System.out.println(list);
    ArrayList newList = getSingle(list);
    System.out.println(newList);
}

/*
 * 去除重复
 * 1,返回ArrayList
 * 2,参数列表ArrayList
 */
public static ArrayList getSingle(ArrayList list) {
    ArrayList newList = new ArrayList();            //创建一个新集合
    Iterator it = list.iterator();                  //获取迭代器
    while(it.hasNext()) {                           //判断老集合中是否有元素
        String temp = (String)it.next();            //将每一个元素临时记录住
        if(!newList.contains(temp)) {               //如果新集合中不包含该元素
            newList.add(temp);                      //将该元素添加到新集合中
        }
    }
    return newList;                                 //将新集合返回
}
```

16.02_集合框架(去除ArrayList中重复自定义对象元素)(掌握)

- A:案例演示
  - 需求：ArrayList去除集合中自定义对象元素的重复值(对象的成员变量值相同)
- B:注意事项
  - 重写equals()方法的

16.03_集合框架(LinkedList的特有功能)(掌握)

- A:LinkedList类概述
- B:LinkedList类特有功能
  - public void addFirst(E e)及addLast(E e)
  - public E getFirst()及getLast()
  - public E removeFirst()及public E removeLast()
  - public E get(int index);

16.04_集合框架(栈和队列数据结构)(掌握)

- 栈
  - 先进后出
- 队列
  - 先进先出

16.05_集合框架(用LinkedList模拟栈数据结构的集合并测试)(掌握)

- A:案例演示
  - 需求：请用LinkedList模拟栈数据结构的集合，并测试

- 创建一个类将Linked中的方法封装

```
public class Stack {
    private LinkedList list = new LinkedList();     //创建LinkedList对象

    public void in(Object obj) {
        list.addLast(obj);                          //封装addLast()方法
    }

    public Object out() {
        return list.removeLast();                   //封装removeLast()方法
    }

    public boolean isEmpty() {
        return list.isEmpty();                      //封装isEmpty()方法
    }
}
```

16.06_集合框架(泛型概述和基本使用)(掌握)

- A:泛型概述
- B:泛型好处
  - 提高安全性(将运行期的错误转换到编译期)
  - 省去强转的麻烦
- C:泛型基本使用
  - <>中放的必须是引用数据类型
- D:泛型使用注意事项
  - 前后的泛型必须一致,或者后面的泛型可以省略不写(1.7的新特性菱形泛型)

16.07_集合框架(ArrayList存储字符串和自定义对象并遍历泛型版)(掌握)

- A:案例演示
  - ArrayList存储字符串并遍历泛型版

16.08_集合框架(泛型的由来)(了解)

- A:案例演示
  - 泛型的由来:通过Object转型问题引入
  - 早期的Object类型可以接收任意的对象类型，但是在实际的使用中，会有类型转换的问题。也就存在这隐患，所以Java提供了泛型来解决这个安全问题。

16.09_集合框架(泛型类的概述及使用)(了解)

- A:泛型类概述
  - 把泛型定义在类上
- B:定义格式
  - public class 类名<泛型类型1,…>
- C:注意事项
  - 泛型类型必须是引用类型
- D:案例演示
  - 泛型类的使用

16.10_集合框架(泛型方法的概述和使用)(了解)

- A:泛型方法概述
  - 把泛型定义在方法上
- B:定义格式
  - public <泛型类型> 返回类型 方法名(泛型类型 变量名)
- C:案例演示
  - 泛型方法的使用

16.11_集合框架(泛型接口的概述和使用)(了解)

- A:泛型接口概述
  - 把泛型定义在接口上
- B:定义格式
  - public interface 接口名<泛型类型>
- C:案例演示
  - 泛型接口的使用

16.12_集合框架(泛型高级之通配符)(了解)

- A:泛型通配符<?>

    - 任意类型，如果没有明确，那么就是Object以及任意的Java类了
- B:? extends E
    - 向下限定，E及其子类
- C:? super E
    - 向上限定，E及其父类

16.13_集合框架(增强for的概述和使用)(掌握)

- A:增强for概述
    - 简化数组和Collection集合的遍历
- B:格式：

```
for(元素数据类型 变量 : 数组或者Collection集合) {
    使用变量即可，该变量就是元素
}
```

- C:案例演示
    - 数组，集合存储元素用增强for遍历
- D:好处
    - 简化遍历

16.14_集合框架(ArrayList存储字符串和自定义对象并遍历增强for版)(掌握)

- A:案例演示
    - ArrayList存储字符串并遍历增强for版

```
ArrayList<String> list = new ArrayList<>();
list.add("a");
list.add("b");
list.add("c");
list.add("d");

for(String s : list) {
    System.out.println(s);
}
```

16.15_集合框架(三种迭代的能否删除)(掌握)

- 普通for循环,可以删除,但是索引要--
- 迭代器,可以删除,但是必须使用迭代器自身的remove方法,否则会出现并发修改异常
- 增强for循环不能删除

16.16_集合框架(静态导入的概述和使用)(掌握)

- A:静态导入概述
- B:格式：
    - import static 包名….类名.方法名;
    - 可以直接导入到方法的级别
- C:注意事项
    - 方法必须是静态的,如果有多个同名的静态方法，容易不知道使用谁?这个时候要使用，必须加前缀。由此可见，意义不大，所以一般不用，但是要能看懂。

16.17_集合框架(可变参数的概述和使用)(掌握)

- A:可变参数概述
    - 定义方法的时候不知道该定义多少个参数
- B:格式
    - 修饰符 返回值类型 方法名(数据类型… 变量名){}
- C:注意事项：
    - 这里的变量其实是一个数组
    - 如果一个方法有可变参数，并且有多个参数，那么，可变参数肯定是最后一个

16.18_集合框架(Arrays工具类的asList()方法的使用)(掌握)

- A:案例演示
    - Arrays工具类的asList()方法的使用
    - Collection中toArray(T[] a)泛型版的集合转数组

16.19_集合框架(集合嵌套之ArrayList嵌套ArrayList)(掌握)

- A:案例演示
    - 集合嵌套之ArrayList嵌套ArrayList

16.20_day16总结

- 把今天的知识点总结一遍。

Set集合的基本特征是不记录添加顺序，不允许元素重复（想想是为什么）。最常用的实现类是HashSet.

17.01_集合框架(HashSet存储字符串并遍历)

HashSet类直接实现了Set接口， 其底层其实是包装了一个HashMap去实现的。HashSet采用HashCode算法来存取集合中的元素，因此具有比较好的读取和查找性能。

- A:Set集合概述及特点
  - 通过API查看即可
- B:案例演示
  - HashSet存储字符串并遍历

```
HashSet<String> hs = new HashSet<>();
boolean b1 = hs.add("a");
boolean b2 = hs.add("a");           //当存储不成功的时候,返回false

System.out.println(b1);
System.out.println(b2);
for(String s : hs) {
    System.out.println(s);
}
```

17.02_集合框架(HashSet存储自定义对象保证元素唯一性)

- A:案例演示
  - 存储自定义对象，并保证元素唯一性。

```
HashSet<Person> hs = new HashSet<>();
hs.add(new Person("张三", 23));
hs.add(new Person("张三", 23));
hs.add(new Person("李四", 23));
hs.add(new Person("李四", 23));
hs.add(new Person("王五", 23));
hs.add(new Person("赵六", 23));
```

- 重写hashCode()和equals()方法

17.03_集合框架(HashSet存储自定义对象保证元素唯一性图解及代码优化)

- A:画图演示
  - 画图说明比较过程
- B:代码优化
  - 为了减少比较，优化hashCode()代码写法。
  - 最终版就是自动生成即可。

17.04_集合框架(HashSet如何保证元素唯一性的原理)

- 1.HashSet原理
  - 我们使用Set集合都是需要去掉重复元素的, 如果在存储的时候逐个equals()比较, 效率较低,哈希算法提高了去重复的效率, 降低了使用equals()方法的次数
  - 当HashSet调用add()方法存储对象的时候, 先调用对象的hashCode()方法得到一个哈希值, 然后在集合中查找是否有哈希值相同的对象
    - 如果没有哈希值相同的对象就直接存入集合
    - 如果有哈希值相同的对象, 就和哈希值相同的对象逐个进行equals()比较,比较结果为false就存入, true则不存
- 2.将自定义类的对象存入HashSet去重复
  - 类中必须重写hashCode()和equals()方法
  - hashCode(): 属性相同的对象返回值必须相同, 属性不同的返回值尽量不同(提高效率)
  - equals(): 属性相同返回true, 属性不同返回false,返回false的时候存储

17.05_集合框架(LinkedHashSet的概述和使用)

1. LinkedHashSet概述：
   LinkedHashSet是具有可预知迭代顺序的Set接口的哈希表和链接列表实现。此实现与HashSet的不同之处在于，后者维护着一个运行于所有条目的双重链接列表。此链接列表定义了迭代顺序，该迭代顺序可为插入顺序或是访问顺序。

LinkedHashSet底层使用LinkedHashMap来保存所有元素，它继承与HashSet，其所有的方法操作上又与HashSet相同，因此LinkedHashSet 的实现上非常简单，只提供了四个构造方法，并通过传递一个标识参数，调用父类的构造器，底层构造一个LinkedHashMap来实现，在相关操作上与父类HashSet的操作相同，直接调用父类HashSet的方法即可

- A:LinkedHashSet的特点
- B:案例演示
  - LinkedHashSet的特点
    - 可以保证怎么存就怎么取

17.06_集合框架(产生10个1-20之间的随机数要求随机数不能重复)

- A:案例演示
  - 需求：编写一个程序，获取10个1至20的随机数，要求随机数不能重复。并把最终的随机数输出到控制台。

```
HashSet<Integer> hs = new HashSet<>();      //创建集合对象
Random r = new Random();                    //创建随机数对象

while(hs.size() < 10) {
    int num = r.nextInt(20) + 1;            //生成1到20的随机数
    hs.add(num);
```

```
}

for (Integer integer : hs) {                //遍历集合
    System.out.println(integer);            //打印每一个元素
}
```

17.07_集合框架(练习)

- 使用Scanner从键盘读取一行输入,去掉其中重复字符, 打印出不同的那些字符
    - aaaabbbcccddd

```
Scanner sc = new Scanner(System.in);            //创建键盘录入对象
System.out.println("请输入一行字符串:");
String line = sc.nextLine();                    //将键盘录入的字符串存储在line中
char[] arr = line.toCharArray();                //将字符串转换成字符数组
HashSet<Character> hs = new HashSet<>();        //创建HashSet集合对象

for(char c : arr) {                             //遍历字符数组
    hs.add(c);                                  //将字符数组中的字符添加到集合中
}

for (Character ch : hs) {                       //遍历集合
    System.out.println(ch);
}
```

17.08_集合框架(练习)

- 将集合中的重复元素去掉

```
public static void main(String[] args) {
    ArrayList<String> list = new ArrayList<>();
    list.add("a");
    list.add("a");
    list.add("a");
    list.add("b");
    list.add("b");
    list.add("b");
    list.add("b");
    list.add("c");
    list.add("c");
    list.add("c");
    list.add("c");

    System.out.println(list);
    System.out.println("去除重复后:");
    getSingle(list);
    System.out.println(list);
}

/*
 * 将集合中的重复元素去掉
 * 1,void
 * 2,List<String> list
 */

public static void getSingle(List<String> list) {
    LinkedHashSet<String> lhs = new LinkedHashSet<>();
    lhs.addAll(list);                                   //将list集合中的所有元素添加到lhs
    list.clear();                                       //清空原集合
    list.addAll(lhs);                                   //将去除重复的元素添回到list中
}
```

17.09_集合框架(TreeSet存储Integer类型的元素并遍历)

- A:案例演示
    - TreeSet存储Integer类型的元素并遍历

17.10_集合框架(TreeSet存储自定义对象)

- A:案例演示
    - 存储Person对象

17.11_集合框架(TreeSet保证元素唯一和自然排序的原理和图解)

- A:画图演示
    - TreeSet保证元素唯一和自然排序的原理和图解

17.12_集合框架(TreeSet存储自定义对象并遍历练习1)

- A:案例演示
  - TreeSet存储自定义对象并遍历练习1(按照姓名排序)

17.13_集合框架(TreeSet存储自定义对象并遍历练习2)

- A:案例演示
  - TreeSet存储自定义对象并遍历练习2(按照姓名的长度排序)

17.14_集合框架(TreeSet保证元素唯一和比较器排序的原理及代码实现)

- A:案例演示
  - TreeSet保证元素唯一和比较器排序的原理及代码实现

17.15_集合框架(TreeSet原理)

- 1.特点
  - TreeSet是用来排序的, 可以指定一个顺序, 对象存入之后会按照指定的顺序排列
- 2.使用方式
  - a.自然顺序(Comparable)
    - TreeSet类的add()方法中会把存入的对象提升为Comparable类型
    - 调用对象的compareTo()方法和集合中的对象比较
    - 根据compareTo()方法返回的结果进行存储
  - b.比较器顺序(Comparator)
    - 创建TreeSet的时候可以制定 一个Comparator
    - 如果传入了Comparator的子类对象, 那么TreeSet就会按照比较器中的顺序排序
    - add()方法内部会自动调用Comparator接口中compare()方法排序
    - 调用的对象是compare方法的第一个参数,集合中的对象是compare方法的第二个参数
  - c.两种方式的区别
    - TreeSet构造函数什么都不传, 默认按照类中Comparable的顺序(没有就报错ClassCastException)
    - TreeSet如果传入Comparator, 就优先按照Comparator

17.16_集合框架(练习)

- 在一个集合中存储了无序并且重复的字符串,定义一个方法,让其有序(字典顺序),而且还不能去除重复

```
public static void main(String[] args) {
    ArrayList<String> list = new ArrayList<>();
    list.add("ccc");
    list.add("ccc");
    list.add("aaa");
    list.add("aaa");
    list.add("bbb");
    list.add("ddd");
    list.add("ddd");

    sort(list);
    System.out.println(list);
}

/*
 * 对集合中的元素排序,并保留重复
 * 1,void
 * 2,List<String> list
 */
public static void sort(List<String> list) {
    TreeSet<String> ts = new TreeSet<>(new Comparator<String>() {       //定义比较器(new Comparator(){}是Comparator的子类对象)

        @Override
        public int compare(String s1, String s2) {                      //重写compare方法
            int num = s1.compareTo(s2);                                 //比较内容
            return num == 0 ? 1 : num;                                  //如果内容一样返回一个不为0的数字即可
        }
    });

    ts.addAll(list);                                                    //将list集合中的所有元素添加到ts中
    list.clear();                                                       //清空list
    list.addAll(ts);                                                    //将ts中排序并保留重复的结果在添加到list中
}
```

17.17_集合框架(练习)

- 从键盘接收一个字符串, 程序对其中所有字符进行排序,例如键盘输入: helloitcast程序打印:acehillostt

```
Scanner sc = new Scanner(System.in);            //创建键盘录入对象
System.out.println("请输入一行字符串:");
String line = sc.nextLine();                    //将键盘录入的字符串存储在line中
char[] arr = line.toCharArray();                //将字符串转换成字符数组
TreeSet<Character> ts = new TreeSet<>(new Comparator<Character>() {

    @Override
    public int compare(Character c1, Character c2) {
        //int num = c1.compareTo(c2);
        int num = c1 - c2;                  //自动拆箱
        return num == 0 ? 1 : num;
    }
});

for(char c : arr) {
    ts.add(c);
}

for(Character ch : ts) {
    System.out.print(ch);
}
```

17.18_集合框架(练习)

- 程序启动后, 可以从键盘输入接收多个整数, 直到输入quit时结束输入. 把所有输入的整数倒序排列打印. Scanner sc = new Scanner(System.in); //创建键盘录入对象 System.out.println("请输入:"); TreeSet ts = new TreeSet<>(new Comparator() {//将比较器传给TreeSet的构造方法

```
        @Override
        public int compare(Integer i1, Integer i2) {
            //int num = i2 - i1;                    //自动拆箱
            int num = i2.compareTo(i1);
            return num == 0 ? 1 : num;
        }
    });

    while(true) {
        String line = sc.nextLine();            //将键盘录入的字符串存储在line中
        if("quit".equals(line))                 //如果字符串常量和变量比较,常量放前面,这样不会出现空指针异常,变量里面可能存储null
            break;
        try {
            int num = Integer.parseInt(line);       //将数字字符串转换成数字
            ts.add(num);
        } catch (Exception e) {
            System.out.println("您录入的数据有误,请输入一个整数");
        }

    }

    for (Integer i : ts) {                      //遍历TreeSet集合
        System.out.println(i);
    }
```

17.19_集合框架(键盘录入学生信息按照总分排序后输出在控制台)

- A:案例演示
  - 需求：键盘录入5个学生信息(姓名,语文成绩,数学成绩,英语成绩),按照总分从高到低输出到控制台。 Scanner sc = new Scanner(System.in); System.out.println("请输入5个学生成绩格式是:(姓名,语文成绩,数学成绩,英语成绩)"); TreeSet ts = new TreeSet<>(new Comparator() { @Override public int compare(Student s1, Student s2) { int num = s2.getSum() - s1.getSum(); //根据学生的总成绩降序排列 return num == 0 ? 1 : num; } });

    ```
    while(ts.size() < 5) {
        String line = sc.nextLine();
        try {
            String[] arr = line.split(",");
            int chinese = Integer.parseInt(arr[1]);             //转换语文成绩
            int math = Integer.parseInt(arr[2]);                //转换数学成绩
            int english = Integer.parseInt(arr[3]);             //转换英语成绩
            ts.add(new Student(arr[0], chinese, math, english));
        } catch (Exception e) {
            System.out.println("录入格式有误,输入5个学生成绩格式是:(姓名,语文成绩,数学成绩,英语成绩");
        }

    }
    ```

```
System.out.println("排序后的学生成绩是:");
for (Student s : ts) {
    System.out.println(s);
}
```

17.20_day17总结

- 1.List
  - a.普通for循环, 使用get()逐个获取
  - b.调用iterator()方法得到Iterator, 使用hasNext()和next()方法
  - c.增强for循环, 只要可以使用Iterator的类都可以用
  - d.Vector集合可以使用Enumeration的hasMoreElements()和nextElement()方法
- 2.Set
  - a.调用iterator()方法得到Iterator, 使用hasNext()和next()方法
  - b.增强for循环, 只要可以使用Iterator的类都可以用
- 3.普通for循环,迭代器,增强for循环是否可以在遍历的过程中删除

18.01_集合框架(Map集合概述和特点)

- A:Map接口概述
  - 查看API可以知道：
    - 将键映射到值的对象
    - 一个映射不能包含重复的键
    - 每个键最多只能映射到一个值
- B:Map接口和Collection接口的不同
  - Map是双列的,Collection是单列的
  - Map的键唯一,Collection的子体系Set是唯一的
  - Map集合的数据结构值针对键有效，跟值无关;Collection集合的数据结构是针对元素有效

18.02_集合框架(Map集合的功能概述)

- A:Map集合的功能概述
  - a:添加功能
    - V put(K key,V value):添加元素。
      - 如果键是第一次存储，就直接存储元素，返回null
      - 如果键不是第一次存在，就用值把以前的值替换掉，返回以前的值
  - b:删除功能
    - void clear():移除所有的键值对元素
    - V remove(Object key)：根据键删除键值对元素，并把值返回
  - c:判断功能
    - boolean containsKey(Object key)：判断集合是否包含指定的键
    - boolean containsValue(Object value):判断集合是否包含指定的值
    - boolean isEmpty()：判断集合是否为空
  - d:获取功能
    - Set<Map.Entry<K,V>> entrySet():
    - V get(Object key):根据键获取值
    - Set keySet():获取集合中所有键的集合
    - Collection values():获取集合中所有值的集合
  - e:长度功能
    - int size()：返回集合中的键值对的个数

18.03_集合框架(Map集合的遍历之键找值)

- A:键找值思路：
  - 获取所有键的集合
  - 遍历键的集合，获取到每一个键
  - 根据键找值
- B:案例演示
  - Map集合的遍历之键找值

```
HashMap<String, Integer> hm = new HashMap<>();
hm.put("张三", 23);
hm.put("李四", 24);
hm.put("王五", 25);
hm.put("赵六", 26);

/*Set<String> keySet = hm.keySet();          //获取集合中所有的键
Iterator<String> it = keySet.iterator();    //获取迭代器
while(it.hasNext()) {                       //判断单列集合中是否有元素
    String key = it.next();                 //获取集合中的每一个元素,其实就是双列集合中的键
    Integer value = hm.get(key);            //根据键获取值
    System.out.println(key + "=" + value);  //打印键值对
}*/

for(String key : hm.keySet()) {             //增强for循环迭代双列集合第一种方式
    System.out.println(key + "=" + hm.get(key));
}
```

18.04_集合框架(Map集合的遍历之键值对对象找键和值)

- A:键值对对象找键和值思路：
  - 获取所有键值对对象的集合
  - 遍历键值对对象的集合，获取到每一个键值对对象
  - 根据键值对对象找键和值
- B:案例演示
  - Map集合的遍历之键值对对象找键和值

```
HashMap<String, Integer> hm = new HashMap<>();
hm.put("张三", 23);
hm.put("李四", 24);
hm.put("王五", 25);
hm.put("赵六", 26);
/*Set<Map.Entry<String, Integer>> entrySet = hm.entrySet(); //获取所有的键值对象的集合
Iterator<Entry<String, Integer>> it = entrySet.iterator();//获取迭代器
while(it.hasNext()) {
    Entry<String, Integer> en = it.next();              //获取键值对对象
    String key = en.getKey();                               //根据键值对对象获取键
    Integer value = en.getValue();                          //根据键值对对象获取值
    System.out.println(key + "=" + value);
}*/

for(Entry<String,Integer> en : hm.entrySet()) {
    System.out.println(en.getKey() + "=" + en.getValue());
}
```

C:源码分析

18.05_集合框架(HashMap集合键是Student值是String的案例)

- A:案例演示
  - HashMap集合键是Student值是String的案例

18.06_集合框架(LinkedHashMap的概述和使用)

- A:案例演示
  - LinkedHashMap的特点
    - 底层是链表实现的可以保证怎么存就怎么取

18.07_集合框架(TreeMap集合键是Student值是String的案例)

- A:案例演示
  - TreeMap集合键是Student值是String的案例

18.08_集合框架(统计字符串中每个字符出现的次数)

- A:案例演示
  - 需求：统计字符串中每个字符出现的次数 String str = "aaaabbbccccccccc"; char[] arr = str.toCharArray(); //将字符串转换成字符数组 HashMap<Character, Integer> hm = new HashMap<>(); //创建双列集合存储键和值

```
for(char c : arr) {                                 //遍历字符数组
    /*if(!hm.containsKey(c)) {                      //如果不包含这个键
        hm.put(c, 1);                               //就将键和值为1添加
    }else {                                         //如果包含这个键
        hm.put(c, hm.get(c) + 1);                   //就将键和值再加1添加进来
    }

    //hm.put(c, !hm.containsKey(c) ? 1 : hm.get(c) + 1);
    Integer i = !hm.containsKey(c) ? hm.put(c, 1) : hm.put(c, hm.get(c) + 1);
            }

for (Character key : hm.keySet()) {                 //遍历双列集合
    System.out.println(key + "=" + hm.get(key));
}
```

18.09_集合框架(集合嵌套之HashMap嵌套HashMap)

- A:案例演示
  - 集合嵌套之HashMap嵌套HashMap

18.10_集合框架(HashMap和Hashtable的区别)

- A:面试题
  - HashMap和Hashtable的区别
    - Hashtable是JDK1.0版本出现的,是线程安全的,效率低,HashMap是JDK1.2版本出现的,是线程不安全的,效率高
    - Hashtable不可以存储null键和null值,HashMap可以存储null键和null值
- B:案例演示
  - HashMap和Hashtable的区别

18.11_集合框架(Collections工具类的概述和常见方法讲解)

- A:Collections类概述
  - 针对集合操作 的工具类
- B:Collections成员方法

```
public static <T> void sort(List<T> list)
public static <T> int binarySearch(List<?> list,T key)
public static <T> T max(Collection<?> coll)
public static void reverse(List<?> list)
public static void shuffle(List<?> list)
```

18.12_集合框架(模拟斗地主洗牌和发牌)

- A:案例演示
  - 模拟斗地主洗牌和发牌，牌没有排序

```
//买一副扑克
String[] num = {"A","2","3","4","5","6","7","8","9","10","J","Q","K"};
String[] color = {"方片","梅花","红桃","黑桃"};
ArrayList<String> poker = new ArrayList<>();

for(String s1 : color) {
    for(String s2 : num) {
        poker.add(s1.concat(s2));
    }
}

poker.add("小王");
poker.add("大王");
//洗牌
Collections.shuffle(poker);
//发牌
ArrayList<String> gaojin = new ArrayList<>();
ArrayList<String> longwu = new ArrayList<>();
ArrayList<String> me = new ArrayList<>();
ArrayList<String> dipai = new ArrayList<>();

for(int i = 0; i < poker.size(); i++) {
    if(i >= poker.size() - 3) {
        dipai.add(poker.get(i));
    }else if(i % 3 == 0) {
        gaojin.add(poker.get(i));
    }else if(i % 3 == 1) {
        longwu.add(poker.get(i));
    }else {
        me.add(poker.get(i));
    }
}

//看牌

System.out.println(gaojin);
System.out.println(longwu);
System.out.println(me);
System.out.println(dipai);
```

18.13_集合框架(模拟斗地主洗牌和发牌并对牌进行排序的原理图解)

- A:画图演示
  - 画图说明排序原理

18.14_集合框架(模拟斗地主洗牌和发牌并对牌进行排序的代码实现)

- A:案例演示
  - 模拟斗地主洗牌和发牌并对牌进行排序的代码实现

```
//买一副牌
String[] num = {"3","4","5","6","7","8","9","10","J","Q","K","A","2"};
String[] color = {"方片","梅花","红桃","黑桃"};
HashMap<Integer, String> hm = new HashMap<>();          //存储索引和扑克牌
ArrayList<Integer> list = new ArrayList<>();            //存储索引
int index = 0;                                          //索引的开始值
for(String s1 : num) {
    for(String s2 : color) {
        hm.put(index, s2.concat(s1));                   //将索引和扑克牌添加到HashMap中
        list.add(index);                                //将索引添加到ArrayList集合中
        index++;
    }
```

```
    }
    hm.put(index, "小王");
    list.add(index);
    index++;
    hm.put(index, "大王");
    list.add(index);
    //洗牌
    Collections.shuffle(list);
    //发牌
    TreeSet<Integer> gaojin = new TreeSet<>();
    TreeSet<Integer> longwu = new TreeSet<>();
    TreeSet<Integer> me = new TreeSet<>();
    TreeSet<Integer> dipai = new TreeSet<>();

    for(int i = 0; i < list.size(); i++) {
        if(i >= list.size() - 3) {
            dipai.add(list.get(i));                         //将list集合中的索引添加到TreeSet集合中会自动排序
        }else if(i % 3 == 0) {
            gaojin.add(list.get(i));
        }else if(i % 3 == 1) {
            longwu.add(list.get(i));
        }else {
            me.add(list.get(i));
        }
    }

    //看牌
    lookPoker("高进", gaojin, hm);
    lookPoker("龙五", longwu, hm);
    lookPoker("冯佳", me, hm);
    lookPoker("底牌", dipai, hm);

}

public static void lookPoker(String name,TreeSet<Integer> ts,HashMap<Integer, String> hm) {
    System.out.print(name + "的牌是:");
    for (Integer index : ts) {
        System.out.print(hm.get(index) + " ");
    }

    System.out.println();
}
```

18.15_集合框架(泛型固定下边界)

- ? super E

18.16_day18总结

- 把今天的知识点总结一遍。

19.01_异常(异常的概述和分类)

- A:异常的概述
  - 异常就是Java程序在运行过程中出现的错误。
- B:异常的分类
  - 通过API查看Throwable
  - Error
    - 服务器宕机,数据库崩溃等
  - Exception C:异常的继承体系
  - Throwable
    - Error
    - Exception
      - RuntimeException

19.02_异常(JVM默认是如何处理异常的)

- A:JVM默认是如何处理异常的
  - main函数收到这个问题时,有两种处理方式:
  - a:自己将该问题处理,然后继续运行
  - b:自己没有针对的处理方式,只有交给调用main的jvm来处理
  - jvm有一个默认的异常处理机制,就将该异常进行处理.
  - 并将该异常的名称,异常的信息.异常出现的位置打印在了控制台上,同时将程序停止运行
- B:案例演示
  - JVM默认如何处理异常

19.03_异常(try...catch的方式处理异常1)

- A:异常处理的两种方式
  - a:try…catch…finally
    - try catch
    - try catch finally
    - try finally
  - b:throws
- B:try...catch处理异常的基本格式
  - try…catch…finally
- C:案例演示
  - try...catch的方式处理1个异常

19.04_异常(try...catch的方式处理异常2)

- A:案例演示
  - try...catch的方式处理多个异常
  - JDK7以后处理多个异常的方式及注意事项

19.05_异常(编译期异常和运行期异常的区别)

- A:编译期异常和运行期异常的区别
  - Java中的异常被分为两大类：编译时异常和运行时异常。
  - 所有的RuntimeException类及其子类的实例被称为运行时异常，其他的异常就是编译时异常
  - 编译时异常
    - Java程序必须显示处理，否则程序就会发生错误，无法通过编译
  - 运行时异常
    - 无需显示处理，也可以和编译时异常一样处理
- B:案例演示
  - 编译期异常和运行期异常的区别

19.06_异常(Throwable的几个常见方法)

- A:Throwable的几个常见方法
  - a:getMessage()
    - 获取异常信息，返回字符串。
  - b:toString()
    - 获取异常类名和异常信息，返回字符串。
  - c:printStackTrace()
    - 获取异常类名和异常信息，以及异常出现在程序中的位置。返回值void。
- B:案例演示
  - Throwable的几个常见方法的基本使用

19.07_异常(throws的方式处理异常)

- A:throws的方式处理异常
  - 定义功能方法时，需要把出现的问题暴露出来让调用者去处理。
  - 那么就通过throws在方法上标识。
- B:案例演示
  - 举例分别演示编译时异常和运行时异常的抛出

19.08_异常(throw的概述以及和throws的区别)

- A:throw的概述
  - 在功能方法内部出现某种情况，程序不能继续运行，需要进行跳转时，就用throw把异常对象抛出。
- B:案例演示
  - 分别演示编译时异常对象和运行时异常对象的抛出
- C:throws和throw的区别
  - a:throws
    - 用在方法声明后面，跟的是异常类名
    - 可以跟多个异常类名，用逗号隔开
    - 表示抛出异常，由该方法的调用者来处理
  - b:throw
    - 用在方法体内，跟的是异常对象名
    - 只能抛出一个异常对象名
    - 表示抛出异常，由方法体内的语句处理

19.09_异常(finally关键字的特点及作用)

- A:finally的特点
  - 被finally控制的语句体一定会执行
  - 特殊情况：在执行到finally之前jvm退出了(比如System.exit(0))
- B:finally的作用
  - 用于释放资源，在IO流操作和数据库操作中会见到
- C:案例演示
  - finally关键字的特点及作用

1.final：如果一个类被final修饰，意味着该类不能派生出新的子类，不能作为父类被继承。因此一个类不能被声明为abstract，又被声明为final。将变量或方法声明为final。可以保证他们在使用的时候不被改变。其初始化可以在两个地方：一是其定义的地方，也就是在final变量在定义的时候就对其赋值；二是在构造函数中。这两个地方只能选其中的一个，要么在定义的时候给值，要么在构造函数中给值。被声明为final的方法也只能使用，不能重写。

2.finally：在异常处理的时候，提供finally块来执行任何的清除操作。如果抛出一个异常，那么相匹配的catch字句就会执行，然后控制就会进入finally块，前提是有finally块。

3.finalize：finalize是方法名，java技术允许使用finalize()方法在垃圾收集器将对象从内存中清除出去之前做必要的清理工作。这个方法是在垃圾收集器确认一个对象没有被引用时对这个对象调用的。它是在Object类中定义的，因此，所有的类都继承了它。子类覆盖finalize()方法已整理系统资源或者执行其他清理工作。finalize()方法是在垃圾收集器删除对象之前对这个对象调用的。

19.10_异常(finally关键字的面试题)

- A:面试题1
  - final,finally和finalize的区别
- B:面试题2
  - 如果catch里面有return语句，请问finally的代码还会执行吗?如果会，请问是在return前还是return后。

19.11_异常(自定义异常概述和基本使用)

- A:为什么需要自定义异常
  - 举例：人的年龄
- B:自定义异常概述
  - 继承自Exception
  - 继承自RuntimeException
- C:案例演示
  - 自定义异常的基本使用

19.12_异常(异常的注意事项及如何使用异常处理)

- A:异常注意事项
  - a:子类重写父类方法时，子类的方法必须抛出相同的异常或父类异常的子类。(父亲坏了,儿子不能比父亲更坏)
  - b:如果父类抛出了多个异常,子类重写父类时,只能抛出相同的异常或者是他的子集,子类不能抛出父类没有的异常
  - c:如果被重写的方法没有异常抛出,那么子类的方法绝对不可以抛出异常,如果子类方法内有异常发生,那么子类只能try,不能throws
- B:如何使用异常处理
  - 原则:如果该功能内部可以将问题处理,用try,如果处理不了,交由调用者处理,这是用throws
  - 区别:
    - 后续程序需要继续运行就try
    - 后续程序不需要继续运行就throws
  - 如果JDK没有提供对应的异常，需要自定义异常。

19.13_异常(练习)

- 键盘录入一个int类型的整数,对其求二进制表现形式
  - 如果录入的整数过大,给予提示,录入的整数过大请重新录入一个整数BigInteger

- 如果录入的是小数,给予提示,录入的是小数,请重新录入一个整数
- 如果录入的是其他字符,给予提示,录入的是非法字符,请重新录入一个整数

19.14_File类(File类的概述和构造方法)

- A:File类的概述
  - File更应该叫做一个路径
    - 文件路径或者文件夹路径
    - 路径分为绝对路径和相对路径
    - 绝对路径是一个固定的路径,从盘符开始
    - 相对路径相对于某个位置,在eclipse下是指当前项目下,在dos下
  - 查看API指的是当前路径
  - 文件和目录路径名的抽象表示形式
- B:构造方法
  - File(String pathname)：根据一个路径得到File对象
  - File(String parent, String child):根据一个目录和一个子文件/目录得到File对象
  - File(File parent, String child):根据一个父File对象和一个子文件/目录得到File对象
- C:案例演示
  - File类的构造方法

19.15_File类(File类的创建功能)

- A:创建功能
  - public boolean createNewFile():创建文件 如果存在这样的文件，就不创建了
  - public boolean mkdir():创建文件夹 如果存在这样的文件夹，就不创建了
  - public boolean mkdirs():创建文件夹,如果父文件夹不存在，会帮你创建出来
- B:案例演示
  - File类的创建功能
  - 注意事项：
    - 如果你创建文件或者文件夹忘了写盘符路径，那么，默认在项目路径下。

19.16_File类(File类的重命名和删除功能)

- A:重命名和删除功能
  - public boolean renameTo(File dest):把文件重命名为指定的文件路径
  - public boolean delete():删除文件或者文件夹
- B:重命名注意事项
  - 如果路径名相同，就是改名。
  - 如果路径名不同，就是改名并剪切。
- C:删除注意事项：
  - Java中的删除不走回收站。
  - 要删除一个文件夹，请注意该文件夹内不能包含文件或者文件夹

19.17_File类(File类的判断功能)

- A:判断功能
  - public boolean isDirectory():判断是否是目录
  - public boolean isFile():判断是否是文件
  - public boolean exists():判断是否存在
  - public boolean canRead():判断是否可读
  - public boolean canWrite():判断是否可写
  - public boolean isHidden():判断是否隐藏
- B:案例演示
  - File类的判断功能

19.18_File类(File类的获取功能)

- A:获取功能
  - public String getAbsolutePath()：获取绝对路径
  - public String getPath():获取路径
  - public String getName():获取名称
  - public long length():获取长度。字节数
  - public long lastModified():获取最后一次的修改时间，毫秒值
  - public String[] list():获取指定目录下的所有文件或者文件夹的名称数组
  - public File[] listFiles():获取指定目录下的所有文件或者文件夹的File数组
- B:案例演示
  - File类的获取功能

19.19_File类(输出指定目录下指定后缀的文件名)

- A:案例演示
  - 需求：判断E盘目录下是否有后缀名为.jpg的文件，如果有，就输出该文件名称

19.20_File类(文件名称过滤器的概述及使用)

- A:文件名称过滤器的概述
  - public String[] list(FilenameFilter filter)
  - public File[] listFiles(FileFilter filter)
- B:文件名称过滤器的使用
  - 需求：判断E盘目录下是否有后缀名为.jpg的文件，如果有，就输出该文件名称
- C:源码分析
  - 带文件名称过滤器的list()方法的源码

19.21_File类(递归)

- 5的阶乘

19.22_day19总结

把今天的知识点总结一遍。

20.01_IO流(IO流概述及其分类)

- 1.概念
  - IO流用来处理设备之间的数据传输
  - Java对数据的操作是通过流的方式
  - Java用于操作流的类都在IO包中
  - 流按流向分为两种：输入流，输出流。
  - 流按操作类型分为两种：
    - 字节流：字节流可以操作任何数据,因为在计算机中任何数据都是以字节的形式存储的
    - 字符流：字符流只能操作纯字符数据，比较方便。
- 2.IO流常用父类
  - 字节流的抽象父类：
    - InputStream
    - OutputStream
  - 字符流的抽象父类：
    - Reader
    - Writer
- 3.IO程序书写
  - 使用前，导入IO包中的类
  - 使用时，进行IO异常处理
  - 使用后，释放资源

20.02_IO流(FileInputStream)

- read()一次读取一个字节

```
FileInputStream fis = new FileInputStream("aaa.txt");   //创建一个文件输入流对象,并关联aaa.txt
int b;                                                  //定义变量,记录每次读到的字节
while((b = fis.read()) != -1) {                         //将每次读到的字节赋值给b并判断是否是-1
    System.out.println(b);                              //打印每一个字节
}

fis.close();                                            //关闭流释放资源
```

20.03_IO流(read()方法返回值为什么是int)

- read()方法读取的是一个字节,为什么返回是int,而不是byte

```
因为字节输入流可以操作任意类型的文件,比如图片音频等,这些文件底层都是以二进制形式的存储的,如果每次读取都返回byte,有可能在读到中间的时候遇到111111111
那么这11111111是byte类型的-1,我们的程序是遇到-1就会停止不读了,后面的数据就读不到了,所以在读取的时候用int类型接收,如果11111111会在其前面补上
24个0凑足4个字节,那么byte类型的-1就变成int类型的255了这样可以保证整个数据读完,而结束标记的-1就是int类型
```

20.04_IO流(FileOutputStream)

- write()一次写出一个字节

```
FileOutputStream fos = new FileOutputStream("bbb.txt"); //如果没有bbb.txt,会创建出一个
//fos.write(97);                        //虽然写出的是一个int数,但是在写出的时候会将前面的24个0去掉,所以写出的一个byte
fos.write(98);
fos.write(99);
fos.close();
```

20.05_IO流(FileOutputStream追加)

- A:案例演示
  - FileOutputStream的构造方法写出数据如何实现数据的追加写入

```
FileOutputStream fos = new FileOutputStream("bbb.txt",true);    //如果没有bbb.txt,会创建出一个
//fos.write(97);                        //虽然写出的是一个int数,但是在写出的时候会将前面的24个0去掉,所以写出的一个byte
fos.write(98);
fos.write(99);
fos.close();
```

20.06_IO流(拷贝图片)

- FileInputStream读取
- FileOutputStream写出

```
FileInputStream fis = new FileInputStream("致青春.mp3");   //创建输入流对象,关联致青春.mp3
FileOutputStream fos = new FileOutputStream("copy.mp3");//创建输出流对象,关联copy.mp3

int b;
while((b = fis.read()) != -1) {
```

```
    fos.write(b);
}

fis.close();
fos.close();
```

20.07_IO流(拷贝音频文件画原理图)

- A:案例演示
  - 字节流一次读写一个字节复制音频
- 弊端:效率太低

20.08_IO流(字节数组拷贝之available()方法)

- A:案例演示
  - int read(byte[] b):一次读取一个字节数组
  - write(byte[] b):一次写出一个字节数组
  - available()获取读的文件所有的字节个数
- 弊端:有可能会内存溢出

```
FileInputStream fis = new FileInputStream("致青春.mp3");
FileOutputStream fos = new FileOutputStream("copy.mp3");
byte[] arr = new byte[fis.available()];                 //根据文件大小做一个字节数组
fis.read(arr);                                          //将文件上的所有字节读取到数组中
fos.write(arr);                                         //将数组中的所有字节一次写到了文件上
fis.close();
fos.close();
```

20.09_IO流(定义小数组)

- write(byte[] b)
- write(byte[] b, int off, int len)写出有效的字节个数

20.10_IO流(定义小数组的标准格式)

- A:案例演示
  - 字节流一次读写一个字节数组复制图片和视频 FileInputStream fis = new FileInputStream("致青春.mp3"); FileOutputStream fos = new FileOutputStream("copy.mp3"); int len; byte[] arr = new byte[1024 * 8]; //自定义字节数组

    while((len = fis.read(arr)) != -1) { //fos.write(arr); fos.write(arr, 0, len); //写出字节数组写出有效个字节个数 }

    fis.close(); fos.close();

20.11_IO流(BufferedInputStream和BufferOutputStream拷贝)

- A:缓冲思想
  - 字节流一次读写一个数组的速度明显比一次读写一个字节的速度快很多，
  - 这是加入了数组这样的缓冲区效果，java本身在设计的时候，
  - 也考虑到了这样的设计思想(装饰设计模式后面讲解)，所以提供了字节缓冲区流
- B.BufferedInputStream
  - BufferedInputStream内置了一个缓冲区(数组)
  - 从BufferedInputStream中读取一个字节时
  - BufferedInputStream会一次性从文件中读取8192个, 存在缓冲区中, 返回给程序一个
  - 程序再次读取时, 就不用找文件了, 直接从缓冲区中获取
  - 直到缓冲区中所有的都被使用过, 才重新从文件中读取8192个
- C.BufferedOutputStream
  - BufferedOutputStream也内置了一个缓冲区(数组)
  - 程序向流中写出字节时, 不会直接写到文件, 先写到缓冲区中
  - 直到缓冲区写满, BufferedOutputStream才会把缓冲区中的数据一次性写到文件里
- D.拷贝的代码

```
FileInputStream fis = new FileInputStream("致青春.mp3");           //创建文件输入流对象,关联致青春.mp3
BufferedInputStream bis = new BufferedInputStream(fis);         //创建缓冲区对fis装饰
FileOutputStream fos = new FileOutputStream("copy.mp3");        //创建输出流对象,关联copy.mp3
BufferedOutputStream bos = new BufferedOutputStream(fos);       //创建缓冲区对fos装饰

int b;
while((b = bis.read()) != -1) {
    bos.write(b);
}

bis.close();                        //只关装饰后的对象即可
bos.close();
```

- E.小数组的读写和带Buffered的读取哪个更快?
    - 定义小数组如果是8192个字节大小和Buffered比较的话
    - 定义小数组会略胜一筹,因为读和写操作的是同一个数组
    - 而Buffered操作的是两个数组

20.12_IO流(flush和close方法的区别)

- flush()方法
    - 用来刷新缓冲区的,刷新后可以再次写出
- close()方法
    - 用来关闭流释放资源的的,如果是带缓冲区的流对象的close()方法,不但会关闭流,还会再关闭流之前刷新缓冲区,关闭后不能再写出

20.13_IO流(字节流读写中文)

- 字节流读取中文的问题
    - 字节流在读中文的时候有可能会读到半个中文,造成乱码
- 字节流写出中文的问题
    - 字节流直接操作的字节,所以写出中文必须将字符串转换成字节数组
    - 写出回车换行 write("\r\n".getBytes());

20.14_IO流(流的标准处理异常代码1.6版本及其以前)

- try finally嵌套

```
FileInputStream fis = null;
FileOutputStream fos = null;
try {
    fis = new FileInputStream("aaa.txt");
    fos = new FileOutputStream("bbb.txt");
    int b;
    while((b = fis.read()) != -1) {
        fos.write(b);
    }
} finally {
    try {
        if(fis != null)
            fis.close();
    }finally {
        if(fos != null)
            fos.close();
    }
}
```

20.15_IO流(流的标准处理异常代码1.7版本)

- try close

```
try(
    FileInputStream fis = new FileInputStream("aaa.txt");
    FileOutputStream fos = new FileOutputStream("bbb.txt");
    MyClose mc = new MyClose();
){
    int b;
    while((b = fis.read()) != -1) {
        fos.write(b);
    }
}
```

- 原理
    - 在try()中创建的流对象必须实现了AutoCloseable这个接口,如果实现了,在try后面的{}(读写代码)执行后就会自动调用,流对象的close方法将流关掉

20.16_IO流(图片加密)

- 给图片加密

```
BufferedInputStream bis = new BufferedInputStream(new FileInputStream("a.jpg"));
BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream("b.jpg"));

int b;
while((b = bis.read()) != -1) {
    bos.write(b ^ 123);
}
```

```
bis.close();
bos.close();
```

20.17_IO流(拷贝文件)

- 在控制台录入文件的路径,将文件拷贝到当前项目下

```
Scanner sc = new Scanner(System.in);
System.out.println("请输入一个文件路径");
String line = sc.nextLine();                //将键盘录入的文件路径存储在line中
File file = new File(line);                 //封装成File对象
FileInputStream fis = new FileInputStream(file);
FileOutputStream fos = new FileOutputStream(file.getName());

int len;
byte[] arr = new byte[8192];                //定义缓冲区
while((len = fis.read(arr)) != -1) {
    fos.write(arr,0,len);
}

fis.close();
fos.close();
```

20.18_IO流(录入数据拷贝到文件)

- 将键盘录入的数据拷贝到当前项目下的text.txt文件中,键盘录入数据当遇到quit时就退出

```
Scanner sc = new Scanner(System.in);
FileOutputStream fos = new FileOutputStream("text.txt");
System.out.println("请输入:");
while(true) {
    String line = sc.nextLine();
    if("quit".equals(line))
        break;
    fos.write(line.getBytes());
    fos.write("\r\n".getBytes());
}

fos.close();
```

20.19_day20总结

- 把今天的知识点总结一遍。

21.01_IO流(字符流FileReader)

- 1.字符流是什么
  - 字符流是可以直接读写字符的IO流
  - 字符流读取字符, 就要先读取到字节数据, 然后转为字符. 如果要写出字符, 需要把字符转为字节再写出.
- 2.FileReader
  - FileReader类的read()方法可以按照字符大小读取

```
FileReader fr = new FileReader("aaa.txt");              //创建输入流对象,关联aaa.txt
int ch;
while((ch = fr.read()) != -1) {                         //将读到的字符赋值给ch
    System.out.println((char)ch);                       //将读到的字符强转后打印
}

fr.close();                                             //关流
```

21.02_IO流(字符流FileWriter)

- FileWriter类的write()方法可以自动把字符转为字节写出

```
FileWriter fw = new FileWriter("aaa.txt");
fw.write("aaa");
fw.close();
```

21.03_IO流(字符流的拷贝)

```
FileReader fr = new FileReader("a.txt");
FileWriter fw = new FileWriter("b.txt");

int ch;
while((ch = fr.read()) != -1) {
    fw.write(ch);
}

fr.close();
fw.close();
```

21.04_IO流(什么情况下使用字符流)

- 字符流也可以拷贝文本文件, 但不推荐使用. 因为读取时会把字节转为字符, 写出时还要把字符转回字节.
- 程序需要读取一段文本, 或者需要写出一段文本的时候可以使用字符流
- 读取的时候是按照字符的大小读取的,不会出现半个中文
- 写出的时候可以直接将字符串写出,不用转换为字节数组

21.05_IO流(字符流是否可以拷贝非纯文本的文件)

- 不可以拷贝非纯文本的文件
- 因为在读的时候会将字节转换为字符,在转换过程中,可能找不到对应的字符,就会用?代替,写出的时候会将字符转换成字节写出去
- 如果是?,直接写出,这样写出之后的文件就乱了,看不了了

21.06_IO流(自定义字符数组的拷贝)

```
FileReader fr = new FileReader("aaa.txt");          //创建字符输入流,关联aaa.txt
FileWriter fw = new FileWriter("bbb.txt");          //创建字符输出流,关联bbb.txt

int len;
char[] arr = new char[1024*8];                      //创建字符数组
while((len = fr.read(arr)) != -1) {                 //将数据读到字符数组中
    fw.write(arr, 0, len);                          //从字符数组将数据写到文件上
}

fr.close();                                         //关流释放资源
fw.close();
```

21.07_IO流(带缓冲的字符流)

- BufferedReader的read()方法读取字符时会一次读取若干字符到缓冲区, 然后逐个返回给程序, 降低读取文件的次数, 提高效率
- BufferedWriter的write()方法写出字符时会先写到缓冲区, 缓冲区写满时才会写到文件, 降低写文件的次数, 提高效率

```
BufferedReader br = new BufferedReader(new FileReader("aaa.txt"));  //创建字符输入流对象,关联aaa.txt
BufferedWriter bw = new BufferedWriter(new FileWriter("bbb.txt"));  //创建字符输出流对象,关联bbb.txt

int ch;
```

```
while((ch = br.read()) != -1) {     //read一次,会先将缓冲区读满,从缓冲去中一个一个的返给临时变量ch
    bw.write(ch);                   //write一次,是将数据装到字符数组,装满后再一起写出去
}

br.close();                         //关流
bw.close();
```

21.08_IO流(readLine()和newLine()方法)

- BufferedReader的readLine()方法可以读取一行字符(不包含换行符号)
- BufferedWriter的newLine()可以输出一个跨平台的换行符号"\r\n"

```
BufferedReader br = new BufferedReader(new FileReader("aaa.txt"));
BufferedWriter bw = new BufferedWriter(new FileWriter("bbb.txt"));
String line;
while((line = br.readLine()) != null) {
    bw.write(line);
    //bw.write("\r\n");                 //只支持windows系统
    bw.newLine();                       //跨平台的
}

br.close();
bw.close();
```

21.09_IO流(将文本反转)

- 将一个文本文档上的文本反转,第一行和倒数第一行交换,第二行和倒数第二行交换

21.10_IO流(LineNumberReader)

- LineNumberReader是BufferedReader的子类, 具有相同的功能, 并且可以统计行号
  - 调用getLineNumber()方法可以获取当前行号
  - 调用setLineNumber()方法可以设置当前行号

```
LineNumberReader lnr = new LineNumberReader(new FileReader("aaa.txt"));
String line;
lnr.setLineNumber(100);                                 //设置行号
while((line = lnr.readLine()) != null) {
    System.out.println(lnr.getLineNumber() + ":" + line);//获取行号
}

lnr.close();
```

21.11_IO流(装饰设计模式)

```
interface Coder {
    public void code();
}

class Student implements Coder {

    @Override
    public void code() {
        System.out.println("javase");
        System.out.println("javaweb");
    }

}

class HeiMaStudent implements Coder {
    private Student s;                      //获取到被包装的类的引用
    public ItcastStudent(Student s) {       //通过构造函数创建对象的时候,传入被包装的对象
        this.s = s;
    }
    @Override
    public void code() {                    //对其原有功能进行升级
        s.code();
        System.out.println("数据库");
        System.out.println("ssh");
        System.out.println("安卓");
        System.out.println(".....");
    }

}
```

21.12_IO流(使用指定的码表读写字符)

- FileReader是使用默认码表读取文件, 如果需要使用指定码表读取, 那么可以使用InputStreamReader(字节流,编码表)
- FileWriter是使用默认码表写出文件, 如果需要使用指定码表写出, 那么可以使用OutputStreamWriter(字节流,编码表)

```
BufferedReader br =                                     //高效的用指定的编码表读
        new BufferedReader(new InputStreamReader(new FileInputStream("UTF-8.txt"), "UTF-8"));
BufferedWriter bw =                                     //高效的用指定的编码表写
        new BufferedWriter(new OutputStreamWriter(new FileOutputStream("GBK.txt"), "GBK"));
int ch;
while((ch = br.read()) != -1) {
    bw.write(ch);
}

br.close();
bw.close();
```

21.13_IO流(转换流图解)

- 画图分析转换流

21.14_IO流(获取文本上字符出现的次数)

- 获取一个文本上每个字符出现的次数,将结果写在times.txt上

21.15_IO流(试用版软件)

- 当我们下载一个试用版软件,没有购买正版的时候,每执行一次就会提醒我们还有多少次使用机会用学过的IO流知识,模拟试用版软件,试用10次机会,执行一次就提示一次您还有几次机会,如果次数到了提示请购买正版

21.16_File类(递归)

- 5的阶乘

21.17_File类(练习)

- 需求:从键盘输入接收一个文件夹路径,打印出该文件夹下所有的.java文件名

21.18_IO流(总结)

- 1.会用BufferedReader读取GBK码表和UTF-8码表的字符
- 2.会用BufferedWriter写出字符到GBK码表和UTF-8码表的文件中
- 3.会使用BufferedReader从键盘读取一行

22.01_IO流(序列流)(了解)

- 1.什么是序列流
  - 序列流可以把多个字节输入流整合成一个, 从序列流中读取数据时, 将从被整合的第一个流开始读, 读完一个之后继续读第二个, 以此类推.
- 2.使用方式
  - 整合两个: SequenceInputStream(InputStream, InputStream)

```
FileInputStream fis1 = new FileInputStream("a.txt");            //创建输入流对象,关联a.txt
FileInputStream fis2 = new FileInputStream("b.txt");            //创建输入流对象,关联b.txt
SequenceInputStream sis = new SequenceInputStream(fis1, fis2);  //将两个流整合成一个流
FileOutputStream fos = new FileOutputStream("c.txt");           //创建输出流对象,关联c.txt

int b;
while((b = sis.read()) != -1) {                                 //用整合后的读
    fos.write(b);                                               //写到指定文件上
}

sis.close();
fos.close();
```

22.02_IO流(序列流整合多个)(了解)

- 整合多个: SequenceInputStream(Enumeration)

```
FileInputStream fis1 = new FileInputStream("a.txt");    //创建输入流对象,关联a.txt
FileInputStream fis2 = new FileInputStream("b.txt");    //创建输入流对象,关联b.txt
FileInputStream fis3 = new FileInputStream("c.txt");    //创建输入流对象,关联c.txt
Vector<InputStream> v = new Vector<>();                 //创建vector集合对象
v.add(fis1);                                            //将流对象添加
v.add(fis2);
v.add(fis3);
Enumeration<InputStream> en = v.elements();             //获取枚举引用
SequenceInputStream sis = new SequenceInputStream(en);  //传递给SequenceInputStream构造
FileOutputStream fos = new FileOutputStream("d.txt");
int b;
while((b = sis.read()) != -1) {
    fos.write(b);
}

sis.close();
fos.close();
```

22.03_IO流(内存输出流*****)(掌握)

- 1.什么是内存输出流
  - 该输出流可以向内存中写数据, 把内存当作一个缓冲区, 写出之后可以一次性获取出所有数据
- 2.使用方式
  - 创建对象: new ByteArrayOutputStream()
  - 写出数据: write(int), write(byte[])
  - 获取数据: toByteArray()

```
FileInputStream fis = new FileInputStream("a.txt");
ByteArrayOutputStream baos = new ByteArrayOutputStream();
int b;
while((b = fis.read()) != -1) {
    baos.write(b);
}

//byte[] newArr = baos.toByteArray();               //将内存缓冲区中所有的字节存储在newArr中
//System.out.println(new String(newArr));
System.out.println(baos);
fis.close();
```

22.04_IO流(内存输出流之黑马面试题)(掌握)

- 定义一个文件输入流,调用read(byte[] b)方法,将a.txt文件中的内容打印出来(byte数组大小限制为5)

```
FileInputStream fis = new FileInputStream("a.txt");             //创建字节输入流,关联a.txt
ByteArrayOutputStream baos = new ByteArrayOutputStream();       //创建内存输出流
byte[] arr = new byte[5];                                       //创建字节数组,大小为5
int len;
while((len = fis.read(arr)) != -1) {                            //将文件上的数据读到字节数组中
    baos.write(arr, 0, len);                                    //将字节数组的数据写到内存缓冲区中
}
System.out.println(baos);                                       //将内存缓冲区的内容转换为字符串打印
```

```
    fis.close();
```

22.05_IO流(对象操作流ObjecOutputStream)(了解)

- 1.什么是对象操作流
  - 该流可以将一个对象写出, 或者读取一个对象到程序中. 也就是执行了序列化和反序列化的操作.
- 2.使用方式
  - 写出: new ObjectOutputStream(OutputStream), writeObject()

```
public class Demo3_ObjectOutputStream {

    /**
     * @param args
     * @throws IOException
     * 将对象写出,序列化
     */
    public static void main(String[] args) throws IOException {
        Person p1 = new Person("张三", 23);
        Person p2 = new Person("李四", 24);
//      FileOutputStream fos = new FileOutputStream("e.txt");
//      fos.write(p1);
//      FileWriter fw = new FileWriter("e.txt");
//      fw.write(p1);
        //无论是字节输出流,还是字符输出流都不能直接写出对象
        ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("e.txt"));//创建对象输出流
        oos.writeObject(p1);
        oos.writeObject(p2);
        oos.close();
    }

}
```

22.06_IO流(对象操作流ObjectInputStream)(了解)

- 读取: new ObjectInputStream(InputStream), readObject()

```
public class Demo3_ObjectInputStream {

    /**
     * @param args
     * @throws IOException
     * @throws ClassNotFoundException
     * @throws FileNotFoundException
     * 读取对象,反序列化
     */
    public static void main(String[] args) throws IOException, ClassNotFoundException {
        ObjectInputStream ois = new ObjectInputStream(new FileInputStream("e.txt"));
        Person p1 = (Person) ois.readObject();
        Person p2 = (Person) ois.readObject();
        System.out.println(p1);
        System.out.println(p2);
        ois.close();
    }

}
```

22.07_IO流(对象操作流优化)(了解)

* 将对象存储在集合中写出

```
Person p1 = new Person("张三", 23);
Person p2 = new Person("李四", 24);
Person p3 = new Person("马哥", 18);
Person p4 = new Person("辉哥", 20);

ArrayList<Person> list = new ArrayList<>();
list.add(p1);
list.add(p2);
list.add(p3);
list.add(p4);

ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("f.txt"));
oos.writeObject(list);                                  //写出集合对象

oos.close();
```

- 读取到的是一个集合对象

```
ObjectInputStream ois = new ObjectInputStream(new FileInputStream("f.txt"));
    ArrayList<Person> list = (ArrayList<Person>)ois.readObject();   //泛型在运行期会被擦除,索引运行期相当于没有泛型
                                                                    //想去掉黄色可以加注解                    @SuppressWarnings("unche
    for (Person person : list) {
        System.out.println(person);
    }

ois.close();
```

22.08_IO流(加上id号)(了解)

- 注意
  - 要写出的对象必须实现Serializable接口才能被序列化
  - 不用必须加id号

22.09_IO流(打印流的概述和特点)(掌握)

- 1.什么是打印流
  - 该流可以很方便的将对象的toString()结果输出, 并且自动加上换行, 而且可以使用自动刷出的模式
  - System.out就是一个PrintStream, 其默认向控制台输出信息

```
PrintStream ps = System.out;
ps.println(97);                 //其实底层用的是Integer.toString(x),将x转换为数字字符串打印
ps.println("xxx");
ps.println(new Person("张三", 23));
Person p = null;
ps.println(p);                  //如果是null,就返回null,如果不是null,就调用对象的toString()
```

- 2.使用方式
  - 打印: print(), println()
  - 自动刷出: PrintWriter(OutputStream out, boolean autoFlush, String encoding)
  - 打印流只操作数据目的

```
PrintWriter pw = new PrintWriter(new FileOutputStream("g.txt"), true);
pw.write(97);
pw.print("大家好");
pw.println("你好");               //自动刷出,只针对的是println方法
pw.close();
```

22.10_IO流(标准输入输出流概述和输出语句)

- 1.什么是标准输入输出流(掌握)
  - System.in是InputStream, 标准输入流, 默认可以从键盘输入读取字节数据
  - System.out是PrintStream, 标准输出流, 默认可以向Console中输出字符和字节数据
- 2.修改标准输入输出流(了解)
  - 修改输入流: System.setIn(InputStream)
  - 修改输出流: System.setOut(PrintStream)

```
System.setIn(new FileInputStream("a.txt"));             //修改标准输入流
System.setOut(new PrintStream("b.txt"));                //修改标准输出流

InputStream in = System.in;                             //获取标准输入流
PrintStream ps = System.out;                            //获取标准输出流
int b;
while((b = in.read()) != -1) {                          //从a.txt上读取数据
    ps.write(b);                                        //将数据写到b.txt上
}

in.close();
ps.close();
```

22.11_IO流(修改标准输入输出流拷贝图片)(了解)

```
    System.setIn(new FileInputStream("IO图片.png"));      //改变标准输入流
    System.setOut(new PrintStream("copy.png"));         //改变标准输出流

    InputStream is = System.in;                         //获取标准输入流
    PrintStream ps = System.out;                        //获取标准输出流

    int len;
```

```
byte[] arr = new byte[1024 * 8];

while((len = is.read(arr)) != -1) {
    ps.write(arr, 0, len);
}

is.close();
ps.close();
```

22.11_IO流(两种方式实现键盘录入)(了解)

- A:BufferedReader的readLine方法。
  - BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
- B:Scanner

22.12_IO流(随机访问流概述和读写数据)(了解)

- A:随机访问流概述
  - RandomAccessFile概述
  - RandomAccessFile类不属于流，是Object类的子类。但它融合了InputStream和OutputStream的功能。
  - 支持对随机访问文件的读取和写入。
- B:read(),write(),seek()

22.13_IO流(数据输入输出流)(了解)

- 1.什么是数据输入输出流
  - DataInputStream, DataOutputStream可以按照基本数据类型大小读写数据
  - 例如按Long大小写出一个数字, 写出时该数据占8字节. 读取的时候也可以按照Long类型读取, 一次读取8个字节.
- 2.使用方式
  - DataOutputStream(OutputStream), writeInt(), writeLong()

```
DataOutputStream dos = new DataOutputStream(new FileOutputStream("b.txt"));
dos.writeInt(997);
dos.writeInt(998);
dos.writeInt(999);

dos.close();
```

  - DataInputStream(InputStream), readInt(), readLong()

```
DataInputStream dis = new DataInputStream(new FileInputStream("b.txt"));
int x = dis.readInt();
int y = dis.readInt();
int z = dis.readInt();
System.out.println(x);
System.out.println(y);
System.out.println(z);
dis.close();
```

22.14_IO流(Properties的概述和作为Map集合的使用)(了解)

- A:Properties的概述
  - Properties 类表示了一个持久的属性集。
  - Properties 可保存在流中或从流中加载。
  - 属性列表中每个键及其对应值都是一个字符串。
- B:案例演示
  - Properties作为Map集合的使用

22.15_IO流(Properties的特殊功能使用)(了解)

- A:Properties的特殊功能
  - public Object setProperty(String key,String value)
  - public String getProperty(String key)
  - public Enumeration stringPropertyNames()
- B:案例演示
  - Properties的特殊功能

22.16_IO流(Properties的load()和store()功能)(了解)

- A:Properties的load()和store()功能
- B:案例演示

- Properties的load()和store()功能

22.17_day22总结

- 把今天的知识点总结一遍。

23.01_File类递归练习(统计该文件夹大小)

- 需求:1,从键盘接收一个文件夹路径,统计该文件夹大小

23.02_File类递归练习(删除该文件夹)

- 需求:2,从键盘接收一个文件夹路径,删除该文件夹

23.03_File类递归练习(拷贝)

- 需求:3,从键盘接收两个文件夹路径,把其中一个文件夹中(包含内容)拷贝到另一个文件夹中

23.04_File类递归练习(按层级打印)

- 需求:4,从键盘接收一个文件夹路径,把文件夹中的所有文件以及文件夹的名字按层级打印, 例如: aaa是文件夹,里面有bbb.txt,ccc.txt,ddd.txt这些文件,有eee这样的文件夹,eee中有fff.txt和ggg.txt,打印出层级来 aaa bbb.txt ccc.txt ddd.txt

```
eee
    fff.txt
    ggg.txt
```

23.05_递归练习(斐波那契数列)

- 不死神兔
- 故事得从西元1202年说起，话说有一位意大利青年，名叫斐波那契。
- 在他的一部著作中提出了一个有趣的问题：假设一对刚出生的小兔一个月后就能长成大兔，再过一个月就能生下一对小兔，并且此后每个月都生一对小兔，一年内没有发生死亡，
- 问：一对刚出生的兔子，一年内繁殖成多少对兔子?
- 1 1 2 3 5 8 13
- 第一个月一对小兔子 1
- 第二个月一对大兔子 1
- 第三个月一对大兔子生了一对小兔子 2
- 第四个月一对大兔子生了一对小兔子
- 一对小兔子长成大兔子 3
- 第五个月两对大兔子生两对小兔子
- 一对小兔子长成大兔子 5

23.06_递归练习(1000的阶乘所有零和尾部零的个数)

- 需求:求出1000的阶乘所有零和尾部零的个数,不用递归做

23.07_递归练习(1000的阶乘尾部零的个数)

- 需求:求出1000的阶乘尾部零的个数,用递归做

23.08_集合练习(约瑟夫环)

- 幸运数字

24.01_多线程(多线程的引入)(了解)

- 1.什么是线程
  - 线程是程序执行的一条路径, 一个进程中可以包含多条线程
  - 多线程并发执行可以提高程序的效率, 可以同时完成多项工作
- 2.多线程的应用场景
  - 红蜘蛛同时共享屏幕给多个电脑
  - 迅雷开启多条线程一起下载
  - QQ同时和多个人一起视频
  - 服务器同时处理多个客户端请求

24.02_多线程(多线程并行和并发的区别)(了解)

- 并行就是两个任务同时运行，就是甲任务进行的同时，乙任务也在进行。(需要多核CPU)
- 并发是指两个任务都请求运行，而处理器只能按受一个任务，就把这两个任务安排轮流进行，由于时间间隔较短，使人感觉两个任务都在运行。
- 比如我跟两个网友聊天，左手操作一个电脑跟甲聊，同时右手用另一台电脑跟乙聊天，这就叫并行。
- 如果用一台电脑我先给甲发个消息，然后立刻再给乙发消息，然后再跟甲聊，再跟乙聊。这就叫并发。

24.03_多线程(Java程序运行原理和JVM的启动是多线程的吗)(了解)

- A:Java程序运行原理
  - Java命令会启动java虚拟机，启动JVM，等于启动了一个应用程序，也就是启动了一个进程。该进程会自动启动一个 “主线程” ，然后主线程去调用某个类的 main 方法。
- B:JVM的启动是多线程的吗
  - JVM启动至少启动了垃圾回收线程和主线程，所以是多线程的。

24.04_多线程(多线程程序实现的方式1)(掌握)

- 1.继承Thread
  - 定义类继承Thread
  - 重写run方法
  - 把新线程要做的事写在run方法中
  - 创建线程对象
  - 开启新线程, 内部会自动执行run方法

```
public class Demo2_Thread {

    /**
     * @param args
     */
    public static void main(String[] args) {
        MyThread mt = new MyThread();                           //4,创建自定义类的对象
        mt.start();                                             //5,开启线程

        for(int i = 0; i < 3000; i++) {
            System.out.println("bb");
        }
    }

}
class MyThread extends Thread {                                 //1,定义类继承Thread
    public void run() {                                         //2,重写run方法
        for(int i = 0; i < 3000; i++) {                         //3,将要执行的代码,写在run方法中
            System.out.println("aaaaaaaaaaaaaaaaaaaaaaaaaaaa");
        }
    }
}
```

24.05_多线程(多线程程序实现的方式2)(掌握)

- 2.实现Runnable
  - 定义类实现Runnable接口
  - 实现run方法
  - 把新线程要做的事写在run方法中
  - 创建自定义的Runnable的子类对象
  - 创建Thread对象, 传入Runnable
  - 调用start()开启新线程, 内部会自动调用Runnable的run()方法

```
public class Demo3_Runnable {
    /**
```

```
     * @param args
     */
    public static void main(String[] args) {
        MyRunnable mr = new MyRunnable();                       //4,创建自定义类对象
        //Runnable target = new MyRunnable();
        Thread t = new Thread(mr);                              //5,将其当作参数传递给Thread的构造函数
        t.start();                                              //6,开启线程

        for(int i = 0; i < 3000; i++) {
            System.out.println("bb");
        }
    }
}

class MyRunnable implements Runnable {                          //1,自定义类实现Runnable接口
    @Override
    public void run() {                                         //2,重写run方法
        for(int i = 0; i < 3000; i++) {                         //3,将要执行的代码,写在run方法中
            System.out.println("aaaaaaaaaaaaaaaaaaaaaaaaaaaa");
        }
    }

}
```

24.06_多线程(实现Runnable的原理)(了解)

- 查看源码
  - 1,看Thread类的构造函数,传递了Runnable接口的引用
  - 2,通过init()方法找到传递的target给成员变量的target赋值
  - 3,查看run方法,发现run方法中有判断,如果target不为null就会调用Runnable接口子类对象的run方法

24.07_多线程(两种方式的区别)(掌握)

- 查看源码的区别:
  - a.继承Thread : 由于子类重写了Thread类的run(), 当调用start()时, 直接找子类的run()方法
  - b.实现Runnable : 构造函数中传入了Runnable的引用, 成员变量记住了它, start()调用run()方法时内部判断成员变量Runnable的引用是否为空, 不为空编译时看的是Runnable的run(),运行时执行的是子类的run()方法
- 继承Thread
  - 好处是:可以直接使用Thread类中的方法,代码简单
  - 弊端是:如果已经有了父类,就不能用这种方法
- 实现Runnable接口
  - 好处是:即使自己定义的线程类有了父类也没关系,因为有了父类也可以实现接口,而且接口是可以多实现的
  - 弊端是:不能直接使用Thread中的方法需要先获取到线程对象后,才能得到Thread的方法,代码复杂

24.08_多线程(匿名内部类实现线程的两种方式)(掌握)

- 继承Thread类

```
new Thread() {                                                  //1,new 类(){}继承这个类
    public void run() {                                         //2,重写run方法
        for(int i = 0; i < 3000; i++) {                         //3,将要执行的代码,写在run方法中
            System.out.println("aaaaaaaaaaaaaaaaaaaaaaaaaaaa");
        }
    }
}.start();
```

- 实现Runnable接口

```
new Thread(new Runnable(){                                      //1,new 接口(){}实现这个接口
    public void run() {                                         //2,重写run方法
        for(int i = 0; i < 3000; i++) {                         //3,将要执行的代码,写在run方法中
            System.out.println("bb");
        }
    }
}).start();
```

24.09_多线程(获取名字和设置名字)(掌握)

- 1.获取名字
  - 通过getName()方法获取线程对象的名字
- 2.设置名字
  - 通过构造函数可以传入String类型的名字

```
new Thread("xxx") {
    public void run() {
        for(int i = 0; i < 1000; i++) {
            System.out.println(this.getName() + "....aaaaaaaaaaaaaaaaaaaaa");
        }
    }
}.start();

new Thread("yyy") {
    public void run() {
        for(int i = 0; i < 1000; i++) {
            System.out.println(this.getName() + "....bb");
        }
    }
}.start();
```

- 通过setName(String)方法可以设置线程对象的名字

```
Thread t1 = new Thread() {
    public void run() {
        for(int i = 0; i < 1000; i++) {
            System.out.println(this.getName() + "....aaaaaaaaaaaaaaaaaaaaa");
        }
    }
};

Thread t2 = new Thread() {
    public void run() {
        for(int i = 0; i < 1000; i++) {
            System.out.println(this.getName() + "....bb");
        }
    }
};
t1.setName("芙蓉姐姐");
t2.setName("凤姐");

t1.start();
t2.start();
```

24.10_多线程(获取当前线程的对象)(掌握)

- Thread.currentThread(), 主线程也可以获取

```
new Thread(new Runnable() {
    public void run() {
        for(int i = 0; i < 1000; i++) {
            System.out.println(Thread.currentThread().getName() + "...aaaaaaaaaaaaaaaaaaaa");
        }
    }
}).start();

new Thread(new Runnable() {
    public void run() {
        for(int i = 0; i < 1000; i++) {
            System.out.println(Thread.currentThread().getName() + "...bb");
        }
    }
}).start();
Thread.currentThread().setName("我是主线程");                    //获取主函数线程的引用,并改名字
System.out.println(Thread.currentThread().getName());       //获取主函数线程的引用,并获取名字
```

24.11_多线程(休眠线程)(掌握)

- Thread.sleep(毫秒,纳秒), 控制当前线程休眠若干毫秒1秒= 1000毫秒 1秒 = 1000 * 1000 * 1000纳秒 1000000000

```
new Thread() {
    public void run() {
        for(int i = 0; i < 10; i++) {
            System.out.println(getName() + "...aaaaaaaaaaaaaaaaaaaaa");
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}.start();
```

```
new Thread() {
    public void run() {
        for(int i = 0; i < 10; i++) {
            System.out.println(getName() + "...bb");
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}.start();
```

24.12_多线程(守护线程)(掌握)

- setDaemon(), 设置一个线程为守护线程, 该线程不会单独执行, 当其他非守护线程都执行结束后, 自动退出

```
Thread t1 = new Thread() {
    public void run() {
        for(int i = 0; i < 50; i++) {
            System.out.println(getName() + "...aaaaaaaaaaaaaaaaaaaaa");
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
};

Thread t2 = new Thread() {
    public void run() {
        for(int i = 0; i < 5; i++) {
            System.out.println(getName() + "...bb");
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
};

t1.setDaemon(true);                    //将t1设置为守护线程

t1.start();
t2.start();
```

24.13_多线程(加入线程)(掌握)

- join(), 当前线程暂停, 等待指定的线程执行结束后, 当前线程再继续
- join(int), 可以等待指定的毫秒之后继续

```
final Thread t1 = new Thread() {
    public void run() {
        for(int i = 0; i < 50; i++) {
            System.out.println(getName() + "...aaaaaaaaaaaaaaaaaaaaa");
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
};

Thread t2 = new Thread() {
    public void run() {
        for(int i = 0; i < 50; i++) {
            if(i == 2) {
                try {
                    //t1.join();                        //插队,加入
                    t1.join(30);                        //加入,有固定的时间,过了固定时间,继续交替执行
                    Thread.sleep(10);
                } catch (InterruptedException e) {

                    e.printStackTrace();
                }
```

```
            }
            System.out.println(getName() + "...bb");

        }
    }
};

t1.start();
t2.start();
```

24.14_多线程(礼让线程)(了解)

- yield让出cpu

24.15_多线程(设置线程的优先级)(了解)

- setPriority()设置线程的优先级

24.16_多线程(同步代码块)(掌握)

- 1.什么情况下需要同步
  - 当多线程并发, 有多段代码同时执行时, 我们希望某一段代码执行的过程中CPU不要切换到其他线程工作. 这时就需要同步.
  - 如果两段代码是同步的, 那么同一时间只能执行一段, 在一段代码没执行结束之前, 不会执行另外一段代码.
- 2.同步代码块
  - 使用synchronized关键字加上一个锁对象来定义一段代码, 这就叫同步代码块
  - 多个同步代码块如果使用相同的锁对象, 那么他们就是同步的

```
class Printer {
    Demo d = new Demo();
    public static void print1() {
        synchronized(d){                //锁对象可以是任意对象,但是被锁的代码需要保证是同一把锁,不能用匿名对象
            System.out.print("黑");
            System.out.print("马");
            System.out.print("程");
            System.out.print("序");
            System.out.print("员");
            System.out.print("\r\n");
        }
    }

    public static void print2() {
        synchronized(d){
            System.out.print("传");
            System.out.print("智");
            System.out.print("播");
            System.out.print("客");
            System.out.print("\r\n");
        }
    }
}
```

24.17_多线程(同步方法)(掌握)

- 使用synchronized关键字修饰一个方法, 该方法中所有的代码都是同步的

```
class Printer {
    public static void print1() {
        synchronized(Printer.class){                //锁对象可以是任意对象,但是被锁的代码需要保证是同一把锁,不能用匿名对象
            System.out.print("黑");
            System.out.print("马");
            System.out.print("程");
            System.out.print("序");
            System.out.print("员");
            System.out.print("\r\n");
        }
    }
    /*
     * 非静态同步函数的锁是:this
     * 静态的同步函数的锁是:字节码对象
     */
    public static synchronized void print2() {
        System.out.print("传");
        System.out.print("智");
        System.out.print("播");
        System.out.print("客");
        System.out.print("\r\n");
```

什么时候会出现线程安全问题？　　在单线程中不会出现线程安全问题，而在多线程编程中，有可能会出现同时访问同一个资源的情况，这种资源可以是各种类型的的资源：一个变量、一个对象、一个文件、一个数据库表等，而当多个线程同时访问同一个资源的时候，就会存在一个问题：

如何解决线程安全问题？

基本上所有的并发模式在解决线程安全问题上，都采用“序列化访问临界资源”的方案，即在同一时刻，只能有一个线程访问临界资源，也称同步互斥访问。

　　通常来说，是在访问临界资源的代码前面加上一个锁，当访问完临界资源后释放锁，让其他线程继续访问。

```
    }
}
```

24.18_多线程(线程安全问题)(掌握)

- 多线程并发操作同一数据时, 就有可能出现线程安全问题
- 使用同步技术可以解决这种问题, 把操作数据的代码进行同步, 不要多个线程一起操作

```
public class Demo2_Synchronized {

    /**
     * @param args
     * 需求:铁路售票,一共100张,通过四个窗口卖完.
     */
    public static void main(String[] args) {
        TicketsSeller t1 = new TicketsSeller();
        TicketsSeller t2 = new TicketsSeller();
        TicketsSeller t3 = new TicketsSeller();
        TicketsSeller t4 = new TicketsSeller();

        t1.setName("窗口1");
        t2.setName("窗口2");
        t3.setName("窗口3");
        t4.setName("窗口4");
        t1.start();
        t2.start();
        t3.start();
        t4.start();
    }

}

class TicketsSeller extends Thread {
    private static int tickets = 100;
    static Object obj = new Object();
    public TicketsSeller() {
        super();

    }
    public TicketsSeller(String name) {
        super(name);
    }
    public void run() {
        while(true) {
            synchronized(obj) {
                if(tickets <= 0)
                    break;
                try {
                    Thread.sleep(10);//线程1睡,线程2睡,线程3睡,线程4睡
                } catch (InterruptedException e) {

                    e.printStackTrace();
                }
                System.out.println(getName() + "...这是第" + tickets-- + "号票");
            }
        }
    }
}
```

synchronized同步方法或者同步块

在了解synchronized关键字的使用方法之前，我们先来看一个概念：互斥锁，顾名思义：能到达到互斥访问目的的锁。

举个简单的例子：如果对临界资源加上互斥锁，当一个线程在访问该临界资源时，其他线程便只能等待。

在Java中，每一个对象都拥有一个锁标记（monitor），也称为监视器，多线程同时访问某个对象时，线程只有获取了该对象的锁才能访问。

在Java中，可以使用synchronized关键字来标记一个方法或者代码块，当某个线程调用该对象的synchronized方法或者访问synchronized代码块时，这个线程便获得了该对象的锁，其他线程暂时无法访问这个方法，只有等待这个方法执行完毕或者代码块执行完毕，这个线程才会释放该对象的锁，其他线程才能执行这个方法或者代码块。

24.19_多线程(火车站卖票的例子用实现Runnable接口)(掌握)

24.20_多线程(死锁)(了解)

- 多线程同步的时候, 如果同步代码嵌套, 使用相同锁, 就有可能出现死锁
    - 尽量不要嵌套使用

```
private static String s1 = "筷子左";
private static String s2 = "筷子右";
public static void main(String[] args) {
    new Thread() {
        public void run() {
            while(true) {
                synchronized(s1) {
                    System.out.println(getName() + "...拿到" + s1 + "等待" + s2);
                    synchronized(s2) {
                        System.out.println(getName() + "...拿到" + s2 + "开吃");
                    }
```

```
                }
            }
        }
    }.start();

    new Thread() {
        public void run() {
            while(true) {
                synchronized(s2) {
                    System.out.println(getName() + "...拿到" + s2 + "等待" + s1);
                    synchronized(s1) {
                        System.out.println(getName() + "...拿到" + s1 + "开吃");
                    }
                }
            }
        }
    }.start();
}
```

24.21_多线程(以前的线程安全的类回顾)(掌握)

- A:回顾以前说过的线程安全问题
  - 看源码：Vector,StringBuffer,Hashtable,Collections.synchroinzed(xxx)
  - Vector是线程安全的,ArrayList是线程不安全的
  - StringBuffer是线程安全的,StringBuilder是线程不安全的
  - Hashtable是线程安全的,HashMap是线程不安全的

24.22_多线程(总结)

单例模式，是一种常用的软件设计模式。在它的核心结构中只包含一个被称为单例的特殊类。通过单例模式可以保证系统中一个类只有一个实例。即一个类只有一个对象实例

25.01_多线程(单例设计模式)(掌握)

- 单例设计模式：保证类在内存中只有一个对象。
- 如何保证类在内存中只有一个对象呢？
  - (1)控制类的创建，不让其他类来创建本类的对象。private
  - (2)在本类中定义一个本类的对象。Singleton s;
  - (3)提供公共的访问方式。 public static Singleton getInstance(){return s}

显然单例模式的要点有三个;一是某个类只能有一个实例;二是它必须自行创建这个实例;三是它必须自行向整个系统提供这个实例。

从具体实现角度来说，就是以下三点:一是单例模式的类只提供私有的构造函数，二是类定义中含有一个该类的静态私有对象，三是该类提供了一个静态的公有的函数用于创建或获取它本身的静态私有对象。

- 单例写法两种：
  - (1)饿汉式 开发用这种方式。

```
//饿汉式
class Singleton {
    //1,私有构造函数
    private Singleton(){}
    //2,创建本类对象
    private static Singleton s = new Singleton();
    //3,对外提供公共的访问方法
    public static Singleton getInstance() {
        return s;
    }

    public static void print() {
        System.out.println("11111111111");
    }
}
```

  - (2)懒汉式 面试写这种方式。多线程的问题？

```
//懒汉式,单例的延迟加载模式
class Singleton {
    //1,私有构造函数
    private Singleton(){}
    //2,声明一个本类的引用
    private static Singleton s;
    //3,对外提供公共的访问方法
    public static Singleton getInstance() {
        if(s == null)
            //线程1,线程2
            s = new Singleton();
        return s;
    }

    public static void print() {
        System.out.println("11111111111");
    }
}
```

  - (3)第三种格式

```
class Singleton {
    private Singleton() {}

    public static final Singleton s = new Singleton();//final是最终的意思,被final修饰的变量不可以被更改
}
```

25.02_多线程(Runtime类)

- Runtime类是一个单例类

```
Runtime r = Runtime.getRuntime();
//r.exec("shutdown -s -t 300");     //300秒后关机
r.exec("shutdown -a");              //取消关机
```

25.03_多线程(Timer)(掌握)

- Timer类:计时器

```
public class Demo5_Timer {
    /**
     * @param args
     * 计时器
     * @throws InterruptedException
     */
    public static void main(String[] args) throws InterruptedException {
        Timer t = new Timer();
```

```
            t.schedule(new MyTimerTask(), new Date(114,9,15,10,54,20),3000);

            while(true) {
                System.out.println(new Date());
                Thread.sleep(1000);
            }
        }
    }
    class MyTimerTask extends TimerTask {
        @Override
        public void run() {
            System.out.println("起床背英语单词");
        }

    }
```

25.04_多线程(两个线程间的通信)(掌握)

- 1.什么时候需要通信
  - 多个线程并发执行时, 在默认情况下CPU是随机切换线程的
  - 如果我们希望他们有规律的执行, 就可以使用通信, 例如每个线程执行一次打印
- 2.怎么通信
  - 如果希望线程等待, 就调用wait()
  - 如果希望唤醒等待的线程, 就调用notify();
  - 这两个方法必须在同步代码中执行, 并且使用同步锁对象来调用

25.05_多线程(三个或三个以上间的线程通信)

- 多个线程通信的问题
  - notify()方法是随机唤醒一个线程
  - notifyAll()方法是唤醒所有线程
  - JDK5之前无法唤醒指定的一个线程
  - 如果多个线程之间通信, 需要使用notifyAll()通知所有线程, 用while来反复判断条件

25.06_多线程(JDK1.5的新特性互斥锁)(掌握)

- 1.同步
  - 使用ReentrantLock类的lock()和unlock()方法进行同步
- 2.通信
  - 使用ReentrantLock类的newCondition()方法可以获取Condition对象
  - 需要等待的时候使用Condition的await()方法, 唤醒的时候用signal()方法
  - 不同的线程使用不同的Condition, 这样就能区分唤醒的时候找哪个线程了

25.07_多线程(线程组的概述和使用)(了解)

- A:线程组概述
  - Java中使用ThreadGroup来表示线程组，它可以对一批线程进行分类管理，Java允许程序直接对线程组进行控制。
  - 默认情况下，所有的线程都属于主线程组。
    - public final ThreadGroup getThreadGroup()//通过线程对象获取他所属于的组
    - public final String getName()//通过线程组对象获取他组的名字
  - 我们也可以给线程设置分组
    - 1,ThreadGroup(String name) 创建线程组对象并给其赋值名字
    - 2,创建线程对象
    - 3,Thread(ThreadGroup?group, Runnable?target, String?name)
    - 4,设置整组的优先级或者守护线程
  - B:案例演示
    - 线程组的使用,默认是主线程组

```
MyRunnable mr = new MyRunnable();
Thread t1 = new Thread(mr, "张三");
Thread t2 = new Thread(mr, "李四");
//获取线程组
// 线程类里面的方法：public final ThreadGroup getThreadGroup()
ThreadGroup tg1 = t1.getThreadGroup();
ThreadGroup tg2 = t2.getThreadGroup();
// 线程组里面的方法：public final String getName()
String name1 = tg1.getName();
String name2 = tg2.getName();
System.out.println(name1);
System.out.println(name2);
// 通过结果我们知道了：线程默认情况下属于main线程组
// 通过下面的测试，你应该能够看到，默任情况下，所有的线程都属于同一个组
System.out.println(Thread.currentThread().getThreadGroup().getName());
```

- 自己设定线程组

```
// ThreadGroup(String name)
ThreadGroup tg = new ThreadGroup("这是一个新的组");

MyRunnable mr = new MyRunnable();
// Thread(ThreadGroup group, Runnable target, String name)
Thread t1 = new Thread(tg, mr, "张三");
Thread t2 = new Thread(tg, mr, "李四");

System.out.println(t1.getThreadGroup().getName());
System.out.println(t2.getThreadGroup().getName());

//通过组名称设置后台线程，表示该组的线程都是后台线程
tg.setDaemon(true);
```

25.08_多线程(线程的五种状态)(掌握)

- 看图说话
- 新建,就绪,运行,阻塞,死亡

25.09_多线程(线程池的概述和使用)(了解)

- A:线程池概述
  - 程序启动一个新线程成本是比较高的，因为它涉及到要与操作系统进行交互。而使用线程池可以很好的提高性能，尤其是当程序中要创建大量生存期很短的线程时，更应该考虑使用线程池。线程池里的每一个线程代码结束后，并不会死亡，而是再次回到线程池中成为空闲状态，等待下一个对象来使用。在JDK5之前，我们必须手动实现自己的线程池，从JDK5开始，Java内置支持线程池
- B:内置线程池的使用概述
  - JDK5新增了一个Executors工厂类来产生线程池，有如下几个方法
    - public static ExecutorService newFixedThreadPool(int nThreads)
    - public static ExecutorService newSingleThreadExecutor()
    - 这些方法的返回值是ExecutorService对象，该对象表示一个线程池，可以执行Runnable对象或者Callable对象代表的线程。它提供了如下方法
    - Future<?> submit(Runnable task)
    - Future submit(Callable task)
  - 使用步骤：
    - 创建线程池对象
    - 创建Runnable实例
    - 提交Runnable实例
    - 关闭线程池
  - C:案例演示
    - 提交的是Runnable

```
// public static ExecutorService newFixedThreadPool(int nThreads)
ExecutorService pool = Executors.newFixedThreadPool(2);

// 可以执行Runnable对象或者Callable对象代表的线程
pool.submit(new MyRunnable());
pool.submit(new MyRunnable());

//结束线程池
pool.shutdown();
```

25.10_多线程(多线程程序实现的方式3)(了解)

- 提交的是Callable

```
// 创建线程池对象
ExecutorService pool = Executors.newFixedThreadPool(2);

// 可以执行Runnable对象或者Callable对象代表的线程
Future<Integer> f1 = pool.submit(new MyCallable(100));
Future<Integer> f2 = pool.submit(new MyCallable(200));

// V get()
Integer i1 = f1.get();
Integer i2 = f2.get();

System.out.println(i1);
System.out.println(i2);

// 结束
pool.shutdown();

public class MyCallable implements Callable<Integer> {
```

```
    private int number;

    public MyCallable(int number) {
        this.number = number;
    }

    @Override
    public Integer call() throws Exception {
        int sum = 0;
        for (int x = 1; x <= number; x++) {
            sum += x;
        }
        return sum;
    }

}
```

- 多线程程序实现的方式3的好处和弊端
  - 好处：
    - 可以有返回值
    - 可以抛出异常
  - 弊端：
    - 代码比较复杂，所以一般不用

25.11_设计模式(简单工厂模式概述和使用)(了解)

- A:简单工厂模式概述
  - 又叫静态工厂方法模式，它定义一个具体的工厂类负责创建一些类的实例
- B:优点
  - 客户端不需要在负责对象的创建，从而明确了各个类的职责
- C:缺点
  - 这个静态工厂类负责所有对象的创建，如果有新的对象增加，或者某些对象的创建方式不同，就需要不断的修改工厂类，不利于后期的维护
- D:案例演示
  - 动物抽象类：public abstract Animal { public abstract void eat(); }
  - 具体狗类：public class Dog extends Animal {}
  - 具体猫类：public class Cat extends Animal {}
  - 开始，在测试类中每个具体的内容自己创建对象，但是，创建对象的工作如果比较麻烦，就需要有人专门做这个事情，所以就知道了一个专门的类来创建对象。

```
public class AnimalFactory {
    private AnimalFactory(){}

    //public static Dog createDog() {return new Dog();}
    //public static Cat createCat() {return new Cat();}

    //改进
    public static Animal createAnimal(String animalName) {
        if(“dog”.equals(animalName)) {}
        else if(“cat”.equals(animale)) {

        }else {
            return null;
        }
    }
}
```

25.12_设计模式(工厂方法模式的概述和使用)(了解)

- A:工厂方法模式概述
  - 工厂方法模式中抽象工厂类负责定义创建对象的接口，具体对象的创建工作由继承抽象工厂的具体类实现。
- B:优点
  - 客户端不需要在负责对象的创建，从而明确了各个类的职责，如果有新的对象增加，只需要增加一个具体的类和具体的工厂类即可，不影响已有的代码，后期维护容易，增强了系统的扩展性
- C:缺点
  - 需要额外的编写代码，增加了工作量
- D:案例演示

```
动物抽象类：public abstract Animal { public abstract void eat(); }
工厂接口：public interface Factory {public abstract Animal createAnimal();}
具体狗类：public class Dog extends Animal {}
具体猫类：public class Cat extends Animal {}
开始，在测试类中每个具体的内容自己创建对象，但是，创建对象的工作如果比较麻烦，就需要有人专门做这个事情，所以就知道了一个专门的类来创建对象。发现每次修改
狗工厂：public class DogFactory implements Factory {
    public Animal createAnimal() {…}
```

```
        }
猫工厂：public class CatFactory implements Factory {
    public Animal createAnimal() {…}
        }
```

25.13_GUI(如何创建一个窗口并显示)

- Graphical User Interface(图形用户接口)。

```
Frame  f = new Frame(“my window”);
f.setLayout(new FlowLayout());//设置布局管理器
f.setSize(500,400);//设置窗体大小
f.setLocation(300,200);//设置窗体出现在屏幕的位置
f.setIconImage(Toolkit.getDefaultToolkit().createImage("qq.png"));
f.setVisible(true);
```

25.14_GUI(布局管理器)

- FlowLayout（流式布局管理器）
  - 从左到右的顺序排列。
  - Panel默认的布局管理器。
- BorderLayout（边界布局管理器）
  - 东，南，西，北，中
  - Frame默认的布局管理器。
- GridLayout（网格布局管理器）
  - 规则的矩阵
- CardLayout（卡片布局管理器）
  - 选项卡
- GridBagLayout（网格包布局管理器）
  - 非规则的矩阵

25.15_GUI(窗体监听)

```
Frame f = new Frame("我的窗体");
//事件源是窗体,把监听器注册到事件源上
//事件对象传递给监听器
f.addWindowListener(new WindowAdapter() {
          public void windowClosing(WindowEvent e) {
                    //退出虚拟机,关闭窗口
       System.exit(0);
    }
});
```

25.16_GUI(鼠标监听)

25.17_GUI(键盘监听和键盘事件)

25.18_GUI(动作监听)

25.19_设计模式(适配器设计模式)(掌握)

- a.什么是适配器
  - 在使用监听器的时候, 需要定义一个类事件监听器接口.
  - 通常接口中有多个方法, 而程序中不一定所有的都用到, 但又必须重写, 这很繁琐.
  - 适配器简化了这些操作, 我们定义监听器时只要继承适配器, 然后重写需要的方法即可.
- b.适配器原理
  - 适配器就是一个类, 实现了监听器接口, 所有抽象方法都重写了, 但是方法全是空的.
  - 适配器类需要定义成抽象的,因为创建该类对象,调用空方法是没有意义的
  - 目的就是为了简化程序员的操作, 定义监听器时继承适配器, 只重写需要的方法就可以了.

25.20_GUI(需要知道的)

- 事件处理
  - 事件: 用户的一个操作
  - 事件源: 被操作的组件
  - 监听器: 一个自定义类的对象, 实现了监听器接口, 包含事件处理方法,把监听器添加在事件源上, 当事件发生的时候虚拟机就会自动调用监听器中的事件处理方法

25.21_day25总结

把今天的知识点总结一遍。

day26授课目录：

26.01_网络编程(网络编程概述)(了解)

- A:计算机网络
  - 是指将地理位置不同的具有独立功能的多台计算机及其外部设备，通过通信线路连接起来，在网络操作系统，网络管理软件及网络通信协议的管理和协调下，实现资源共享和信息传递的计算机系统。
- B:网络编程
  - 就是用来实现网络互连的不同计算机上运行的程序间可以进行数据交换。

26.02_网络编程(网络编程三要素之IP概述)(掌握)

- 每个设备在网络中的唯一标识
- 每台网络终端在网络中都有一个独立的地址，我们在网络中传输数据就是使用这个地址。
- ipconfig：查看本机IP192.168.12.42
- ping：测试连接192.168.40.62
- 本地回路地址：127.0.0.1 255.255.255.255是广播地址
- IPv4：4个字节组成，4个0-255。大概42亿，30亿都在北美，亚洲4亿。2011年初已经用尽。
- IPv6：8组，每组4个16进制数。
- 1a2b:0000:aaaa:0000:0000:0000:aabb:1f2f
- 1a2b::aaaa:0000:0000:0000:aabb:1f2f
- 1a2b:0000:aaaa::aabb:1f2f
- 1a2b:0000:aaaa::0000:aabb:1f2f
- 1a2b:0000:aaaa:0000::aabb:1f2f

26.03_网络编程(网络编程三要素之端口号概述)(掌握)

- 每个程序在设备上的唯一标识
- 每个网络程序都需要绑定一个端口号，传输数据的时候除了确定发到哪台机器上，还要明确发到哪个程序。
- 端口号范围从0-65535
- 编写网络应用就需要绑定一个端口号，尽量使用1024以上的，1024以下的基本上都被系统程序占用了。
- 常用端口
  - mysql: 3306
  - oracle: 1521
  - web: 80
  - tomcat: 8080
  - QQ: 4000
  - feiQ: 2425

网络上的两个程序通过一个双向的通信连接实现数据的交换，这个连接的一端称为一个socket。

建立网络通信连接至少要一对端口号(socket)。socket本质是编程接口(API)，对TCP/IP的封装，TCP/IP也要提供可供程序员做网络开发所用的接口，这就是Socket编程接口;HTTP是轿车，提供了封装或者显示数据的具体形式;Socket是发动机，提供了网络通信的能力。

26.04_网络编程(网络编程三要素协议)(掌握)

- 为计算机网络中进行数据交换而建立的规则、标准或约定的集合。
- UDP
  - 面向无连接，数据不安全，速度快。不区分客户端与服务端。
- TCP * 面向连接（三次握手），数据安全，速度略低。分为客户端和服务端。
  - 三次握手: 客户端先向服务端发起请求, 服务端响应请求, 传输数据

26.05_网络编程(Socket通信原理图解)(了解)

- A:Socket套接字概述：
  - 网络上具有唯一标识的IP地址和端口号组合在一起才能构成唯一能识别的标识符套接字。
  - 通信的两端都有Socket。
  - 网络通信其实就是Socket间的通信。
  - 数据在两个Socket间通过IO流传输。
  - Socket在应用程序中创建，通过一种绑定机制与驱动程序建立关系，告诉自己所对应的IP和port。

26.06_网络编程(UDP传输)(了解)

- 1.发送Send
  - 创建DatagramSocket, 随机端口号
  - 创建DatagramPacket, 指定数据, 长度, 地址, 端口
  - 使用DatagramSocket发送DatagramPacket
  - 关闭DatagramSocket
- 2.接收Receive
  - 创建DatagramSocket, 指定端口号
  - 创建DatagramPacket, 指定数组, 长度
  - 使用DatagramSocket接收DatagramPacket
  - 关闭DatagramSocket
  - 从DatagramPacket中获取数据

- 3.接收方获取ip和端口号
  - String ip = packet.getAddress().getHostAddress();
  - int port = packet.getPort();

26.07_网络编程(UDP传输优化)

- 接收端Receive

```
DatagramSocket socket = new DatagramSocket(6666);                       //创建socket相当于创建码头
DatagramPacket packet = new DatagramPacket(new byte[1024], 1024);       //创建packet相当于创建集装箱

while(true) {
    socket.receive(packet);                                             //接收货物
    byte[] arr = packet.getData();
    int len = packet.getLength();
    String ip = packet.getAddress().getHostAddress();
    System.out.println(ip + ":" + new String(arr,0,len));
}
```

- 发送端Send

```
DatagramSocket socket = new DatagramSocket();       //创建socket相当于创建码头
Scanner sc = new Scanner(System.in);

while(true) {
    String str = sc.nextLine();
    if("quit".equals(str))
        break;
    DatagramPacket packet =                             //创建packet相当于创建集装箱
            new DatagramPacket(str.getBytes(), str.getBytes().length, InetAddress.getByName("127.0.0.1"), 6666);
    socket.send(packet);            //发货
}
socket.close();
```

26.08_网络编程(UDP传输多线程)

- A发送和接收在一个窗口完成

```
public class Demo3_MoreThread {

    /**
     * @param args
     */
    public static void main(String[] args) {
        new Receive().start();

        new Send().start();
    }

}

class Receive extends Thread {
    public void run() {
        try {
            DatagramSocket socket = new DatagramSocket(6666);                   //创建socket相当于创建码头
            DatagramPacket packet = new DatagramPacket(new byte[1024], 1024);   //创建packet相当于创建集装箱

            while(true) {
                socket.receive(packet);                                             //接收货物
                byte[] arr = packet.getData();
                int len = packet.getLength();
                String ip = packet.getAddress().getHostAddress();
                System.out.println(ip + ":" + new String(arr,0,len));
            }
        } catch (IOException e) {

            e.printStackTrace();
        }
    }
}

class Send extends Thread {
    public void run() {
        try {
            DatagramSocket socket = new DatagramSocket();       //创建socket相当于创建码头
            Scanner sc = new Scanner(System.in);

            while(true) {
```

```
                String str = sc.nextLine();
                if("quit".equals(str))
                    break;
                DatagramPacket packet =                             //创建packet相当于创建集装箱
                        new DatagramPacket(str.getBytes(), str.getBytes().length, InetAddress.getByName("127.0.0.1"), 6666);
                socket.send(packet);            //发货
            }
            socket.close();
        }  catch (IOException e) {

            e.printStackTrace();
        }
    }
}
```

26.09_网络编程(UDP聊天图形化界面)

26.10_网络编程(UDP聊天发送功能)

26.11_网络编程(UDP聊天记录功能)

26.12_网络编程(UDP聊天清屏功能)

26.13_网络编程(UDP聊天震动功能)

26.14_网络编程(UDP聊天快捷键和代码优化)

26.15_网络编程(UDP聊天生成jar文件)

26.16_网络编程(TCP协议)(掌握)

- 1.客户端
  - 创建Socket连接服务端(指定ip地址,端口号)通过ip地址找对应的服务器
  - 调用Socket的getInputStream()和getOutputStream()方法获取和服务端相连的IO流
  - 输入流可以读取服务端输出流写出的数据
  - 输出流可以写出数据到服务端的输入流
- 2.服务端
  - 创建ServerSocket(需要指定端口号)
  - 调用ServerSocket的accept()方法接收一个客户端请求，得到一个Socket
  - 调用Socket的getInputStream()和getOutputStream()方法获取和客户端相连的IO流
  - 输入流可以读取客户端输出流写出的数据
  - 输出流可以写出数据到客户端的输入流

26.17_网络编程(TCP协议代码优化)

- 客户端

```
Socket socket = new Socket("127.0.0.1", 9999);      //创建Socket指定ip地址和端口号
InputStream is = socket.getInputStream();           //获取输入流
OutputStream os = socket.getOutputStream();         //获取输出流
BufferedReader br = new BufferedReader(new InputStreamReader(is));
PrintStream ps = new PrintStream(os);

System.out.println(br.readLine());
ps.println("我想报名就业班");
System.out.println(br.readLine());
ps.println("爷不学了");
socket.close();
```

- 服务端

```
ServerSocket server = new ServerSocket(9999);   //创建服务器
Socket socket = server.accept();                //接受客户端的请求
InputStream is = socket.getInputStream();       //获取输入流
OutputStream os = socket.getOutputStream();     //获取输出流

BufferedReader br = new BufferedReader(new InputStreamReader(is));
PrintStream ps = new PrintStream(os);

ps.println("欢迎咨询传智播客");
System.out.println(br.readLine());
ps.println("报满了,请报下一期吧");
System.out.println(br.readLine());
```

```
server.close();
socket.close();
```

26.18_网络编程(服务端是多线程的)(掌握)

```
ServerSocket server = new ServerSocket(9999);   //创建服务器
    while(true) {
        final Socket socket = server.accept();              //接受客户端的请求
        new Thread() {
            public void run() {
                try {
                    BufferedReader br = new BufferedReader(new InputStreamReader(socket.getInputStream()));
                    PrintStream ps = new PrintStream(socket.getOutputStream());
                    ps.println("欢迎咨询传智播客");
                    System.out.println(br.readLine());
                    ps.println("报满了,请报下一期吧");
                    System.out.println(br.readLine());
                    socket.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }.start();
    }
}
```

26.19_网络编程(练习)

- 客户端向服务器写字符串(键盘录入),服务器(多线程)将字符串反转后写回,客户端再次读取到是反转后的字符串

26.20_网络编程(练习)

- 客户端向服务器上传文件

26.21_day26总结

- 把今天的知识点总结一遍。

day27授课目录：

27.01_反射(类的加载概述和加载时机)

- A:类的加载概述
  - 当程序要使用某个类时，如果该类还未被加载到内存中，则系统会通过加载，连接，初始化三步来实现对这个类进行初始化。
  - 加载
    - 就是指将class文件读入内存，并为之创建一个Class对象。任何类被使用时系统都会建立一个Class对象。
  - 连接
    - 验证 是否有正确的内部结构，并和其他类协调一致
    - 准备 负责为类的静态成员分配内存，并设置默认初始化值
    - 解析 将类的二进制数据中的符号引用替换为直接引用
  - 初始化 就是我们以前讲过的初始化步骤
- B:加载时机
  - 创建类的实例
  - 访问类的静态变量，或者为静态变量赋值
  - 调用类的静态方法
  - 使用反射方式来强制创建某个类或接口对应的java.lang.Class对象
  - 初始化某个类的子类
  - 直接使用java.exe命令来运行某个主类

27.02_反射(类加载器的概述和分类)

- A:类加载器的概述
  - 负责将.class文件加载到内存中，并为之生成对应的Class对象。虽然我们不需要关心类加载机制，但是了解这个机制我们就能更好的理解程序的运行。
- B:类加载器的分类
  - Bootstrap ClassLoader 根类加载器
  - Extension ClassLoader 扩展类加载器
  - Sysetm ClassLoader 系统类加载器
- C:类加载器的作用
  - Bootstrap ClassLoader 根类加载器
    - 也被称为引导类加载器，负责Java核心类的加载
    - 比如System,String等。在JDK中JRE的lib目录下rt.jar文件中
  - Extension ClassLoader 扩展类加载器
    - 负责JRE的扩展目录中jar包的加载。
    - 在JDK中JRE的lib目录下ext目录
  - Sysetm ClassLoader 系统类加载器
    - 负责在JVM启动时加载来自java命令的class文件，以及classpath环境变量所指定的jar包和类路径

27.03_反射(反射概述)

- A:反射概述
  - JAVA反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；
  - 对于任意一个对象，都能够调用它的任意一个方法和属性；
  - 这种动态获取的信息以及动态调用对象的方法的功能称为java语言的反射机制。
  - 要想解剖一个类,必须先要获取到该类的字节码文件对象。
  - 而解剖使用的就是Class类中的方法，所以先要获取到每一个字节码文件对应的Class类型的对象。
- B:三种方式
  - a:Object类的getClass()方法,判断两个对象是否是同一个字节码文件
  - b:静态属性class,锁对象
  - c:Class类中静态方法forName(),读取配置文件
- C:案例演示
  - 获取class文件对象的三种方式

27.04_反射(Class.forName()读取配置文件举例)

- 榨汁机(Juicer)榨汁的案例
- 分别有水果(Fruit)苹果(Apple)香蕉(Banana)桔子(Orange)榨汁(squeeze)

```
public class Demo2_Reflect {

    /**
     * 榨汁机(Juicer)榨汁的案例
     * 分别有水果(Fruit)苹果(Apple)香蕉(Banana)桔子(Orange)榨汁(squeeze)
     * @throws Exception
```

```
     */
    public static void main(String[] args) throws Exception {
        /*Juicer j = new Juicer();
        //j.run(new Apple());
        j.run(new Orange());*/
        BufferedReader br = new BufferedReader(new FileReader("config.properties"));     //创建输入流对象,关联配置文件
        Class<?> clazz = Class.forName(br.readLine());                                   //读取配置文件一行内容,获取该类的字节码对象
        Fruit f = (Fruit) clazz.newInstance();                                           //通过字节码对象创建实例对象
        Juicer j = new Juicer();
        j.run(f);
    }

}
interface Fruit {
    public void squeeze();
}

class Apple implements Fruit {
    public void squeeze() {
        System.out.println("榨出一杯苹果汁儿");
    }
}

class Orange implements Fruit {
    public void squeeze() {
        System.out.println("榨出一杯桔子汁儿");
    }
}

class Juicer {
    public void run(Fruit f) {
        f.squeeze();
    }

}
```

27.05_反射(通过反射获取带参构造方法并使用)

- Constructor
  - Class类的newInstance()方法是使用该类无参的构造函数创建对象, 如果一个类没有无参的构造函数, 就不能这样创建了,可以调用Class类的getConstructor(String.class,int.class)方法获取一个指定的构造函数然后再调用Constructor类的newInstance("张三",20)方法创建对象

27.06_反射(通过反射获取成员变量并使用)

- Field
  - Class.getField(String)方法可以获取类中的指定字段(可见的), 如果是私有的可以用getDeclaedField("name")方法获取,通过set(obj, "李四")方法可以设置指定对象上该字段的值, 如果是私有的需要先调用setAccessible(true)设置访问权限,用获取的指定的字段调用get(obj)可以获取指定对象中该字段的值

27.07_反射(通过反射获取方法并使用)

- Method
  - Class.getMethod(String, Class...) 和 Class.getDeclaredMethod(String, Class...)方法可以获取类中的指定方法,调用invoke(Object, Object...)可以调用该方法,Class.getMethod("eat") invoke(obj) Class.getMethod("eat",int.class) invoke(obj,10)

27.08_反射(通过反射越过泛型检查)

- A:案例演示
  - ArrayList的一个对象，在这个集合中添加一个字符串数据，如何实现呢？

27.09_反射(通过反射写一个通用的设置某个对象的某个属性为指定的值)

- A:案例演示
  - public void setProperty(Object obj, String propertyName, Object value){}，此方法可将obj对象中名为propertyName的属性的值设置为value。

27.10_反射(练习)

- 已知一个类，定义如下：
  - package cn.itcast.heima;
  - public class DemoClass { public void run() { System.out.println("welcome to heima!"); } }
  - (1) 写一个Properties格式的配置文件，配置类的完整名称。
  - (2) 写一个程序，读取这个Properties配置文件，获得类的完整名称并加载这个类，用反射的方式运行run方法。

27.11_反射(动态代理的概述和实现)

- A:动态代理概述
  - 代理：本来应该自己做的事情，请了别人来做，被请的人就是代理对象。
  - 举例：春节回家买票让人代买
  - 动态代理：在程序运行过程中产生的这个对象,而程序运行过程中产生对象其实就是我们刚才反射讲解的内容，所以，动态代理其实就是通过反射来生成一个代理
  - 在Java中java.lang.reflect包下提供了一个Proxy类和一个InvocationHandler接口，通过使用这个类和接口就可以生成动态代理对象。JDK提供的代理只能针对接口做代理。我们有更强大的代理cglib，Proxy类中的方法创建动态代理类对象
  - public static Object newProxyInstance(ClassLoader loader,Class<?>[] interfaces,InvocationHandler h)
  - 最终会调用InvocationHandler的方法
  - InvocationHandler Object invoke(Object proxy,Method method,Object[] args)

27.12_设计模式(模版(Template)设计模式概述和使用)

- A:模版设计模式概述
  - 模版方法模式就是定义一个算法的骨架，而将具体的算法延迟到子类中来实现
- B:优点和缺点
  - a:优点
    - 使用模版方法模式，在定义算法骨架的同时，可以很灵活的实现具体的算法，满足用户灵活多变的需求
  - b:缺点
    - 如果算法骨架有修改的话，则需要修改抽象类 1,装饰 2,单例 3,简单工厂 4,工厂方法 5,适配器 6,模版

27.13_JDK5新特性(自己实现枚举类)

- A:枚举概述
  - 是指将变量的值一一列出来,变量的值只限于列举出来的值的范围内。举例：一周只有7天，一年只有12个月等。
- B:回想单例设计模式：单例类是一个类只有一个实例
  - 那么多例类就是一个类有多个实例，但不是无限个数的实例，而是有限个数的实例。这才能是枚举类。
- C:案例演示
  - 自己实现枚举类 1,自动拆装箱 2,泛型 3,可变参数 4,静态导入 5,增强for循环 6,互斥锁 7,枚举

27.14_JDK5新特性(通过enum实现枚举类)

- A:案例演示
  - 通过enum实现枚举类

27.15_JDK5新特性(枚举的注意事项)

- A:案例演示
  - 定义枚举类要用关键字enum
  - 所有枚举类都是Enum的子类
  - 枚举类的第一行上必须是枚举项，最后一个枚举项后的分号是可以省略的，但是如果枚举类有其他的东西，这个分号就不能省略。建议不要省略
  - 枚举类可以有构造器，但必须是private的，它默认的也是private的。
  - 枚举类也可以有抽象方法，但是枚举项必须重写该方法
  - 枚举在switch语句中的使用

27.16_JDK5新特性(枚举类的常见方法)

- A:枚举类的常见方法
  - int ordinal()
  - int compareTo(E o)
  - String name()
  - String toString()
  - T valueOf(Class type,String name)
  - values()
  - 此方法虽然在JDK文档中查找不到，但每个枚举类都具有该方法，它遍历枚举类的所有枚举值非常方便
- B:案例演示
  - 枚举类的常见方法

27.17_JDK7新特性(JDK7的六个新特性回顾和讲解)

- A:二进制字面量
- B:数字字面量可以出现下划线
- C:switch 语句可以用字符串
- D:泛型简化,菱形泛型
- E:异常的多个catch合并,每个异常用或|
- F:try-with-resources 语句

27.18_JDK8新特性(JDK8的新特性)

- 接口中可以定义有方法体的方法,如果是非静态,必须用default修饰
- 如果是静态的就不用了

```
class Test {
    public void run() {
        final int x = 10;
        class Inner {
            public void method() {
                System.out.println(x);
            }
        }

        Inner i = new Inner();
        i.method();
    }

}

局部内部类在访问他所在方法中的局部变量必须用final修饰,为什么?
因为当调用这个方法时,局部变量如果没有用final修饰,他的生命周期和方法的生命周期是一样的,当方法弹栈,这个局部变量也会消失,那么如果局部内部类对象还没有马上消
```

27.19_day27总结

- 把今天的知识点总结一遍。

第五章
面向对象

传智风清扬

本章内容

- 继承
- 多态
- 抽象类
- 接口
- 包和导包
- 权限修饰符
- 内部类

继承概述

- 继承概述
  - 多个类中存在相同属性和行为时，将这些内容抽取到单独一个类中，那么多个类无需再定义这些属性和行为，只要继承那个类即可。
  - 通过extends关键字可以实现类与类的继承
    - class 子类名 extends 父类名 {}
  - 单独的这个类称为父类，基类或者超类；这多个类可以称为子类或者派生类。
  - 有了继承以后，我们定义一个类的时候，可以在一个已经存在的类的基础上，还可以定义自己的新成员。

继承的案例和继承的好处

- 通过一个具体案例来演示代码
  - 案例1：学生类和老师。定义两个功能(吃饭，睡觉)
  - 案例2：加入人类后改进。
- 继承的好处
  - 提高了代码的复用性
    - 多个类相同的成员可以放到同一个类中
  - 提高了代码的维护性
    - 如果功能的代码需要修改，修改一处即可
  - 让类与类之间产生了关系，是多态的前提
    - 其实这也是继承的一个弊端：类的耦合性很强

Java中继承的特点

- Java只支持单继承，不支持多继承。
  - 一个类只能有一个父类，不可以有多个父类。
  - class SubDemo extends Demo{} //ok
  - class SubDemo extends Demo1,Demo2...//error
- Java支持多层继承(继承体系)
  - class A{}
  - class B extends A{}
  - class C extends B{}

Java中继承的注意事项

- 子类只能继承父类所有非私有的成员(成员方法和成员变量)
  - 其实这也体现了继承的另一个弊端：打破了封装性
- 子类不能继承父类的构造方法，但是可以通过super(后面讲)关键字去访问父类构造方法。
- 不要为了部分功能而去继承

- 我们到底在什么时候使用继承呢?
  - 继承中类之间体现的是：”is a”的关系。

继承中成员变量的关系

- 案例演示
  - 子父类中同名和不同名的成员变量
- 结论：
  - 在子类方法中访问一个变量
    - 首先在子类局部范围找
    - 然后在子类成员范围找
    - 最后在父类成员范围找(肯定不能访问到父类局部范围)
    - 如果还是没有就报错。(不考虑父亲的父亲…)

super关键字

- super的用法和this很像
  - this代表本类对应的引用。
  - super代表父类存储空间的标识(可以理解为父类引用)
- 用法(this和super均可如下使用)
  - 访问成员变量
    - this.成员变量　　super.成员变量
  - 访问构造方法(子父类的构造方法问题讲)
    - this(…)　　super(…)
  - 访问成员方法(子父类的成员方法问题讲)
    - this.成员方法()　　super.成员方法()

继承中构造方法的关系

- 子类中所有的构造方法默认都会访问父类中空参数的构造方法
- 为什么呢?
  - 因为子类会继承父类中的数据，可能还会使用父类的数据。所以，子类初始化之前，一定要先完成父类数据的初始化。
  - 每一个构造方法的第一条语句默认都是：super()

继承中构造方法的关系

- 如何父类中没有构造方法，该怎么办呢?
  - 子类通过super去显示调用父类其他的带参的构造方法
  - 子类通过this去调用本类的其他构造方法
    - 本类其他构造也必须首先访问了父类构造
  - 一定要注意：
    - super(…)或者this(….)必须出现在第一条语句山
    - 否则，就会有父类数据的多次初始化
- 看程序写结果

继承中成员方法的关系

- 案例演示
  - 子父类中同名和不同名的成员方法
- 结论：
  - 通过子类对象去访问一个方法
    - 首先在子类中找
    - 然后在父类中找
    - 如果还是没有就报错。(不考虑父亲的父亲…)

继承中成员方法的关系

- 方法重写概述
  - 子类中出现了和父类中一模一样的方法声明，也被称为方法覆盖，方法复写。
  - 使用特点：
    - 如果方法名不同，就调用对应的方法
    - 如果方法名相同，最终使用的是子类自己的
- 方法重写的应用：
  - 当子类需要父类的功能，而功能主体子类有自己特有内容时，可以重写父类中的方法，这样，即沿袭了父类的功能，又定义了子类特有的内容。

继承中成员方法的关系

- 方法重写的注意事项
  - 父类中私有方法不能被重写
  - 子类重写父类方法时，访问权限不能更低
  - 父类静态方法，子类也必须通过静态方法进行重写。(其实这个算不上方法重写，但是现象确实如此，至于为什么算不上方法重写，多态中我会讲解)

两个面试题

- 方法重写和方法重载的区别?方法重载能改变返回值类型吗?
  - Overload
  - Override
- this关键字和super关键字分别代表什么?以及他们各自的使用场景和作用。

继承练习

- 学生案例和老师案例讲解
  - 使用继承前
  - 使用继承后
    - 父类中成员private修饰，子类如何访问呢?
- 猫狗案例讲解
  - 分析和实现

final关键字

- final关键字是最终的意思，可以修饰类，成员变量，成员方法。
  - 修饰类，类不能被继承
  - 修饰变量，变量就变成了常量，只能被赋值一次
  - 修饰方法，方法不能被重写

final关键字

- final关键字面试题
  - final修饰局部变量
    - 在方法内部，该变量不可以被改变
    - 在方法声明上，分别演示基本类型和引用类型作为参数的情况
      - 基本类型，是值不能被改变
      - 引用类型，是地址值不能被改变
  - final修饰变量的初始化时机
    - 在对象构造完毕前即可

多态概述

- 多态概述
  - 某一个事物，在不同时刻表现出来的不同状态。
  - 举例：
    - 猫可以是猫的类型。猫 m = new 猫();
    - 同时猫也是动物的一种，也可以把猫称为动物。
      - 动物 d = new 猫();
    - 在举一个例子：水在不同时刻的状态
- 多态前提和体现
  - 有继承关系
  - 有方法重写
  - 有父类引用指向子类对象

多态案例及成员访问特点

- 多态案例
  - 按照前提写一个多态的案例
- 成员访问特点
  - 成员变量
    - 编译看左边，运行看左边
  - 成员方法
    - 编译看左边，运行看右边
  - 静态方法
    - 编译看左边，运行看左边
    - 所以前面我说静态方法不能算方法的重写

多态的好处和弊端

- 多态的好处
  - 提高了程序的维护性(由继承保证)
  - 提高了程序的扩展性(由多态保证)
- 多态的弊端
  - 不能访问子类特有功能
  - 那么我们如何才能访问子类的特有功能呢?
    - 多态中的转型

多态中的转型问题

- 向上转型
  - 从子到父
  - 父类引用指向子类对象
- 向下转型
  - 从父到子
  - 父类引用转为子类对象

- 多态成员访问及转型的理解
  - 孔子装爹案例

多态练习

- 猫狗案例练习多态版
- 不同地方饮食文化不同的案例
  - Person
    - eat()
  - SouthPerson
    - eat()
  - NorthPerson
    - eat()

抽象类概述

- 抽象类概述
  - 回想前面我们的猫狗案例，提取出了一个动物类。并且我们在前面也创建过了动物对象，其实这是不对的。为什么呢?因为，我说动物，你知道我说的是什么动物吗?只有看到了具体的动物，你才知道，这是什么动物。 所以说，动物本身并不是一个具体的事物，而是一个抽象的事物。只有真正的猫，狗才是具体的动物。同理，我们也可以推想，不同的动物吃的东西应该是不一样的，所以，我们不应该在动物类中给出具体体现，而是应该给出一个声明即可。

抽象类概述

- 抽象类概述
  - 回想前面我们的猫狗案例，提取出了一个动物类。并且我们在前面也创建过了动物对象，其实这是不对的。为什么呢?因为，我说动物，你知道我说的是什么动物吗?只有看到了具体的动物，你才知道，这是什么动物。 所以说，动物本身并不是一个具体的事物，而是一个抽象的事物。只有真正的猫，狗才是具体的动物。同理，我们也可以推想，不同的动物吃的东西应该是不一样的，所以，我们不应该在动物类中给出具体体现，而是应该给出一个声明即可。在Java中，一个没有方法体的方法应该定义为抽象方法，而类中如果有抽象方法，该类必须定义为抽象类。

抽象类特点

- 抽象类特点
  - 抽象类和抽象方法必须用abstract关键字修饰
    - 格式
    - abstract class 类名 {}
    - public abstract void eat();
  - 抽象类不一定有抽象方法，有抽象方法的类一定是抽象类
  - 抽象类不能实例化
    - 那么，抽象类如何实例化呢?
    - 按照多态的方式，由具体的子类实例化。其实这也是多态的一种，抽象类多态。
  - 抽象类的子类
    - 要么是抽象类
    - 要么重写抽象类中的所有抽象方法

抽象类的成员特点

- 成员变量
  - 可以是变量
  - 也可以是常量
- 构造方法
  - 有构造方法，但是不能实例化
  - 那么，构造方法的作用是什么呢?
    - 用于子类访问父类数据的初始化
- 成员方法
  - 可以有抽象方法 限定子类必须完成某些动作
  - 也可以有非抽象方法 提高代码服用性

抽象类练习

- 猫狗案例
  - 具体事物：猫，狗
  - 共性：姓名，年龄，吃饭
- 老师案例
  - 具体事物：基础班老师，就业班老师
  - 共性：姓名，年龄，讲课。
- 学生案例
  - 具体事务：基础班学员，就业班学员
  - 共性：姓名，年龄，班级，学习，吃饭
- 员工案例(备注部分)

抽象类的几个小问题

- 一个类如果没有抽象方法，可不可以定义为抽象类?如果可以，有什么意义?
- abstract不能和哪些关键字共存
  - private　冲突
  - final　冲突
  - static　无意义

接口概述

- 接口概述
  - 继续回到我们的猫狗案例，我们想想狗一般就是看门，猫一般就是作为宠物了，对不。但是，现在有很多的驯养员或者是驯兽师，可以训练出：猫钻火圈，狗跳高，狗做计算等。而这些额外的动作，并不是所有猫或者狗一开始就具备的，这应该属于经过特殊的培训训练出来的，对不。所以，这些额外的动作定义到动物类中就不合适，也不适合直接定义到猫或者狗中，因为只有部分猫狗具备这些功能。所以，为了体现事物功能的扩展性，Java中就提供了接口来定义这些额外功能，并不给出具体实现，将来哪些猫狗需要被培训，只需要这部分猫狗把这些额外功能实现即可。

接口特点

- 接口特点
  - 接口用关键字interface表示
    - 格式：interface 接口名 {}
  - 类实现接口用implements表示
    - 格式：class 类名 implements 接口名 {}
  - 接口不能实例化
    - 那么，接口如何实例化呢?
    - 按照多态的方式，由具体的子类实例化。其实这也是多态的一种，接口多态。
  - 接口的子类
    - 要么是抽象类
    - 要么重写接口中的所有抽象方法

接口成员特点

- 成员变量
  - 只能是常量
  - 默认修饰符 public static final
- 构造方法
  - 没有，因为接口主要是扩展功能的，而没有具体存在
- 成员方法
  - 只能是抽象方法
  - 默认修饰符 public abstract

类与类,类与接口以及接口与接口的关系

- 类与类
  - 继承关系，只能单继承，但是可以多层继承
- 类与接口
  - 实现关系，可以单实现，也可以多实现。还可以在继承一个类的同时实现多个接口
- 接口与接口
  - 继承关系，可以单继承，也可以多继承

抽象类和接口的区别

- 成员区别
  - 抽象类 变量,常量;有抽象方法;抽象方法,非抽象方法
  - 接口 常量;抽象方法
- 关系区别
  - 类与类 继承，单继承
  - 类与接口 实现，单实现，多实现
  - 接口与接口 继承，单继承，多继承
- 设计理念区别
  - 抽象类 被继承体现的是：”is a”的关系。共性功能
  - 接口 被实现体现的是：”like a”的关系。扩展功能

接口练习

- 猫狗案例,加入跳高的额外功能
- 老师和学生案例,加入抽烟的额外功能
- 教练和运动员案例(学生分析然后讲解)
  - 乒乓球运动员和篮球运动员。
  - 乒乓球教练和篮球教练。
  - 为了出国交流，跟乒乓球相关的人员都需要学习英语。
  - 请用所学知识：
  - 分析，这个案例中有哪些抽象类，哪些接口，哪些具体类。

形式参数和返回值问题案例

- 形式参数
  - 基本类型
  - 引用类型
- 返回值类型
  - 基本类型
  - 引用类型
- 链式编程

包

- 包的概述
  - 其实就是文件夹
  - 作用：对类进行分类管理
  - 包的划分：
    - 举例：
      - 学生的增加，删除，修改，查询
      - 老师的增加，删除，修改，查询
      - 以及以后可能出现的其他的类的增加，删除，修改，查询
      - 基本的划分：按照模块和功能分。
      - 高级的划分：就业班做项目的时候你就能看到了。

包的定义及注意事项

- 定义包的格式
  - package 包名;
    - 多级包用.分开即可
  - 注意事项：
    - package语句必须是程序的第一条可执行的代码
    - package语句在一个java文件中只能有一个
    - 如果没有package，默认表示无包名

带包的类的编译和运行

- 手动式
  - a:javac编译当前类文件。
  - b:手动建立包对应的文件夹。
  - c:把a步骤的class文件放到b步骤的最终文件夹下。
  - d:通过java命令执行。注意了：需要带包名称的执行
    - java cn.itcast.HelloWorld
- 自动式
  - a:javac编译的时候带上-d即可
    - javac -d . HelloWorld.java
  - b:通过java命令执行。和手动式一样

不同包下类之间的访问

- 定义两个类：Demo,Test。
  - Demo
    - 求和方法(sum)
  - Test
    - 测试方法(main)

导包

- 导包概述
  - 不同包下的类之间的访问，我们发现，每次使用不同包下的类的时候，都需要加包的全路径。比较麻烦。这个时候，java就提供了导包的功能。
- 导包格式
  - import 包名;
  - 注意：
    - 这种方式导入是到类的名称。
    - 虽然可以最后写*，但是不建议。
- package,import,class有没有顺序关系(面试题)

权限修饰符

	public	protected	默认	private
同一类中	√	√	√	√
同一包子类,其他类	√	√	√	
不同包子类	√	√		
不同包其他类	√			

类及其组成可以用的修饰符

- 类：
  - 默认，public，final，abstract
  - 我们自己定义：public居多
- 成员变量：
  - 四种权限修饰符均可,final,static
  - 我们自己定义：private居多
- 构造方法：
  - 四种权限修饰符均可,其他不可
  - 我们自己定义：public 居多
- 成员方法：
  - 四种权限修饰符均可，fianl,static,abstract
  - 我们自己定义：public居多

内部类概述

- 把类定义在其他类的内部，这个类就被称为内部类。
  - 举例：在类A中定义了一个类B，类B就是内部类。
- 内部类的访问特点：
  - 内部类可以直接访问外部类的成员，包括私有。
  - 外部类要访问内部类的成员，必须创建对象。

内部类位置

- 按照内部类在类中定义的位置不同，可以分为如下两种格式：
  - 成员位置(成员内部类)
  - 局部位置(局部内部类)

- 成员内部类
  - 外界如何创建对象
    - 外部类名.内部类名 对象名 = 外部类对象.内部类对象;

成员内部类

- 刚才我们讲解过了，成员内部类的使用，但是一般来说，在实际开发中是不会这样使用的。因为一般内部类就是不让外界直接访问的。
  - 举例讲解这个问题：Body和Heart，电脑和CPU。
- 成员内部的常见修饰符
  - private 为了保证数据的安全性
  - static 为了让数据访问更方便
    - 被静态修饰的成员内部类只能访问外部类的静态成员
    - 内部类被静态修饰后的方法
      - 静态方法
      - 非静态方法

成员内部类面试题

- 补齐程序(注意:内部类和外部类没有继承关系)

```
class Outer {
        public int num = 10;
        class Inner {
                public int num = 20;
                public void show() {
                        int num = 30;
                        System.out.println(?);
                        System.out.println(??);
                        System.out.println(???);
                }
        }
}
```

在控制分别输出：30，20，10

局部内部类

- 可以直接访问外部类的成员
- 可以创建内部类对象，通过对象调用内部类方法，来使用局部内部类功能
- 局部内部类访问局部变量的注意事项：
  - 必须被final修饰?
  - 为什么呢?
    - 因为局部变量会随着方法的调用完毕而消失，这个时候，局部对象并没有立马从堆内存中消失，还要使用那个变量。为了让数据还能继续被使用，就用fianl修饰，这样，在堆内存里面存储的其实是一个常量值。通过反编译工具可以看一下。

匿名内部类

- 就是内部类的简化写法。
- 前提：存在一个类或者接口
  - 这里的类可以是具体类也可以是抽象类。
- 格式：

  new 类名或者接口名() {重写方法;}
- 本质：
  - 是一个继承了类或者实现了接口的子类匿名对象

匿名内部类案例

- 写案例，并测试
  - 如何调用方法
  - 加入方法有多个，如何调用呢?
    - 方式1：每一种格式调用一个，太麻烦
    - 方式2：用类或者接口接收该子类对象，多态思想

匿名内部类在开发中的使用

- 首先回顾我们曾经讲过的方法的形式参数是引用类型的情况，重点是接口的情况，我们知道这里需要一个子类对象。而匿名内部类就是一个子类匿名对象，所以，可以使用匿名内部类改进以前的做法。

匿名内部类面试题

- 按照要求，补齐代码

```
interface Inter { void show(); }
class Outer { //补齐代码 }
class OuterDemo {
    public static void main(String[] args) {
        Outer.method().show();
    }
}
```

要求在控制台输出”HelloWorld”

API-常用类

传智风清扬

本章内容

- API概述
- 常用类
  - Object类/Scanner类
  - String类/StringBuffer类/StringBuilder类
  - 数组高级和Arrays类
  - 基本类型包装类(Integer,Character)
  - 正则表达式(Pattern,Matcher)
  - Math类/Random类/System类
  - BigInteger类/BigDecimal类
  - Date类/DateFormat类/Calendar类

API概述

- API(Application Programming Interface)
  - 应用程序编程接口
  - 编写一个机器人程序去控制机器人踢足球，程序就需要向机器人发出向前跑、向后跑、射门、抢球等各种命令，没有编过程序的人很难想象这样的程序如何编写。但是对于有经验的开发人员来说，知道机器人厂商一定会提供一些用于控制机器人的Java类，这些类中定义好了操作机器人各种动作的方法。其实，这些Java类就是机器人厂商提供给应用程序编程的接口，大家把这些类称为Xxx Robot API。本章涉及的Java API指的就是JDK中提供的各种功能的Java类。

学习汉语和学习编程的异同点

- 相同点
  - 基本语法
  - 大量成语
  - 写文章的手法和技巧
- 不同点
  - 学习汉语 必须先学后用
  - 学习编程 可以现用现学

Object类概述及其构造方法

- Object类概述
  - 类层次结构的根类
  - 所有类都直接或者间接的继承自该类
- 构造方法
  - public Object()
  - 回想面向对象中为什么说：
    - 子类的构造方法默认访问的是父类的无参构造方法

Object类的成员方法

- public int hashCode()
- public final Class getClass()
- public String toString()
- public boolean equals(Object obj)
- protected void finalize()
- protected Object clone()

Scanner类概述及其构造方法

- Scanner类概述
  - JDK5以后用于获取用户的键盘输入
- 构造方法
  - public Scanner(InputStream source)

Scanner类的成员方法

- 基本格式
  - hasNextXxx() 判断是否还有下一个输入项,其中Xxx可以是Int,Double等。如果需要判断是否包含下一个字符串，则可以省略Xxx
  - nextXxx() 获取下一个输入项。Xxx的含义和上个方法中的Xxx相同
  - 默认情况下，Scanner使用空格，回车等作为分隔符
- 常用方法
  - public int nextInt()
  - public String nextLine()

String类概述及其构造方法

- String类概述
  - 字符串是由多个字符组成的一串数据(字符序列)
  - 字符串可以看成是字符数组
- 构造方法
  - public String()
  - public String(byte[] bytes)
  - public String(byte[] bytes,int offset,int length)
  - public String(char[] value)
  - public String(char[] value,int offset,int count)
  - public String(String original)

String类的特点及面试题

- 字符串是常量,它的值在创建之后不能更改
  - String s = “hello”; s += “world”; 问s的结果是多少?
- 面试题
  - String s = new String(“hello”)和String s = “hello”;的区别?
  - 字符串比较之看程序写结果
  - 字符串拼接之看程序写结果

String类的判断功能

- boolean equals(Object obj)
- boolean equalsIgnoreCase(String str)
- boolean contains(String str)
- boolean startsWith(String str)
- boolean endsWith(String str)
- boolean isEmpty()

String类的获取功能

- int length()
- char charAt(int index)
- int indexOf(int ch)
- int indexOf(String str)
- int indexOf(int ch,int fromIndex)
- int indexOf(String str,int fromIndex)
- String substring(int start)
- String substring(int start,int end)

String类的转换功能

- byte[] getBytes()
- char[] toCharArray()
- static String valueOf(char[] chs)
- static String valueOf(int i)
- String toLowerCase()
- String toUpperCase()
- String concat(String str)

String类的其他功能

- 替换功能
  - String replace(char old,char new)
  - String replace(String old,String new)
- 去除字符串两空格
  - String trim()
- 按字典顺序比较两个字符串
  - int compareTo(String str)
  - int compareToIgnoreCase(String str)

String类练习

- 把数组中的数据按照指定个格式拼接成一个字符串
  - 举例：int[] arr = {1,2,3};　　输出结果：[1, 2, 3]
- 字符串反转
  - 举例：键盘录入”abc”　　输出结果：”cba”
- 统计大串中小串出现的次数
  - 举例：在字符串”woaijavawozhenaijavawozhendeaijavawozhendehenaijavaxinbuxinwoaijavagun”中java出现了5次

StringBuffer类概述及其构造方法

- StringBuffer类概述
  - 我们如果对字符串进行拼接操作，每次拼接，都会构建一个新的String对象，既耗时，又浪费空间。而StringBuffer就可以解决这个问题
  - 线程安全的可变字符序列
- StringBuffer和String的区别?
- 构造方法
  - public StringBuffer()
  - public StringBuffer(int capacity)
  - public StringBuffer(String str)

StringBuffer类的成员方法

- 添加功能
  - public StringBuffer append(String str)
  - public StringBuffer insert(int offset,String str)
- 删除功能
  - public StringBuffer deleteCharAt(int index)
  - public StringBuffer delete(int start,int end)
- 替换功能
  - public StringBuffer replace(int start,int end,String str)
- 反转功能 public StringBuffer reverse()

StringBuffer类的成员方法

- 截取功能
  - public String substring(int start)
  - public String substring(int start,int end)
- 截取功能和前面几个功能的不同
  - 返回值类型是String类型，本身没有发生改变

StringBuffer类练习

- String和StringBuffer的相互转换
- 把数组拼接成一个字符串
- 把字符串反转
- 判断一个字符串是否是对称字符串
  - 例如"abc"不是对称字符串，"aba"、"abba"、"aaa"、"mnanm"是对称字符串

StringBuffer类面试题

- 通过查看API了解一下StringBuilder类
- String,StringBuffer,StringBuilder的区别
- StringBuffer和数组的区别
- 看程序写结果：
  - String作为参数传递
  - StringBuffer作为参数传递

数组高级(排序和查找)

- 排序
  - 冒泡排序
    - 相邻元素两两比较，大的往后放，第一次完毕，最大值出现在了最大索引处
  - 选择排序
    - 从0索引开始，依次和后面元素比较，小的往前放，第一次完毕，最小值出现在了最小索引处
- 查找
  - 基本查找 数组元素无序
  - 二分查找 数组元素有序

数组高级练习题

- 把字符串中的字符进行排序。
  - 举例：”dacgebf”
  - 结果：”abcdefg”

Arrays类概述及其常用方法

- Arrays类概述
  - 针对数组进行操作的工具类。
  - 提供了排序，查找等功能。
- 成员方法
  - public static String toString(int[] a)
  - public static void sort(int[] a)
  - public static int binarySearch(int[] a,int key)

Arrays类常用方法源码详细解释

- public static String toString(int[] a)
  源码解析
- public static int binarySearch(int[] a,int key)
  源码解析

基本类型包装类概述

- 将基本数据类型封装成对象的好处在于可以在对象中定义更多的功能方法操作该数据。
- 常用的操作之一：用于基本数据类型与字符串之间的转换。
- 基本类型和包装类的对应
  - Byte,Short,Integer,Long,Float,Double
    Character,Boolean

Integer类概述及其构造方法

- Integer类概述
  - Integer 类在对象中包装了一个基本类型 int 的值
  - 该类提供了多个方法，能在 int 类型和 String 类型之间互相转换，还提供了处理 int 类型时非常有用的其他一些常量和方法
- 构造方法
  - public Integer(int value)
  - public Integer(String s)

Integer类成员方法

- int类型和String类型的相互转换
  - int – String
  - String – int

- public int intValue()
- public static int parseInt(String s)
- public static String toString(int i)
- public static Integer valueOf(int i)
- public static Integer valueOf(String s)

Integer类成员方法

- 常用的基本进制转换
  - public static String toBinaryString(int i)
  - public static String toOctalString(int i)
  - public static String toHexString(int i)
- 十进制到其他进制
  - public static String toString(int i,int radix)
- 其他进制到十进制
  - public static int parseInt(String s,int radix)

JDK5的新特性

- JDK1.5以后，简化了定义方式。
  - Integer x = new Integer(4);可以直接写成
  - Integer x = 4;//自动装箱。
  - x = x + 5;//自动拆箱。通过intValue方法。
- 需要注意：
  - 在使用时，Integer x = null;上面的代码就会出现NullPointerException。

Integer的面试题

- Integer i = 1; i += 1;做了哪些事情
- 缓冲池(看程序写结果)
  - 通过查看源码知道为什么

Character类概述及其构造方法

- Character类概述
  - Character 类在对象中包装一个基本类型 char 的值
  - 此外，该类提供了几种方法，以确定字符的类别（小写字母，数字，等等），并将字符从大写转换成小写，反之亦然
- 构造方法
  - public Character(char value)

Character类成员方法

- public static boolean isUpperCase(char ch)
- public static boolean isLowerCase(char ch)
- public static boolean isDigit(char ch)
- public static char toUpperCase(char ch)
- public static char toLowerCase(char ch)

正则表达式概述及基本使用

- 正则表达式：是指一个用来描述或者匹配一系列符合某个句法规则的字符串的单个字符串。其实就是一种规则。有自己特殊的应用。

- 举例：校验qq号码.

  1:要求必须是5-15位数字

  2:0不能开头

正则表达式的组成规则

- 规则字符在java.util.regex Pattern类中
- 常见组成规则
  - 字符
  - 字符类
  - 预定义字符类
  - 边界匹配器
  - 数量词

正则表达式的应用

- 判断功能
  - public boolean matches(String regex)
- 分割功能
  - public String[] split(String regex)
- 替换功能
  - public String replaceAll(String regex,String replacement)
- 获取功能
  - Pattern和Matcher类的使用

正则表达式的练习

- 判断功能：
  - 校验邮箱
- 分割功能：
  - 我有如下一个字符串:”91 27 46 38 50”
  - 请写代码实现最终输出结果是：”27 38 46 50 91”
- 替换功能：
  - 论坛中不能出现数字字符，用*替换
- 获取功能：
  - 获取由三个字符组成的单词

Math类概述及其成员方法

- Math类概述
  - Math 类包含用于执行基本数学运算的方法，如初等指数、对数、平方根和三角函数。
- 成员方法
  - public static int abs(int a)
  - public static double ceil(double a)
  - public static double floor(double a)
  - public static int max(int a,int b) min自学
  - public static double pow(double a,double b)
  - public static double random()
  - public static int round(float a) 参数为double的自学
  - public static double sqrt(double a)

Random类概述及其构造方法

- Random类概述
  - 此类用于产生随机数
  - 如果用相同的种子创建两个 Random 实例，则对每个实例进行相同的方法调用序列，它们将生成并返回相同的数字序列。
- 构造方法
  - public Random()
  - public Random(long seed)

Random类成员方法

- public int nextInt()
- public int nextInt(int n)

System类概述及其成员方法

- System类概述
  - System 类包含一些有用的类字段和方法。它不能被实例化。
- 成员方法
  - public static void gc()
  - public static void exit(int status)
  - public static long currentTimeMillis()
  - public static void arraycopy(Object src,int srcPos,Object dest,int destPos,int length)

BigInteger类概述及其构造方法

- BigInteger类概述
  - 可以让超过Integer范围内的数据进行运算
- 构造方法
  - public BigInteger(String val)

BigInteger类成员方法

- public BigInteger add(BigInteger val)
- public BigInteger subtract(BigInteger val)
- public BigInteger multiply(BigInteger val)
- public BigInteger divide(BigInteger val)
- public BigInteger[] divideAndRemainder(BigInteger val)

BigDecimal类概述及其构造方法

- 由于在运算的时候，float类型和double很容易丢失精度，演示案例。所以，为了能精确的表示、计算浮点数，Java提供了BigDecimal
- BigDecimal类概述
  - 不可变的、任意精度的有符号十进制数。
- 构造方法
  - public BigDecimal(String val)

BigDecimal类成员方法

- public BigDecimal add(BigDecimal augend)
- public BigDecimal subtract(BigDecimal subtrahend)
- public BigDecimal multiply(BigDecimal multiplicand)
- public BigDecimal divide(BigDecimal divisor)
- public BigDecimal divide(BigDecimal divisor,int scale, int roundingMode)

Date类概述及其方法

- Date类概述
  - 类 Date 表示特定的瞬间，精确到毫秒。
- 构造方法
  - public Date()
  - public Date(long date)
- 成员方法
  - public long getTime()
  - public void setTime(long time)

DateFormat类概述及其方法

- DateFormat类概述
  - DateFormat 是日期/时间格式化子类的抽象类，它以与语言无关的方式格式化并解析日期或时间。
  - 是抽象类，所以使用其子类SimpleDateFormat
- SimpleDateFormat构造方法
  - public SimpleDateFormat()
  - public SimpleDateFormat(String pattern)
- 成员方法
  - public final String format(Date date)
  - public Date parse(String source)

Date类及DateFormat类练习

- 制作一个工具类。DateUtil
- 算一下你来到这个世界多少天?

Calendar类概述及其方法

- Calendar类概述
  - Calendar 类是一个抽象类，它为特定瞬间与一组诸如 YEAR、MONTH、DAY_OF_MONTH、HOUR 等日历字段之间的转换提供了一些方法，并为操作日历字段（例如获得下星期的日期）提供了一些方法。
- 成员方法
  - public static Calendar getInstance()
  - public int get(int field)
  - public void add(int field,int amount)
  - public final void set(int year,int month,int date)

Calendar类练习

- 算一下你来到这个世界多少天?
- 获取任意一年的二月有多少天