# 第1集-登陆新手村：JAVA的背景知识和JDK的安装和配置

1. 目前企业最流行的JAVA版本是？

JAVA8

1. JVM、JRE、JDK的全称分别是？

JVM: Java Virtual Machine

JRE: Java Runtime Environment

JDK: Java Development Kit

3. 实操：完成JDK的安装和配置。

1. 下载

2. 安装（和安装一般软件一样。建议安装到默认目录）

3. 配置环境变量

(1) 新建系统变量：JAVA\_HOME

(2) 修改path变量：%JAVA\_HOME%\bin

# 第2集-写出第一个JAVA程序和错误调试技巧

1. 完成自己的第一个JAVA程序（实操）

|  |
| --- |
| public class Welcome{  public static void main(String[] args){  System.out.println("Hello world！");  }  } |

1. 写出JAVA程序的运行过程（简答题）
2. 进入java源文件的目录
3. javac Welcome.java
4. java Welcome

3. 写出解决第一个bug的过程（简答题）

根据自己遇到的问题解决步骤，自己完成。

# 第3集-名字很重要：标识符\_变量\_JAVA关键字\_注释

**作业**

1. 如下合法的标识符是哪些？
2. age B. $123 C. 3a D. \_go E. 年龄
3. Salary#

答案：A,B,D,E

1. 针对自己的JAVA程序，写出单行、多行注释【实操】

略。简单写出注释即可。

1. 高猿猿在京东上班，月薪3万，每年15薪。计算出我的年收入。 【实操】

|  |
| --- |
| int monthlySalary = 30000; int monthNum = 15; int yearSalary = monthlySalary\*monthNum; System.*out*.println(yearSalary); |

# 第4集-盘点武器库：（数据类型有八种）基本数据类型\_整型\_浮点型(上)

**作业**

1. byte、short、int、long分别占用几个字节？以及表数范围是多少？【简答】

|  |  |  |
| --- | --- | --- |
| **类型** | **占用存储空间** | **表数范围** |
| byte | 1字节 | -27 ~ 27-1（-128~127） |
| short | 2字节 | -215 ~ 215-1 （-32768~32767） |
| int | 4字节 | -231 ~ 231-1 (-2147483648~2147483647) 约21亿 |
| long | 8字节 | -263 ~ 263-1 |

1. float、double分别占用几个字节？以及表述范围是多少？【简答】

|  |  |  |
| --- | --- | --- |
| **类型** | **占用存储空间** | **表数范围** |
| float | 4字节 | -3.403E38~3.403E38 |
| double | 8字节 | -1.798E308~1.798E308 |

1. 我拥有3万个比特币，按照今日市价，折算成人民币，我的身价是多少？如果用整型变量表示，用哪种类型合适？【实操】

|  |
| --- |
| int bitcoinNum = 30000;  int bitcoinValue = 50000; *//2020-3-8，网上查询的价格。按照整数计算,5万美金 // int myWorth = bitcoinNum\*bitcoinValue; //身价15亿。考虑到比特币升值情况，很快突破20亿。int就不够了，建议使用long* long myWorth = bitcoinNum\*bitcoinNum;   System.*out*.println(myWorth); |

1. long和float的表数范围，谁更大？【简答】

float是浮点数表数不精确，表数范围更大。

# 第5集-盘点武器库：(万物皆可数字化)\_字符集char\_boolean(下)

**作业**

1. char类型采用的是什么字符集？【简答】

Unicode字符集

1. 完成课堂上的测试代码【实操】

|  |
| --- |
| public class TestChar {  public static void main(String[] args) {    //测试char  char c1 = 'A';  char c2 = '高';    System.out.println(c1);  System.out.print(c2);    char c3 = '\n';  char c4 = '淇';  System.out.print(c2);  System.out.print(c3);  System.out.print(c4);    //字符串不是基本数据类型，是类！  String str = "\n\n我爱你，\t\t\"中国\"！China！";  System.out.print(str);      }      } |

1. 说明一下：ASCII、ISO88591、GBK、Unicode、UTF-8的关系？【简答】
2. ASCII字符集表示了英文字母、数字、特殊字符、控制符，所有字符集的老祖宗，大家都会兼容它。
3. ISO8859-1,别名叫latin-1, 包含了256个字符。前128个字符与ASCII中完全相同。后128个包括了西欧语言、希腊语、泰语、阿拉伯语、希伯来语对应的文字符号
4. GB2312/GBK、GB18030：两个字节表示1个汉字。两个字节可以表示65536个状态，汉字再多也能全部包含。后来，又有了。
5. Unicode字符集是为了给全世界所有字符一个唯一的编码，两个字节表示1个字符。

# 第6集-七十二变运算符基本运算符

1. 位运算&符号、&逻辑运算两个&符号它们区别

位运算&两边可以是数字，按位对数字进行运算。

&逻辑运算：是对boolean进行运算。

&&短路与：是逻辑运算的短路运算。

⒉.变量a小于100大于50如何表示

a<100&&a>50

3.描述一下短路与、短路或

短路与：前一个结果是false，则后面不做运算，返回false。

短路或：前一个结果是true，则后面不做运算，返回true。

4.完成老师课堂上敲的测试代码

|  |
| --- |
| public class TestOperator {  public static void main(String[] args) {  System.out.println("=======算术运算符========");  int a = 3;  int b = 4;  int c = (a+b)\*4;  System.out.println(c);  int d = 15/4;  System.out.println(d);  int d2 = 5%3; //结果是：余数2  System.out.println(d2);    a = 10;  //b = a++; //先赋值，后自增  //b = ++a; //先自增，后赋值  b = a--; //b = --a;      System.out.println(a);  System.out.println(b);    System.out.println("=======扩展运算符========");  a = 20;  b = 30;  a += b; //a = a+b;  System.out.println(a);  System.out.println(b);    System.out.println("=======关系运算符========");  a = 20;  b = 30;  boolean result = a<b;  System.out.println(result);    System.out.println("=======逻辑运算符========");  boolean b1 = true & false; //false  boolean b2 = true | false; //true  boolean b3 = !b2; //false  boolean b4 = true^true; //false    System.out.println(b1);  System.out.println(b2);  System.out.println(b3);  System.out.println(b4);      //短路与、短路或  // int w = 3/0;  boolean b5 = 3<4 || (4<4/0);  System.out.println(b5);    System.out.println("=======位运算符========");  int m = 3;  int n = 7;  int p2 = m|n;  int p1 = m&n;  int p3 = m^n; //^异或的意思；不是数学中的幂运算  int p4 = ~m; //按位取反    System.out.println(p1);  System.out.println(p2);  System.out.println(p3);  System.out.println(p4);    int m2 = 3<<3; //3\*2\*2\*2 24  int m3 = 12>>2; //12/2/2 3    System.out.println(m2);  System.out.println(m3);    System.out.println("=======字符串连接符========");  int r1 = 3;  int r2 = 4;  System.out.println(r1+r2); //7  System.out.println("结果是："+r1+r2); //结果是：34        System.out.println("=======条件运算符========");  int y1 = 300;  int y2 = 40;  int minY = y1<y2?y1:y2; //总是返回y1和y2比较小的值    System.out.println(minY);    }    } |

# 第7集-类型转换小暗器：数据类型自动和强制转换

1.人的心跳一分钟80次，人的一生活了80岁,计算出一共心跳多少次?

|  |
| --- |
| int heartbeatPerMinute = 80; *//每分钟心跳次数* int age = 80; long heartbeatNum = 1L\*heartbeatPerMinute\*80\*365\*24\*60; System.*out*.println(heartbeatNum); |

⒉.用户名、年龄、月薪整个的输入程序

|  |
| --- |
| /\*\*  \* 测试键盘输入  \*/  import java.util.Scanner;  public class TestSystemIn {    public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  System.out.println("请输入您的账户名：");  String username = scanner.nextLine();  System.out.println("请输入年龄：");  int age = scanner.nextInt();  System.out.println("请输入月薪：");  double salary = scanner.nextDouble();  System.out.println("=======录入的信息如下=========");  System.out.println("账户名："+username);  System.out.println("年龄："+age);  System.out.println("年薪："+(salary\*12));    }  } |

# 第8集-开发神器帮大忙：IDEA的安装和使用

1. 安装和配置IDEA

根据课堂视频，完成操作。

1. 完成你的第一个Java项目

根据课堂视频，完成操作。

# 第9集-正确选择是胜利：控制语句\_if\_else

1. 随机生成0-100的分数如果60以下则输出不及格；60-69则输出一般；70-79输出良好；80-89输出优秀；90以上输出天才；

|  |
| --- |
| int score = (int)(Math.*random*()\*100); System.*out*.println("分数："+score); if(score<60){  System.*out*.println("不及格！"); }else if(score<70){  System.*out*.println("一般！"); }else if(score<80){  System.*out*.println("良好！"); }else if(score<90) {  System.*out*.println("优秀！"); }else{  System.*out*.println("天才！"); } |

# 第10集-值选择真特殊：控制语句\_switch

1. 完成老师课堂上关于switch的代码

|  |
| --- |
| /\*\*  \* 测试switch语句  \*/  public class TestSwitch {  public static void main(String[] args) {  //grade表示大学年级  int grade = 1;  if(grade==1) {  System.out.println("大学一年级，可以放松一下，学着谈谈恋爱");  }else if(grade==2){  System.out.println("大学二年级，少玩点游戏，不空虚，不慌嘛？");  }else if(grade==3) {  System.out.println("大学三年级，专业课开始了，好好学，找份好工作");  }else{  System.out.println("大四了，要毕业了。因为跟着尚学堂学习，好工作搞定！");  }  switch (grade){  case 1:  System.out.println("大学一年级！");  break;  case 2:  System.out.println("大学二年级！");  break;  case 3:  System.out.println("大学三年级！");  break;  default:  System.out.println("大学四年级");  break;  }  int month = 2; //1表示1月，2表示2月...  if(month==1||month==2||month==3) {  System.out.println("春季！");  }else if(month==4||month==5||month==6){  System.out.println("夏季！");  }else if(month==7||month==8||month==9){  System.out.println("秋季！");  }else{  System.out.println("冬季！");  }  switch (month) {  case 1:  case 2:  case 3:  System.out.println("春季！");  break;  case 4:  case 5:  case 6:  System.out.println("夏季！");  break;  case 7:  case 8:  case 9:  System.out.println("秋季！");  break;  default:  System.out.println("冬季！");  break;  }  String car = "比亚迪";  //switch表达式接收的：整数、枚举、字符串  switch (car){  case "奥迪":  System.out.println("我买了一个奥迪车！");  break;  case "奔驰":  System.out.println("买了一个大奔！");  break;  case "比亚迪":  System.out.println("比亚迪，挺好！");  break;  default:  System.out.println("吉利！");  break;  }  }  } |

1. 使用switch语句完成。生成代表星期的随机数字（0代表星期日，1代表星期一，2代表星期二，...）。如果是1，则输出“星期一”，如果是2，则输出“星期二”，以此类推。

|  |
| --- |
| *//生成代表星期的随机数字（0代表星期日，1代表星期一，2代表星期二，...）* int weekday = (int)(Math.*random*()\*7); System.*out*.println("随机数："+weekday);  String weekStr = ""; switch (weekday){  case 0:  weekStr="星期日";  break;  case 1:  weekStr="星期一";  break;  case 2:  weekStr="星期二";  break;  case 3:  weekStr="星期三";  break;  case 4:  weekStr="星期四";  break;  case 5:  weekStr="星期五";  break;  case 6:  weekStr="星期六";  break; } System.*out*.println(weekStr); |

# 第11集-重复一定生力量：控制语句\_循环(while/for)

1. 分别使用while和for循环，打印出1-100之间的数字

|  |
| --- |
| int i=1; while(i<=100){  System.*out*.println(i);  i++; } |
| *for(int i=0;i<=100;i++){  System.out.println(i); }* |

1. 分别使用while和for循环，计算1+2+...+100的和。

|  |
| --- |
| int i=0;  int sum = 0;  while(i<=100){ *// sum = sum + i;* sum += i;  i++;  }   System.*out*.println("和："+sum); |
| int sum02 = 0; for(int m=0;m<=100;m++) {  sum02 += m; } System.*out*.println(sum02); |

# 第12集-重复重复成天才：控制语句\_嵌套循环

1. 使用嵌套循环，打印输出5\*5的排版方阵，格式如下：

\* # \* # \*

# \* # \* #

\* # \* # \*

# \* # \* #

\* # \* # \*

|  |
| --- |
| for(int i=0;i<5;i++) {  for(int j=0;j<5;j++) {  if(j%2==0) {  System.*out*.print("\*\t");  }else{  System.*out*.print("#\t");  }  }  System.*out*.println(); } |

# 第13集-停止重复想静静：break和continue语句

1. 产生100以内的随机数，直到随机数为88时终止循环

|  |
| --- |
| public static void main(String[] args) {  //产生100以内的随机数，直到随机数为88时终止循环  int count = 0;  while (true) {  int m = (int)(Math.random()\*100);  System.out.println(m);  count++;  if(m==88){  break;  }  }  System.out.println("一共循环次数："+count);  } |

1. 把100~150之间不能被3整除的数输出，并且每行输出5个

|  |
| --- |
| public static void main(String[] args) {  //把100~150之间不能被3整除的数输出，并且每行输出5个  int count = 0; //计数器：表示每行输出几个  for(int i=100;i<=150;i++) {  if(i%3==0) {  continue;  }  System.out.print(i+"\t");  count++;  if(count%5==0) {  System.out.println();  }  }  } |

# 第14集 融会贯通任督脉：控制语句综合练习

1. 使用for循环，打印出从a-z的26个字母。

|  |
| --- |
| //使用for循环，打印出从a-z的26个字母  char c = 'a';  for(int i=0;i<26;i++){  char temp = (char)(c+i);  System.out.print(temp+",");  } |

1. 打印输出9×9乘法表

|  |
| --- |
| //打印99乘法表  for(int n=1;n<=9;n++) {  for(int m=1;m<=n;m++){  System.out.print(m+"\*"+n+"="+(m\*n)+"\t");  }  System.out.println();  } |

1. 完成薪水计算器小软件的开发

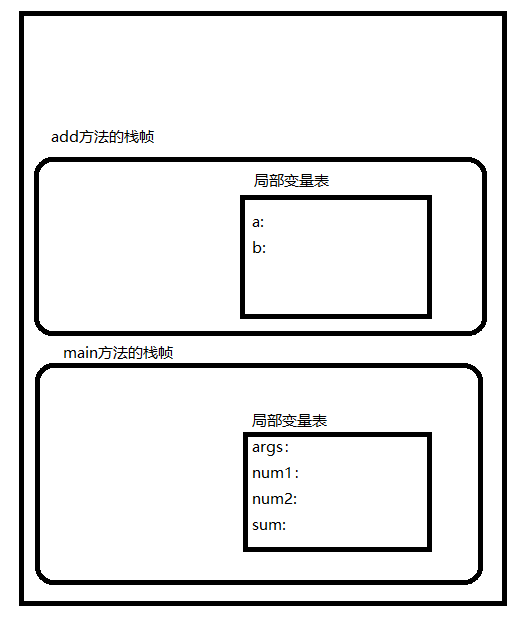
|  |
| --- |
| import java.util.Scanner;  /\*\*  \* 年薪计算小软件  \*/  public class SalarySoft {  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  System.out.println("年薪计算小软件");  while(true) {  System.out.println("请输入月薪：");  int monthSalary = scanner.nextInt();  System.out.println("请输入一年多少个月薪资：");  int months = scanner.nextInt();  int yearSalary = monthSalary\*months;  System.out.println("您的年薪："+yearSalary);  if(yearSalary>=100000&&yearSalary<=200000) {  System.out.println("您表现不错，已经超过90%的国人！");  }  if(yearSalary>200000) {  System.out.println("您是大款，已经超过98%的国人！");  }  System.out.println("请输入命令88[退出],66[继续]");  int commandNum = scanner.nextInt();  if(88==commandNum) {  System.out.println("退出软件");  break;  }  if(66==commandNum) {  System.out.println("请重新输入数据...");  continue;  }  }  }  } |

# 第15集 有了方法万事通：方法的定义\_内存结构

1. 定义一个计算器类，类实现了两个数字的相加、相减、相乘、相除。

|  |
| --- |
| public class Calculator {   public static int add(int a,int b) {  return a+b;  }  public static int minus(int a,int b) {  return a-b;  }   public static int multiple(int a,int b) {  return a\*b;  }   public static int division(int a,int b) {  return a/b;  }   public static void main(String[] args) {  int num1 = 100;  int num2 = 200;  int sum = *add*(num1,num2);   } } |

1. 画出课堂上，调用add()方法时，add()方法对应栈帧的示意图。【选做】



# 第16集 同名方法大比拼：方法的重载

1. 完成课堂上关于重载的代码练习。

|  |
| --- |
| /\*\*  \* 测试方法的重载  \*/  public class TestOverload {  public static void main(String[ ] args) {  add(100.25,200);  add(100,200,300);  }  /\*\* 求和的方法 \*/  public static int add(int n1, int n2) {  int sum = n1 + n2;  return sum;  }  //方法名相同，参数类型不同，构成重载  public static double add(double n1, int n2) {  double sum = n1+n2;  return sum;  }  //方法名相同，参数个数不同，构成重载  public static int add(int n1, int n2,int n3) {  int sum = n1 + n2 + n3;  return sum;  }  //方法名相同，参数顺序不同，构成重载  public static double add(int n1, double n2) {  double sum = n1+n2;  return sum;  }  /\*  //编译错误：只有返回值不同，不构成重载  public static void add(int n1, double n2) {  double sum = n1+n2;  return sum;  }  \*/  /\*  //编译错误：只有参数名称不同，不构成重载  public static double add(int m1, double m2) {  double sum = m1+m2;  return sum;  }  \*/  } |

1. 重载的方法是完全不同的方法，只是名字相同而已。这句话，对吗？

是的。

# 第17集 万事万物皆对象：面向对象编程\_类和对象

1. 按照课堂代码，定义一个学生类，包含：id、name、score、age四个属性。包含：study()方法。

|  |
| --- |
| public class Student {  //静态特征（数据）  int id;  String name;  int score;  int age;  //方法（动态行为）（对数据的操作）  public void study(){  System.out.println("正在学习："+name);  }  } |

1. 按照课堂代码，调用创建好的学生类。创建两个学生对象：
   * + 1. id：1001，name：高淇，score：90，age：18
       2. id：1002，name：高小二、score：80，age：19
       3. 调用其中一个对象的study()方法

|  |
| --- |
| //main方法是程序的入口！  public static void main(String[] args) {  Student stu01 = new Student();  stu01.id = 1001;  stu01.name = "高淇";  stu01.score = 90;  stu01.age = 18;  Student stu02 = new Student();  stu02.id = 1002;  stu02.name = "高晓尔";  stu02.score = 80;  stu02.age = 19;  stu01.study();  stu02.study();  } |

# 第18集 典型对象深掌握：面向对象编程\_一个典型的类定义和调用

1. 完成课堂上的学生类、电脑类的定义，并处理两个类之间的关系。

|  |
| --- |
| public class Computer {  String brand; //品牌  int price; //价格  } |
| public class BaizhanStudent {  int id;  String sname;  int age;  Computer computer;  void study(){  System.out.println("我正在学习！使用的电脑是："+computer.brand);  }  public static void main(String[] args) {  Computer c1 = new Computer();  c1.brand = "联想";  c1.price = 6000;  Computer c2 = new Computer();  c2.brand = "dell";  c2.price = 5000;  BaizhanStudent stu1 = new BaizhanStudent();  stu1.id = 1001;  stu1.sname = "高淇";  stu1.age = 18;  stu1.computer = c1;  stu1.study();  }  } |

# 第19集 毛坯房子装修美：面向对象编程\_构造方法详解

1. 构造方法也是方法，只是一种特殊的方法而已。请写出使用它的4个要点。
   1. 构造器通过new关键字调用。
   2. 构造器不能定义返回值类型(返回值的类型肯定是本类)，不能在构造器里使用return返回某个值。
   3. 如果我们没有定义构造器，则编译器会自动定义一个无参的构造方法。如果已定义则编译器不会自动添加！
   4. 构造器的方法名必须和类名一致！
2. 完成课堂上定义：二维点的练习。

|  |
| --- |
| /\*\*  \* 描述二维空间中的一个点  \*/  public class Point {  double x;  double y;  public Point(double \_x, double \_y){  x = \_x;  y = \_y;  }  public static void main(String[] args) {  Point p1 = new Point(3.0, 4);  System.out.println(p1.x);  }  } |

1. 完成课堂上“User”类，并重载多个构造方法。

|  |
| --- |
| /\*\*  \* 模拟网站的账户（用于测试构造方法的重载）  \*/  public class User {  int id;  String uname; //用户名  String pwd; //密码  User(){  }  public User(int \_id) {  id = \_id;  }  public User(int \_id, String \_uname, String \_pwd ) {  id = \_id;  uname = \_uname;  pwd = \_pwd;  }  public static void main(String[] args) {  User u1 = new User();  User u2 = new User(1001);  User u3 = new User(1002,"高淇","123456");  }  } |

# 第20集 内存图形底层通：图形化分析内存\_栈和堆\_类的加载和对象创建

1. 绘制课堂上TestPerson类执行时的内存变化图【简答和绘图】

参考视频，自行绘制。

1. 分别说明：虚拟机栈、堆、方法区的特点【简答】

**栈的特点如下：**

1. 栈描述的是方法执行的内存模型。每个方法被调用都会创建一个栈帧（存储局部变量、操作数、方法出口等）
2. 栈的存储特性是“先进后出，后进先出”
3. 栈是由系统自动分配，速度快！栈是一个连续的内存空间！

**其他特性（先了解，后面多线程中会深入）：**

1. JVM为每个线程创建一个栈，用于存放该线程执行方法的信息（实际参数、局部变量等）
2. 栈属于线程私有，不能实现线程间的共享！

**堆的特点如下：**

1. 堆用于存储创建好的对象和数组(数组也是对象)
2. 堆是一个不连续的内存空间，分配灵活，速度慢！
3. JVM只有一个堆，被所有线程共享

**方法区（又叫静态区,也是堆）特点如下：**

1. 方法区是JAVA虚拟机规范，可以有不同的实现。
   1. JDK8是“元数据空间”和堆结合起来。
2. 方法区实际也是堆，只是用于存储类、常量相关的信息！
3. 用来存放程序中永远是不变或唯一的内容。（类信息【Class对象，反射机制中会重点讲授】、静态变量、字符串常量等）
4. JVM只有一个方法区，被所有线程共享！

# 第21集 自我本我都是我：图形化分析内存\_this的本质

1. this的本质是什么？【简答】

this的本质就是“当前对象的地址”！普通方法中，this总是指向调用该方法的对象。构造方法中，this总是指向正要初始化的对象。

1. 课堂代码中，写出this在构造方法中的典型用法。【实操】

见作业3

1. 课堂代码中，写出this()调用重载构造方法的案例。【实操】

|  |
| --- |
| /\*\*  \* this的用法测试  \*/  public class User2 {  int id;  String name;  String pwd;  public User2(){  }  public User2(int id,String name) {  System.out.println("正在要初始化的对象："+this);  this.id = id;  this.name = name;  }  public User2(int id,String name,String pwd) {  this(id,name); //调用了重载的构造器！这种调用，必须位于构造方法的第一行！  this.pwd = pwd;  }  public void login(){  System.out.println("要登录："+this.name+",密码："+this.pwd);  }  public static void main(String[] args) {  User2 u = new User2(101,"高晓琪");  u.login();  User2 u3 = new User2(102,"高晓武","123456");  System.out.println(u3.pwd);  System.out.println(u3.name);  }  } |

# 第22集 类的下属都有谁：图形化分析内存\_static的本质

1. static修饰的属性和方法，从属于类。普通属性从属于对象。这句话对吗？

对。

1. 完成课堂代码，并画内存图或文字说明，说明“static方法中不能使用this的原因”。

|  |
| --- |
| /\*\*  \* 测试static的用法  \*/  public class TestStatic {  int id;  static String company = "北京尚学堂";  static int companyId = 0;  static {  System.out.println("类被初始化的调用！");  companyId = 111222;  }  static void printCompany(){  System.out.println(company);  // System.out.println(this); //static方法中不能使用非static成员！  }  void login(){  System.out.println("登陆");  }  public static void main(String[] args) {  TestStatic.printCompany();  TestStatic t1 = new TestStatic();  t1.login();  TestStatic t2 = new TestStatic();  t2.login();  }  } |

内存图，请参考视频自行绘制出。或者对视频截图。

# 第23集 百类千类用包管：面向对象编程\_package和import

1. package在java中的作用类似于“操作系统中文件夹的作用”。主要解决哪两个问题？

主要解决两个问题：多个类的管理、类的重名问题。

1. 完成课堂上导入“同名类”的处理代码，并说明如何处理？

com.itbaizhan.test2.Car car = new com.itbaizhan.test2.Car();

遇到同名类。类名前面增加包名区分即可。

1. 哪一个包中的代码，我们是不需要导入可以直接使用的？

java.lang核心包

# 第24集 父爱一切给孩子：面向对象编程\_继承\_方法重写

1. 完成课堂上关于继承的代码

|  |
| --- |
| package com.itbaizhan.inherit;  import java.util.Objects;  /\*\*  \* 测试继承的基本用法  \*/  public class TestExtends {  public static void main(String[] args) {  Student stu1 = new Student("高淇",172,"java");  stu1.rest();  stu1.study();  System.out.println(stu1.name);  //instanceof判断对象是否是这个类的实例对象  System.out.println(stu1 instanceof Student);  System.out.println(stu1 instanceof Person);  }  }  class Person {  String name;  int height;  public void rest(){  System.out.println("休息一会！");  }  }  class Student extends Person {  String major;  public void study(){  System.out.println("Student.study");  }  public Student(String name,int height,String major){  this.name = name;  this.height = height;  this.major = major;  }  } |

1. 完成课堂上关于方法重写的代码

|  |
| --- |
| package com.itbaizhan.inherit;  /\*\*  \* 测试重写  \*/  public class TestOverride {  public static void main(String[] args) {  Horse h1 = new Horse();  h1.run();  Plane p = new Plane();  p.stop();  }  }  class Vehicle{  //final修饰方法。则这个方法不能在子类中重写！  public final void addOil(){  System.out.println("oil!!!");  }  public void run(){  System.out.println("run....");  }  public void stop(){  System.out.println("stop...");  }  }  class Horse extends Vehicle {  public void run(){  System.out.println("四踢翻飞，嘚嘚...");  }  }  class Plane extends Vehicle {  public void run(){  System.out.println("天上飞..");  }  public void stop(){  System.out.println("在机场停下，不能再天上停");  }  } |

1. final修饰类、修饰变量、修饰方法分别代表什么意思？

final修饰变量：常量

final修饰方法：不可重写

final修饰类： 不能被继承

# 第25集 组合继承谁更好：面向对象编程\_继承和组合

1. 组合的核心实现原理是什么？

组合”的核心就是“将父类对象作为子类的属性”，然后，“子类通过调用这个属性来获得父类的属性和方法”。

1. is-a, has-a这两种关系，哪种使用继承？哪种使用组合？

is-a关系用继承；

has-a关系用组合；

1. 完成课堂上关于组合的测试代码

|  |
| --- |
| package com.itbaizhan.inherit;  /\*\*  \* 测试组合  \*/  public class TestComposition {  public static void main(String[] args) {  Stu stu = new Stu("高淇",172,"java");  stu.person.rest();  System.out.println(stu.person.name);  System.out.println(stu.person.height);  System.out.println(stu.major);  }  }  class Stu {  Person person = new Person();  String major;  public void study(){  System.out.println("Stu.study");  }  public Stu(String name,int height,String major){  this.person.name = name;  this.person.height = height;  this.major = major;  }  }  class CPU {  void calculate(){  System.out.println("CPU.calculate");  }  }  class Memory {  void store(){  System.out.println("Memory.store");  }  }  class Computer {  CPU cpu = new CPU();  Memory memory = new Memory();  } |

# 第26集 万物同宗道生一：面向对象编程\_Object类\_重写toString等方法

1. 学会查看Object类源码（注意：暂不要求看懂）

鼠标放到类上，ctrl+左键即可查看源码

1. 完成课堂上，重写toString的代码

见作业3

1. 完成课堂上，重写equals的代码

|  |
| --- |
| import java.util.Objects;  /\*\*  \* 测试Object类的用法  \*/  public class TestObject {  public static void main(String[] args) {  Employee e1 = new Employee(1001,"张三");  Employee e2 = new Employee(1001,"张三");  System.out.println(e1); //打印对象默认是调用的toString()  System.out.println(e1==e2); //两个对象是否相同！  System.out.println(e1.equals(e2)); //两个对象是否相等（逻辑上进行某些值的比较）  }  }  class Employee extends Object {  int id;  String name;  public Employee(int id, String name) {  this.id = id;  this.name = name;  }  @Override  public String toString() {  return "雇员编号："+id+"，姓名："+name;  }  @Override  public boolean equals(Object o) {  if (this == o) return true;  if (o == null || getClass() != o.getClass()) return false;  Employee employee = (Employee) o;  return id == employee.id;  }  @Override  public int hashCode() {  return Objects.hash(id);  }  } |

# 第27集 向上追根好溯源：面向对象编程\_super和继承内存分析

1. 创建子类对象的过程中，实际只创建了一个子类对象并没有创建父类对象。 这句话，对吗？

是的。

1. super就是父类对象的引用。这句话，对吗？

为了方便理解super可以看做父类对象的引用。但是，创建子类对象过程并没有产生父类对象，super只是代表了父类的特征。

1. 完成课堂上关于super的测试代码。

|  |
| --- |
| public class TestSuper {  public static void main(String[] args) {  Child c = new Child();  c.show();  }  }  class Parent {  int num = 300;  public void show(){  System.out.println("父类中，show()");  }  public Parent(){  // super();  System.out.println("初始化父类对象！");  }  }  class Child extends Parent{  int num = 1000;  public Child(){  super();  System.out.println("初始化子类对象！");  }  @Override  public void show() {  System.out.println("子类中，show()");  super.show();  System.out.println("子类中，num："+num);  System.out.println("父类的num："+super.num);  }  } |

# 第28集 内聚低耦是封装：面向对象编程\_封装

1. 将下表内容填出来：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **访问权限修饰符** | | | | |
| **修饰符** | **同一个类** | **同一个包中** | **子类** | **所有类** |
| private | **√** |  |  |  |
| default | **√** | **√** |  |  |
| protected | **√** | **√** | **√** |  |
| public | **√** | **√** | **√** | **√** |

1. 完成课堂上的封装实现的测试代码。

由于有多个类。大家直接看课堂源代码即可。

# 第29集 一句多义即多态：面向对象编程\_多态\_对象的转型（向上、向下）

1. 一句话说：多态指的是什么？

**多态指的是同一个方法调用，由于对象不同可能会有不同的行为。**

1. 多态的三个必要条件是什么？

继承，方法重写，父类引用指向子类对象

1. 完成课堂中，多态的代码案例。

|  |
| --- |
| package com.itbaizhan.polymorphism;  public class Animal {  public void shout(){  System.out.println("Animal.shout");  }  }  class Dog extends Animal {  public void shout(){  System.out.println("汪汪汪！");  }  public void watchDoor(){  System.out.println("Dog.watchDoor");  }  }  class Cat extends Animal {  public void shout() {  System.out.println("喵喵喵！");  }  } |
| package com.itbaizhan.polymorphism;  public class TestPolym {  public static void main(String[] args) {  Dog d = new Dog();  animalCry(d);  Cat c = new Cat();  animalCry(c);  //对象的转型  Animal a = new Dog();  a.shout();  // a.watchDoor();  //强制转型  Dog a2 = (Dog)a;  a2.watchDoor();  // Cat a3 = (Cat)a; //运行时报错：java.lang.ClassCastException  }  static void animalCry(Animal a){  System.out.println("step1");  a.shout(); //多态的三个必要条件：继承、方法的重写、父类引用指向子类对象  System.out.println("step2");  }  } |

1. 完成课堂中，对象转型的代码案例。

见作业3.

# 第30集 共同特征是抽象：面向对象编程\_抽象类和抽象方法

1. 抽象方法是什么？如何定义？

使用abstract修饰的方法，没有方法体，只有声明。

2. 包含抽象方法的类一定是抽象类。对吗？

是的。

1. 完成课堂上，抽象方法和抽象类的测试代码。

|  |
| --- |
| package com.itbaizhan.abstractClass;  //抽象类  public abstract class Animal {  //抽象方法  public abstract void shout();  public void sleep(){  System.out.println("Animal.sleep");  }  public static void main(String[] args) {  Dog d = new Dog();  d.shout();  // Animal a = new Animal();  }  }  class Dog extends Animal {  @Override  public void shout() {  System.out.println("Dog.shout");  }  }  class Cat extends Animal {  @Override  public void shout() {  System.out.println("Cat.shout");  }  } |

1. 完成课堂上，模板方法模式的测试案例。

|  |
| --- |
| package com.itbaizhan.abstractClass;  //模板方法模式（抽象类的应用）  public abstract class DBOperator {  //1. 建立连接 2. 打开数据库 3. 使用数据库 4. 关闭连接  public abstract void connection();  public void open(){  System.out.println("打开数据库");  }  public void use(){  System.out.println("使用数据库");  }  public void close(){  System.out.println("关闭连接");  }  public void process(){  connection();  open();  use();  close();  }  public static void main(String[] args) {  // new MySqlOperator().process();  new OracleOperator().process();  }  }  class MySqlOperator extends DBOperator {  @Override  public void connection() {  System.out.println("建立和Mysql数据库的连接");  }  }  class OracleOperator extends DBOperator {  @Override  public void connection() {  System.out.println("建立和Oracle数据库的连接");  }  } |

# 第31集 最最抽象是接口：面向对象编程\_接口

1. 普通类、抽象类、接口的区别是什么？

普通类：只有具体实现

抽象类：有具体实现、有抽象方法

接口： 只有抽象方法（JDK8以前）

1. 完成课堂代码，接口的定义和使用。

|  |
| --- |
| /\*\*  \* 测试接口的语法，基本用法  \*/  public class TestInterface {  public static void main(String[] args) {  Volant a = new Angel();  a.fly();  Honest h = new Angel();  h.helpOthers();  }  }  /\*\*  \* 飞行接口  \*/  interface Volant {  int FLY\_HEIGHT = 100; //public static final  void fly(); //public abstract void fly();  }  /\*\*  \* 善良接口  \*/  interface Honest {  void helpOthers();  }  class Angel implements Volant,Honest {  @Override  public void fly() {  System.out.println("我是天使，飞呀飞呀！");  }  @Override  public void helpOthers() {  System.out.println("我是天使，喜欢扶老奶奶过马路！");  }  }  class BirdMan implements Volant {  @Override  public void fly() {  System.out.println("我是鸟人！像鸟一样飞！");  }  } |

1. 完成课堂代码，接口的多继承。

|  |
| --- |
| /\*\*  \* 测试接口的多继承  \*/  public class TestInterface2 implements C {  @Override  public void testa() {  System.out.println("TestInterface2.testa");  }  @Override  public void testb() {  System.out.println("TestInterface2.testb");  }  @Override  public void testc() {  System.out.println("TestInterface2.testc");  }  public static void main(String[] args) {  C c = new TestInterface2();  c.testa();  c.testb();  c.testc();  }  }  interface A {  void testa();  }  interface B{  void testb();  }  interface C extends A,B {  void testc();  } |

# 第32集 抽象具体混起来：面向对象编程\_接口的新特性

1. 完成课堂上，默认方法的测试代码
2. 完成课堂上，静态方法的测试代码

|  |
| --- |
| /\*\*  \* 测试接口的：默认方法、静态方法  \*/  public class TestInterface3 {  public static void main(String[] args) {  TestA ta = new TestA();  ta.moren1();  A1.staticMethod(); //通过接口名称直接调用静态方法  TestA.staticMethod(); //调用实现类的静态方法  TestB tb = new TestB();  tb.moren1();  }  }  interface A1 {  default void moren1(){  System.out.println("A1.moren1");  }  default void moren2(){  System.out.println("A1.moren2");  }  static void staticMethod(){  System.out.println("A1.staticMethod");  }  }  interface B1 {  default void moren1(){  System.out.println("B1.moren1");  }  }  class TestA implements A1 {  @Override  public void moren1() {  System.out.println("TestA.moren1");  }  static void staticMethod(){  System.out.println("TestA.staticMethod");  }  }  class TestB implements A1,B1 {  @Override  public void moren1() {  System.out.println("TestB.moren1");  }  } |

# 第33集 内部类像画中画：面向对象编程\_内部类

1. 完成课堂上，非静态内部类的定义和调用代码。

|  |
| --- |
| public class Outer1 {  private int age = 18;  public void show(){  System.out.println("外部类，age："+age);  }  //内部类。定义在外部类Outer1里面  //非静态内部类不能有静态的方法、静态的属性、静态初始化块  class Inner1 {  private int id = 10001;  private int age = 28;  public void test1(){  System.out.println("Inner1.test1");  System.out.println("内部类的属性，age："+this.age);  System.out.println("外部类的属性，age："+Outer1.this.age); //通过：外部类名.this.属性名，访问外部类的属性  Outer1.this.show();  }  }  } |

1. 完成课堂上，匿名内部类的代码。

|  |
| --- |
| /\*\*  \* 测试匿名内部类  \*/  public class TestAnonymousInnerClass {  public void test(A3 a){  System.out.println("TestAnonymousInnerClass.test");  a.run();  }  public static void main(String[] args) {  TestAnonymousInnerClass tc = new TestAnonymousInnerClass();  //匿名内部类只使用一次！！  tc.test(new A3() {  @Override  public void run() {  System.out.println("匿名内部类的run()");  }  });  tc.test(new A3() {  @Override  public void run() {  System.out.println("TestAnonymousInnerClass.run");  }  });  }  }  interface A3 {  void run();  } |

# 第34集 同类信息轻松存：数组和算法\_一维数组

1. 将如下这些数字：123,213,453,658保存到数组中，并分别使用三种初始化方式进行初始化。

|  |
| --- |
| *//静态初始化* int[] a = {123,213,453,658};  int[] b = new int[4]; System.*out*.println("默认初始化都是0："+b[0]); *//动态初始化* b[0] = 123; b[1] = 213; b[2] = 453; b[3] = 658; |

1. 完成课堂上，引用类型User放入数组中的测试代码。

|  |
| --- |
| /\*\*  \* 测试数组的使用  \*/  public class Test02 {  public static void main(String[] args) {  User[] users = new User[3];  //静态初始化  User user2[] = {  new User(101,"tome"),  new User(102,"smith"),  new User(101,"lily")  };  users[0] = new User(1001,"张三");  users[1] = new User(1002,"李四");  users[2] = new User(1003,"王五");  for(int i=0;i<users.length;i++) {  System.out.println(users[i]);  }  for(User u:user2) {  System.out.println(u);  }  }  }  class User {  private int id;  private String name;  public User(int id, String name) {  this.id = id;  this.name = name;  }  @Override  public String toString() {  return "User{" +  "id=" + id +  ", name='" + name + '\'' +  '}';  }  } |

1. 代码实现：定义一个数组[200,100,90,65,156]，用普通遍历和for-each两种方式遍历打印数组中大于100的元素。

|  |
| --- |
| int[] a = {200,100,90,65,156};  *//普通遍历* for(int i=0;i<a.length;i++) {  if(a[i]>100) {  System.*out*.println(a[i]);  } }  *//foreach遍历* for(int temp:a) {  if(a[i]>100) {  System.*out*.println(a[i]);  } } |

# 第35集 多维数组碟中谍：数组和算法\_多维数组

1. 完成课堂上，数组存储表格数据的练习。

|  |
| --- |
| import java.util.Arrays;  /\*\*  \* 测试二维数组：利用Object数组存储表格信息  \*/  public class Test06 {  public static void main(String[] args) {  Object[] a1 = {1001,"高淇",18,"讲师","2021-4-15"}; //包装类  Object[] a2 = {1002,"高小七",19,"助教","2021-4-10"};  Object[] a3 = {1003,"高小八",20,"班主任","2021-3-10"};  Object[][] emps = new Object[3][];  emps[0] = a1;  emps[1] = a2;  emps[2] = a3;  for(int i=0;i<emps.length;i++){  System.out.println(Arrays.toString(emps[i]));  }  Emp e1 = new Emp(1001,"高淇",18,"讲师","2021-4-15");  Emp e2 = new Emp(1002,"高小七",19,"助教","2021-4-10");  Emp e3 = new Emp(1003,"高小八",20,"班主任","2021-3-10");  Emp[] employees = {e1,e2,e3};  for(Emp temp:employees) {  System.out.println(temp);  }  }  }  class Emp {  private int id;  private String name;  private int age;  private String job;  private String hiredate;  @Override  public String toString() {  return "Emp{" +  "id=" + id +  ", name='" + name + '\'' +  ", age=" + age +  ", job='" + job + '\'' +  ", hiredate='" + hiredate + '\'' +  '}';  }  public Emp(int id, String name, int age, String job, String hiredate) {  this.id = id;  this.name = name;  this.age = age;  this.job = job;  this.hiredate = hiredate;  }  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public int getAge() {  return age;  }  public void setAge(int age) {  this.age = age;  }  public String getJob() {  return job;  }  public void setJob(String job) {  this.job = job;  }  public String getHiredate() {  return hiredate;  }  public void setHiredate(String hiredate) {  this.hiredate = hiredate;  }  } |

1. 提高作业。如下表是一张商品信息表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | 名称 | 型号 | 价格 | 折扣 |
| 1 | 百战牌鼠标 | BZ\_001 | 99.21 | 0.9 |
| 2 | 键盘侠玩偶 | WO\_102 | 403.00 | 0.7 |
| 3 | 实战java程序设计 | BK\_001 | 89.00 | 0.8 |
| 4 | 高淇牌西装 | GQ\_XF\_12 | 700.00 | 0.5 |
| 5 | 大米牌手机 | DM\_PH\_13 | 900.00 | 0.3 |

完成功能：

1. 定义一个类表示上表中的商品。
2. 创建5个对象存储上面表的信息。
3. 创建一个数组，存储这5个对象。
4. 再创建一个方法，遍历（toString()）这个数组，打印表的信息。
5. 创建一个方法：查询最终购买价，大于指定数字的所有商品。

|  |
| --- |
| package array;  public class Test {   public static void main(String[] args) {  GoodsInfo goods0 = new GoodsInfo(1,"百战牌鼠标","BZ\_001",99.21,0.9);  GoodsInfo goods1 = new GoodsInfo(2,"键盘侠玩偶","WO\_102",403.00,0.7);  GoodsInfo goods2 = new GoodsInfo(3,"实战java程序设计","BK\_001",89.00,0.8);  GoodsInfo goods3 = new GoodsInfo(4,"高淇牌西装","GQ\_XF\_12",700.00,0.5);  GoodsInfo goods4 = new GoodsInfo(5,"大米牌手机","DM\_PH\_13",900.00,0.3);   GoodsInfo[] infos = new GoodsInfo[5];  infos[0]=goods0;  infos[1]=goods1;  infos[2]=goods2;  infos[3]=goods3;  infos[4]=goods4;   *printInfos*(infos);  *selectByPrice*(infos,100);   }   public static void selectByPrice(GoodsInfo[] infos,double price) {  System.*out*.println("=====商品查询（按价格）=======");  System.*out*.println("最终购买价大于"+price+"的商品如下：");  for(GoodsInfo info:infos) {  double resultPrice = info.getPrice()\*info.getDiscount();  if(resultPrice>price){  System.*out*.println(info.getName()+",最终购买价："+resultPrice);  }   }  }  public static void printInfos(GoodsInfo[] infos){  System.*out*.println("ID\t名称\t\t\t\t型号\t\t价格\t\t折扣");  for(GoodsInfo gi:infos) {  System.*out*.println(gi.getId()+"\t"+  gi.getName()+"\t\t"+  gi.getType()+"\t"+  gi.getPrice()+"\t"+  gi.getDiscount());  }  }  }  *//商品信息* class GoodsInfo {  private int id; *//商品编号* private String name; *//商品名称* private String type; *//类型* private double price; *//价格* private double discount; *//折扣* public GoodsInfo(int id, String name, String type, double price, double discount) {  this.id = id;  this.name = name;  this.type = type;  this.price = price;  this.discount = discount;  }   public int getId() {  return id;  }   public void setId(int id) {  this.id = id;  }   public String getName() {  return name;  }   public void setName(String name) {  this.name = name;  }   public String getType() {  return type;  }   public void setType(String type) {  this.type = type;  }   public double getPrice() {  return price;  }   public void setPrice(double price) {  this.price = price;  }   public double getDiscount() {  return discount;  }   public void setDiscount(double discount) {  this.discount = discount;  } } |

# 第36集 排序算法更优化：数组和算法\_冒泡算法和优化

1. 完成冒泡排序的代码实现
2. 完成课堂上，冒泡排序的优化代码。

|  |
| --- |
| import java.util.Arrays;  /\*\*  \* 测试冒泡排序  \*/  public class Test07 {  public static void main(String[] args) {  int[] values = {3,1,6,2,9,0,7,4,8,5};  bubbleSort(values);  }  public static void bubbleSort(int[] values){  //int[] values = {3,1,6,2,9,0,7,4,8,5};  System.out.println("排序前,数组:"+ Arrays.toString(values));  int temp; //用于交换的临时变量  //定义一个布尔类型变量，标记数组是否已达到有序状态  boolean flag = true;  for(int i=0;i<values.length;i++){  for(int j=0; j<values.length-i-1;j++){  if(values[j]>values[j+1]){  //两两比较。如果大于，则互换元素  temp = values[j];  values[j] = values[j+1];  values[j+1] = temp;  //本趟发生了交换，表明该数组在本趟处于无序状态，需要继续排序。  flag = false;  }  System.out.println("第"+i+"趟:第"+j+"次:"+Arrays.toString(values));  }  //冒泡排序的优化  if(flag){  System.out.println("结束排序！");  break;  }else{  flag = true; //重置回原值！  }  System.out.println("第"+i+"趟,完成时:"+Arrays.toString(values));  }  }  } |

# 第37集 二分查找快如电：数组和算法\_二分法查找（折半查找）

1. 二分查找只适用：排好序的数组。 这句话，对吗？

正确。

1. 完成课堂上二分法查找的代码实现

|  |
| --- |
| import java.util.Arrays;  /\*\*  \* 测试二分法查找元素  \*/  public class Test08 {  public static void main(String[] args) {  int[] arr = {30,20,50,10,80,9,7,12,100,40,8};  int searchWord = 80; //所要查找的值  System.out.println(searchWord+"索引位置:"+binarySearch(arr,searchWord));  }  /\*\*  \* 二分法查找  \* @param array 目标数组  \* @param value 所要查找的值  \* @return 如果查找到，则返回这个值在数组中索引；如果未找到，则返回-1  \*/  public static int binarySearch(int[] array, int value){  //二分法查找的数组，必须先做排序  Arrays.sort(array);  System.out.println(Arrays.toString(array));  int low = 0;  int high = array.length-1;  while(low<=high){  int middle = (low+high)/2;  if(value==array[middle]){  return middle; //查询到这个值，返回对应的索引位置！  }  if(value>array[middle]) {  low = middle+1;  }  if(value<array[middle]){  high = middle-1;  }  }  return -1; //上面循环完毕，说明未找到，返回-1  }  } |

# 第38集 不可变的字符串：常用类和JDK源码解读\_String类

1. String类代表字符串，是“不可变字符序列”。 √

2. 分别写出下面代码的打印结果：

String str1 = "abc";

String str2 = new String("abc");

String str3 = "abc";

System.out.println(str1 == str3);

System.out.println(str2 == str3);

System.out.println(str2.equals(str3));

答案如下：

true

false

true

3. 完成课堂代码中关于String类常用方法的测试（至少两遍）

|  |
| --- |
| import java.util.Arrays;  /\*\*  \* 测试String类的基本用法  \*/  public class TestString {  public static void main(String[] args) {  test3();  }  public static void test1(){  //String类的定义  String s1 = "abc"; //凡是字符串常量，都会放到字符串的常量池中  String s2 = new String("abc"); //建了一个新的字符串对象  String s3 = "abc";  String s4 = "aBC";  //比较字符串  System.out.println(s1==s2); //false  System.out.println(s1==s3); //true  System.out.println(s1.equals(s2)); //字符串的值是否相等  System.out.println(s1.equalsIgnoreCase(s4)); //忽略大小写  }  public static void test2(){  String s1 = "0123456789,How are you, How old are you";  System.out.println(s1.charAt(4)); //索引位置4的字符  System.out.println(s1.length());  //字符串转化成数组  char[] chars = s1.toCharArray();  System.out.println(Arrays.toString(chars));  String[] strs = s1.split(",") ;//可以传入正则表达式  System.out.println(Arrays.toString(strs));  //判断是否包含子字符串  System.out.println(s1.indexOf("are"));  System.out.println(s1.lastIndexOf("are"));  System.out.println(s1.contains("How"));  System.out.println(s1.startsWith("0123"));  System.out.println(s1.endsWith("you"));  }  public static void test3(){  String s1 = "0123456789,How are you";  String s2 = s1.replace(' ','&');  System.out.println(s1);  System.out.println(s2);  s2 = s1.substring(4);  System.out.println(s2);  s2 = s1.substring(4,10); //[4,10) 不包含10这个位置  System.out.println(s2);  s2 = s1.toLowerCase();  System.out.println(s2);  s2 = s1.toUpperCase();  System.out.println(s2);  //去除首尾空格  String s3 = " How are you ";  s2 = s3.trim();  System.out.println(s3);  System.out.println(s2);  }  } |

# 第39集 可变随改字符串：常用类\_StringBuilder和StringBuffer

1. 阅读源码，String类、StringBuilder、StringBuffter存储字符时，分别依赖的是什么属性？

答：字符数组

1. 完成课堂上，StringBuilder的测试代码。

|  |
| --- |
| public static void test1(){  String s = "";  StringBuilder sb = new StringBuilder();  StringBuffer sb2 = new StringBuffer();  sb.append("a");  sb.append("b");  sb.append("c").append("d").append("e");  System.out.println(sb);  sb2.append("北京尚学堂");  sb2.insert(0,"爱").insert(0,"我");  System.out.println(sb2);  sb2.delete(0,2); //[0,2) 包头不包尾  System.out.println(sb2);  sb2.deleteCharAt(0).deleteCharAt(0);  System.out.println(sb2);  System.out.println(sb2.reverse()); //字符串逆序  } |

1. 完成课堂上，String和StringBuilder在字符串频繁修改时的效率测试。

|  |
| --- |
| public static void test2(){  //使用String进行字符串的拼接  String str = "";  long num1 = Runtime.getRuntime().freeMemory(); //获取JVM剩余的内存空间，单位是字节  long time1 = System.currentTimeMillis(); //获取当前时间,单位是毫秒  for(int i=0; i<5000;i++){  str += i; //相当于产生5000个String对象  }  long num2 = Runtime.getRuntime().freeMemory();  long time2 = System.currentTimeMillis();  System.out.println("String占用内存："+(num1-num2));  System.out.println("String占用时间："+(time2-time1));  System.out.println("使用可变字符序列，完成拼接=========");  StringBuilder sb = new StringBuilder();  long num3 = Runtime.getRuntime().freeMemory(); //获取JVM剩余的内存空间，单位是字节  long time3 = System.currentTimeMillis(); //获取当前时间,单位是毫秒  for(int i=0; i<5000;i++){  sb.append(i);  }  long num4 = Runtime.getRuntime().freeMemory();  long time4 = System.currentTimeMillis();  System.out.println("StringBuilder占用内存："+(num3-num4));  System.out.println("StringBuilder占用时间："+(time4-time3));  } |

1. 说出如下代码，产生了多少个String对象：

|  |
| --- |
| String a = **"gao"**; **for**(**int** i=0;i<1000;i++){  a += i; } |

答：一共产生1001个String对象。

# 第40集 数字对象轻松转：常用类\_包装类

1. 写出八种基本数据类型对应的包装类的名字。

|  |  |
| --- | --- |
| **基本数据类型** | **包装类** |
| byte | Byte |
| boolean | Boolean |
| short | Short |
| **char** | **Character** |
| **int** | **Integer** |
| long | Long |
| float | Float |
| double | Double |

1. 课堂代码，完成“包装类的使用”测试代码。

|  |
| --- |
| /\*\*  \* Integer的基本用法  \*/  public static void testInteger(){  //基本数据类型转化成Integer对象  Integer int1 = Integer.valueOf(100);  //包装类对象转成基本数据类型  int int2 = int1.intValue();  long long1 = int1.longValue();  //字符串转成Integer对象  Integer int3 = Integer.parseInt("324");  System.out.println(int3);  System.out.println(int3.toString()); //包装类对象转成字符串  System.out.println("int能表示的最大整数："+Integer.MAX\_VALUE);  } |

1. 自动装箱和拆箱指的是什么？写出测试代码。

自动装箱和拆箱就是将基本数据类型和包装类之间进行自动的互相转换。JDK1.5后，Java引入了自动装箱(autoboxing)/拆箱(unboxing)。

自动装箱与拆箱的功能事实上是编译器来帮的忙，编译器在编译时依据您所编写的语法，决定是否进行装箱或拆箱动作。

|  |
| --- |
| public static void testAutoBox(){  Integer a = 100; //自动装箱。编译器添加：Integer a = Integer.valueOf(100);  int b = a; //自动拆箱。编译器添加：int b = a.intValue();  //空指针异常  Integer c = null;  // int d = c; //自动拆箱。编译器添加:int d = c.intValue();  } |

1. Integer类缓存处理的范围是什么？请打开源代码寻找，确认范围。

整型、char类型所对应的包装类，在自动装箱时，对于-128~127之间的值会进行缓存处理，其目的是提高效率。

5.【提高作业】自定义一个简单的包装类MyIneger（模仿老师课堂代码）。

|  |
| --- |
| /\*\*  \* 自定义一个简单的包装类。仅用于学习和娱乐！  \*/  public class MyInteger {  private int value;  private static MyInteger[] cache = new MyInteger[256];  public static final int LOW = -128;  public static final int HIGH = 127;  //静态初始化块：是在类被加载的时候，初始化类的静态属性  static {  for(int i=MyInteger.LOW;i<=HIGH;i++){  //-128,0;-127,1;-126,2;  cache[i+(-LOW)] = new MyInteger(i);  }  }  public static MyInteger valueOf(int i){  //如果在-128到127之间，则返回数组中缓存的对象；否则创建新的对象  if(i>=LOW&&i<=HIGH) {  return cache[i+(-LOW)];  }else{  return new MyInteger(i);  }  }  public int intValue(){  return value;  }  @Override  public String toString() {  return value+"";  }  private MyInteger(int i) {  this.value = i;  }  public static void main(String[] args) {  MyInteger a = MyInteger.valueOf(100);  MyInteger b = MyInteger.valueOf(100);  System.out.println(a==b);  MyInteger c = MyInteger.valueOf(400);  MyInteger d = MyInteger.valueOf(400);  System.out.println(c==d);  }  } |

# 第41集 时间转化字符串：常用类\_时间转化字符串

1. 课堂代码，完成课堂上Date类的测试代码。

|  |
| --- |
| /\*\*  \* 测试Date类  \*/  public static void test1(){  long nowNum = System.currentTimeMillis(); //返回当前时刻的毫秒数  System.out.println(nowNum);  Date d1 = new Date();  System.out.println(d1.getTime());  Date d2 = new Date(1000L\*3600\*24\*365\*150); //距离1970年150年后的日期。忽略闰月。  System.out.println(d2);  Date d3 = new Date();  d3.getDate(); //方法被废弃。可以用，但是不建议使用！  } |

1. DateFormat的作用是什么？

完成字符串和Date对象的转换。

1. DateFormat和SimpleDateFormat的关系是什么？

DateFormat是抽象类。

SimpleDateFormat是DateFormat的子类。

1. 课堂代码，完成SimpleDateFormat的测试代码。

|  |
| --- |
| /\*\*  \* 测试DateFormat类的使用：时间和字符串之间互相转换  \*/  public static void test2() throws ParseException {  DateFormat df = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");  Date d2 = new Date(1000L\*3600\*24\*365\*150);  String str2 = df.format(d2);  System.out.println(str2);  String str3 = "2049-10-1 10:10:20";  Date d3 = df.parse(str3);  System.out.println(d3.getTime());  DateFormat df2 = new SimpleDateFormat("yyyy年MM月dd日");  //利用格式化字符可以方便的做一些其他的事：获取当前时间是今年的第几天  DateFormat df3 = new SimpleDateFormat("D");  Date date3 = new Date();  System.out.println(df3.format(date3));  } |

1. Calendar类的作用是什么？

Calendar用于日期计算。

1. 课堂代码，完成Canlendar类的测试代码。

|  |
| --- |
| /\*\*  \* 测试日期类Calendar  \*/  public static void test3(){  //月份：0-11。 0:1月，1:2月，...，11:12月  //星期：1-7. 1:周日，2：周一，... 7:周六  Calendar calendar = new GregorianCalendar(2999,9,10,11,25,40);  int year = calendar.get(Calendar.YEAR);  int month = calendar.get(Calendar.MONTH);  System.out.println(year);  System.out.println(month);  calendar.set(Calendar.YEAR,2049);  System.out.println(calendar.get(Calendar.YEAR));  System.out.println(calendar.getTime()); //返回对应的Date对象  System.out.println(calendar.getTimeInMillis());//返回对应的毫秒数  //日期的计算  calendar.add(Calendar.DATE,1000); //往后1000天  System.out.println(calendar.getTime());  calendar.add(Calendar.YEAR,-30); //往前30年  System.out.println(calendar.getTime());  } |

# 第42集 异常到底是什么：异常机制1

1. 说出如下异常的含义：

ArithmeticException：数学异常

NullPointerException：空指针异常

ClassCastException：类型转换异常

ArrayIndexOutOfBoundsException：数组下标越界异常

NumberFormatException：数字格式化异常

2. 完成课堂代码

|  |
| --- |
| package com.itbaizhan;  import java.util.regex.Matcher;  import java.util.regex.Pattern;  public class Test01 {  public static void main(String[] args) {  test3();  }  public static void test1(){  System.out.println("111");  int a = 1/0;  System.out.println("222");  }  public static void test2(){  System.out.println("111");  try {  int a = 1/0; //ctrl+alt+t快捷键：增加try,catch  } catch (Exception e) {  e.printStackTrace();  }  System.out.println("222");  }  /\*\*  \* RunTimeException需要程序员做逻辑处理  \*/  public static void test3(){  //ArithmeticException:by Zero  int a = 1;  int b = 0;  // System.out.println(a/b);  if(b!=0) {  System.out.println(a/b);  }  //空指针异常：NullPointerException  String str = null;  if(str!=null) {  System.out.println(str.charAt(2));  }  //数字格式化异常:NumberFormatException  String str2 = "234abc";  // int c = Integer.parseInt(str);  // System.out.println(c);  Pattern p = Pattern.compile("^\\d+$");  Matcher m = p.matcher(str2);  if(m.matches()){  System.out.println(Integer.parseInt(str2));  }else{  System.out.println("数字格式不对！");  }  //数组下标越界异常:ArrayIndexOutOfBoundsException  int[] arrs = new int[5];  int f = 5;  if(f>=0&&f<arrs.length) {  System.out.println(arrs[f]); //[0,length-1]  }  //类型转化异常：ClassCastException  Animal am = new Dog();  // Cat cat = (Cat) am;  if(am instanceof Cat) {  Cat cat = (Cat) am;  }  }  }  class Animal {}  class Dog extends Animal {}  class Cat extends Animal {} |

# 第43集 抓住异常搞定它：异常机制2

1. CheckException和RuntimeException的区别是什么？

CheckedException：编译期处理的异常

RuntimeException：运行时异常

1. 完成课堂上，try-catch-finally的测试代码。

|  |
| --- |
| //测试try-catch-finally  public static void test1(){  FileReader reader = null;  try {  reader = new FileReader("d:/eeee.txt");  char c1 = (char)reader.read();  char c2 = (char)reader.read();  System.out.println("文件的内容："+c1+c2);  } catch (FileNotFoundException e) {  e.printStackTrace();  } catch (IOException e) {  e.printStackTrace();  }finally {  if(reader!=null) {  try {  reader.close();  } catch (IOException e) {  e.printStackTrace();  }  }  }  } |

1. 完成课堂上，try-witch-resouce的测试代码。

|  |
| --- |
| /\*\*  \* "try-with-resource":可以自动关闭实现了Closable接口的类。  \* 将try-catch-finally:try-catch  \* 其实一种语法糖。编程器帮我们做了处理，转化成了：try-catch-finally  \*/  public static void test3(){  try(FileReader reader = new FileReader("d:/a.txt")){  char c1 = (char)reader.read();  char c2 = (char)reader.read();  System.out.println("文件的内容："+c1+c2);  }catch (Exception e) {  e.printStackTrace();  }  } |

# 第44集 看见意味被消灭：可视化bug追踪

1. 完成课堂上关于如何百度解决异常的案例。

|  |
| --- |
| import java.util.Scanner;  /\*\*  \* 测试通过百度/google搜索解决问题  \*/  public class Test04 {  public static void main(String[] args) {  String str = null;  if(str!=null) {  System.out.println(str.substring(0));  }  Scanner scanner = new Scanner(System.in);  System.out.println("请输入您的年龄：");  int age = scanner.nextInt();  scanner.nextLine();  System.out.println("请输入您的姓名：");  String name = scanner.nextLine();  System.out.println("============");  System.out.println("姓名："+name+",年龄："+age);  //尽量精确的描述问题：Scanner nextInt nextLine 跳过 没有执行  }  } |

1. 完成课堂上关于IDE调试功能的测试案例。

|  |
| --- |
| /\*\*  \* 测试调试功能  \*/  public class Test05 {  private int num = 10;  public void run(int a) {  System.out.println("Test05.run");  for(int i=0;i<a;i++) {  num += a;  go();  }  }  public void go(){  System.out.println("Test05.go");  System.out.println("num\*10="+(num\*10));  }  public static void main(String[] args) {  Test05 test05 = new Test05();  test05.run(3);  }  } |

# 第45集 元素编号可重复：容器\_List的用法

1. ArrayList、LinkedList、Vector的区别是什么？

**ArrayList：**ArrayList底层是用数组实现的存储。 特点：查询效率高，增删效率低，线程不安全。

**LinkedList：**LinkedList底层用双向链表实现的存储。特点：查询效率低，增删效率高，线程不安全。

**Vector：**Vector底层是用数组实现的List，相关的方法都加了同步检查，因此“线程安全,效率低”。

1. 如何选用ArrayList、LinkedList、Vector？

* 需要线程安全时，用Vector。
* 不存在线程安全问题时，并且查找较多用ArrayList（一般使用它）。
* 不存在线程安全问题时，增加或删除元素较多用LinkedList。

1. 完成课堂上关于List的测试代码。

|  |
| --- |
| import java.util.ArrayList;  import java.util.Arrays;  import java.util.LinkedList;  import java.util.List;  /\*\*  \* 测试List接口的常用方法  \*/  public class ListTest {  public static void main(String[] args) {  test03();  }  //测试isEmpty,add,remove,size,contains,toArrays等常见方法.  //这些方法多位于Collection接口中。  public static void test01(){  //有序、可重复  List list = new ArrayList(); //泛型  System.out.println(list.isEmpty());  list.add("高淇");  System.out.println(list);  System.out.println(list.isEmpty());  list.add("高三");  list.add("高四");  System.out.println(list);  System.out.println("list的大小："+list.size());  System.out.println("是否包含指定的元素："+list.contains("高淇"));  list.remove("高四");  System.out.println(list);  Object[] objs = list.toArray();  System.out.println("转化成Object数组："+ Arrays.toString(objs));  list.clear();  System.out.println(list);  }  //测试和索引操作相关的方法  public static void test02(){  //List存储的是：有序、可重复  List list = new ArrayList();  list.add("A");  list.add("B");  list.add("C");  list.add("D");  System.out.println(list);  list.add(2,"高");  System.out.println(list);  list.remove(2);  System.out.println(list);  list.set(2,"张");  System.out.println(list);  System.out.println(list.get(1));  list.add("B");  System.out.println(list);  System.out.println(list.indexOf("B")); //从左到右找到第一个  System.out.println(list.lastIndexOf("B")); //从尾到头找到第一个  }  //测试两个容器之间的元素处理  public static void test03(){  // List list = new ArrayList();  List list = new LinkedList();  list.add("高淇");  list.add("高qi");  list.add("GQ");  List list2 = new ArrayList();  list2.add("高淇");  list2.add("张三");  list2.add("李四");  System.out.println(list);  System.out.println(list.containsAll(list2));  list.addAll(list2);  System.out.println(list);  // list.removeAll(list2); //删除掉list2和list中存在的元素  // System.out.println(list);  System.out.println("list:"+list);  System.out.println("list2:"+list2);  list.retainAll(list2); //取交集  System.out.println(list);  }  } |

# 第46集 独一无二不重复：容器\_Set的用法

1. Set的不可重复实现的核心是：equals()方法。对吗？

是的。如果equals()返回true，则认为是重复的。否则不重复。

1. 完成课堂测试HashSet的代码。

|  |
| --- |
| import java.util.HashSet;  import java.util.Objects;  import java.util.Set;  /\*\*  \* 测试Set  \* Set：无序、不可重复  \*/  public class SetTest {  public static void main(String[] args) {  test02();  }  //Set的常用方法  public static void test01(){  Set s = new HashSet();  s.add("hello");  s.add("world");  s.add("hello"); //相等的元素不会再被加入！  System.out.println(s);  //其他的方法和List一致。因为Set和List都是Collection接口的子接口！  System.out.println(s.size());  System.out.println(s.isEmpty());  }  //Set中不可重复的核心：equals()方法  public static void test02(){  Emp e1 = new Emp(1001,"张三");  Emp e2 = new Emp(1002,"李四");  Emp e3 = new Emp(1001,"王五");  Set s = new HashSet();  s.add(e1);  s.add(e2);  s.add(e3); //相当(equals()返回true)的元素不会再被加入！  System.out.println(s);  }  }  class Emp {  private int id;  private String ename;  @Override  public boolean equals(Object o) {  if (this == o) return true;  if (o == null || getClass() != o.getClass()) return false;  Emp emp = (Emp) o;  return id == emp.id;  }  @Override  public int hashCode() {  return Objects.hash(id);  }  public Emp(int id, String ename) {  this.id = id;  this.ename = ename;  }  @Override  public String toString() {  return "Emp{" +  "id=" + id +  ", ename='" + ename + '\'' +  '}';  }  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  public String getEname() {  return ename;  }  public void setEname(String ename) {  this.ename = ename;  }  } |

# 第47集 成对存取更高效：容器\_Map的用法

1. Map主要用于成对存储信息的场景。对吗？

是的。

1. 完成HashMap的常见用法测试代码。

|  |
| --- |
| //Map的常见用法  public static void test01(){  Map m1 = new HashMap(); //new HashTable();  m1.put(1,"one");  m1.put(2,"two");  m1.put(3,"three");  m1.put(2,"二"); //如果键重复(equals()为核心进行判断)，则替换掉旧的键值对  Object obj = m1.get(2);  System.out.println(obj);  System.out.println(m1);  System.out.println(m1.size());  System.out.println(m1.isEmpty());  System.out.println(m1.containsKey(4));  System.out.println(m1.containsValue("three"));  m1.remove(3); //从Map容器中移除这个键值对  System.out.println(m1);  Map m2 = new HashMap();  m2.put("yi","1111");  m2.put("er","2222");  m2.put(3,"3333");  m1.putAll(m2);  System.out.println(m1);  } |

1. 完成Map的遍历方式测试代码。

|  |
| --- |
| //泛型的简单用法  public static void test02(){  List<String> list = new ArrayList<>();  list.add("aaa");  String str = list.get(0);  System.out.println(str);  Set<Integer> set = new HashSet<>();  // set.add("dddd");  Map<String,String> map = new HashMap<>();  map.put("一","one");  map.put("二","two");  }  //遍历List、Set  public static void iterateListSet(){  List<String> s = new ArrayList<>();  // Set<String> s = new HashSet<>();  s.add("aa");  s.add("bb");  s.add("cc");  //通过索引遍历List (只适用于List)  for(int i=0;i<s.size();i++) {  String temp = s.get(i);  System.out.println(temp);  }  //增强for循环（foreach) （适用于：List、Set）  for(String temp:s) {  System.out.println(temp);  }  //使用Iterator对象 （适用于：List、Set）  for(Iterator<String> iter=s.iterator();iter.hasNext();) {  String temp = iter.next();  System.out.println(temp);  }  }  //遍历Map（遍历Key、遍历Value）  public static void test03(){  Map<String,String> map = new HashMap<>();  map.put("one","一");  map.put("two","二");  map.put("three","三");  //遍历键  Set<String> keys = map.keySet();  for(String temp:keys) {  // System.out.println(temp);  System.out.println(map.get(temp));  System.out.println(temp+"----"+map.get(temp));  }  //遍历值  Collection<String> values = map.values();  for(String temp:values) {  System.out.println(temp);  }  //使用EntrySet遍历key、value  Set<Map.Entry<String,String>> entrySet = map.entrySet();  for(Map.Entry e:entrySet) {  System.out.println(e.getKey()+"===="+e.getValue());  }  } |

# 第48集 表格数据容器存：容器\_存取二维表格信息

1. 完成课堂上使用List+Map存储表格信息的代码

|  |
| --- |
| //Map对应一行数据；  public static void test01(){  Map<String,Object> m1 = new HashMap<>();  m1.put("id",1001);  m1.put("title","我爱java");  m1.put("createTime","2021-8-10");  m1.put("length",300); //300秒  Map<String,Object> m2 = new HashMap<>();  m2.put("id",1002);  m2.put("title","我爱百战");  m2.put("createTime","2021-9-10");  m2.put("length",400);  Map<String,Object> m3 = new HashMap<>();  m3.put("id",1003);  m3.put("title","我爱编程");  m3.put("createTime","2021-10-10");  m3.put("length",500);  //存储了整个表格的信息  List<Map<String,Object>> list = new ArrayList();  list.add(m1);  list.add(m2);  list.add(m3);  System.out.println("id\t\ttitle\t\tcreateTime\tlength");  for(int i=0;i<list.size();i++) {  Map<String,Object> temp = list.get(i);  System.out.println(temp.get("id")+"\t"  +temp.get("title")+"\t\t"  +temp.get("createTime")+"\t"  +temp.get("length"));  }  } |

1. 完成课堂上使用List+Javabean存储表格信息的代码

|  |
| --- |
| package com.itbaizhan;  import java.util.ArrayList;  import java.util.HashMap;  import java.util.List;  import java.util.Map;  /\*\*  \* 使用容器的不同方式，存储表格信息  \*/  public class StoreTable {  public static void main(String[] args) {  test02();  }  //List+Javabean的方式  public static void test02(){  VideoInfo v1 = new VideoInfo(1001,"我爱java","2021-8-10",300);  VideoInfo v2 = new VideoInfo(1002,"我爱百战","2021-9-10",400);  VideoInfo v3 = new VideoInfo(1003,"我爱编程","2021-10-10",5000);  List<VideoInfo> list = new ArrayList<>();  list.add(v1);  list.add(v2);  list.add(v3);  System.out.println("id\t\ttitle\t\tcreateTime\tlength");  for(VideoInfo temp:list) {  System.out.println(temp);  }  }  }  class VideoInfo {  private int id;  private String title;  private String createTime;  private int length; //秒为单位  public VideoInfo(int id, String title, String createTime, int length) {  this.id = id;  this.title = title;  this.createTime = createTime;  this.length = length;  }  @Override  public String toString() {  return  getId() + "\t"  + getTitle() + "\t\t"  +getCreateTime()+"\t"  +getLength();  }  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  public String getTitle() {  return title;  }  public void setTitle(String title) {  this.title = title;  }  public String getCreateTime() {  return createTime;  }  public void setCreateTime(String createTime) {  this.createTime = createTime;  }  public String getLength(){  //超过60秒，显示几分几秒；超过3600秒，显示，几小时几分几秒  String lengthInfo = "";  if(this.length<60) {  return this.length+"秒";  }else if(this.length<3600) {  int minutes = length/60;  int seconds = length%60;  return minutes+"分"+seconds+"秒";  }else {  int hours = length/3600;  int minutes = (length%3600)/60;  int seconds = (length%3600)%60;  return hours+"时"+minutes+"分"+seconds+"秒";  }  }  public void setLength(int length) {  this.length = length;  }  } |

# 第49集 容器存取贴标签：泛型详解\_自定义泛型

1. 说一下，使用泛型的两个好处。

第一好处：代码可读性更好【不用强制转换】

第二好处：程序更加安全【只要编译时期没有警告，运行时期就不会出现ClassCastException异常】

1. 完成泛型的测试代码。

|  |
| --- |
| import java.util.ArrayList;  import java.util.List;  /\*\*  \* 测试泛型的基本用法  \*/  public class GenericTest1 {  public static void main(String[] args) {  }  public static void test1(){  List<String> list = new ArrayList<>();  list.add("ddd");  // list.add(2332);  Generic1<Integer> g1 = new Generic1<>();  g1.aa(22);  Generic1<String> g2 = new Generic1<>();  g2.aa("hello");  }  }  //定义的一个简单的泛型类  class Generic1<T> {  public T aa(T obj) {  System.out.println(obj);  return obj;  }  public <N> void bb(N obj){  System.out.println(obj);  }  }  interface MyList<E> {  int size();  boolean isEmpty();  void add(E e);  E get(int index);  }  class MyArrayList<E> implements MyList<E> {  @Override  public int size() {  return 0;  }  @Override  public boolean isEmpty() {  return false;  }  @Override  public void add(E e) {  }  @Override  public E get(int index) {  return null;  }  } |
| import java.util.ArrayList;  import java.util.List;  /\*\*  \* 通配符？和上下限限定  \*/  public class GenericTest2 {  //Dog和他的子类  public static void test01(List<? extends Dog> list){  System.out.println(list);  }  public static void test02(List<? super Dog> list){  System.out.println(list);  }  public static void main(String[] args) {  List<Animal> list1 = new ArrayList<>();  List<Dog> list2 = new ArrayList<>();  List<Taidi> list3 = new ArrayList<>();  // test01(list1);  test01(list2);  test01(list3);  test02(list1);  test02(list2);  // test02(list3);  }  }  class Animal {}  class Dog extends Animal{}  class Taidi extends Dog{} |

# 第50集 数据底层做容器：手写容器\_自定义实现ArrayList1

完成课堂上手写MyArrayList的代码实现！

|  |
| --- |
| import java.util.ArrayList;  import java.util.List;  /\*\*  \* 自定义的MyList接口。模仿的是：List接口  \* @param <E>  \*/  public interface MyList<E> { //Element  // List list = new ArrayList();  int size();  boolean isEmpty();  void add(E obj);  public void add(int index, E obj);  public E set(int index, E obj);  public E get(int index);  public boolean contains(E obj);  Object[] toArray();  boolean remove(E obj);  void remove(int index);  void clear();  } |
| import java.util.Arrays; import java.util.LinkedList;  public class MyArrayList<E> implements MyList<E> {   private int size;  private Object[] elementData;    @Override  public int size() {  return size;  }   @Override  public boolean isEmpty() {  return size==0;  }   public MyArrayList(){  this(10);  }   public MyArrayList(int initialCapacity){  if(initialCapacity<0) {  try {  throw new Exception("容器初始化大小不能为负数！");  } catch (Exception e) {  e.printStackTrace();  }  }  elementData = new Object[initialCapacity];  }   public static void main(String[] args) {  MyArrayList<String> list = new MyArrayList<>(10);  System.*out*.println(list);  list.add("aa");  list.add("bb");  list.add("cc");  list.add("dd");  list.add("ee");  list.add("ff");  list.add(2,"gaoqi");  }    @Override  public void add(E obj) {  if(size==elementData.length) { *//需要数组扩容* Object[] newArray = new Object[elementData.length\*2+1];  System.*arraycopy*(elementData,0,newArray,0,elementData.length);  elementData = newArray;  }   elementData[size] = obj;  size++;   }   @Override  public void add(int index, E obj) { *//在index位置处插入一个元素；其余元素后移！* rangeCheck(index);   ensureCapacity();   System.*arraycopy*(elementData,index,elementData,index+1,size-index);  elementData[index] = obj;  size++;  }   @Override  public E set(int index, E obj) {  return null;  }   @Override  public E get(int index) {  return null;  }   @Override  public boolean contains(E obj) {  return false;  }   @Override  public Object[] toArray() {  return new Object[0];  }   @Override  public boolean remove(E obj) {  return false;  }   @Override  public void remove(int index) {   }   @Override  public void clear() {   }   private void ensureCapacity(){  *//如果已经放满了元素，则扩容* if(size==elementData.length) { *//需要数组扩容* Object[] newArray = new Object[elementData.length\*2+1];  System.*arraycopy*(elementData,0,newArray,0,elementData.length);  elementData = newArray;  }  }   private void rangeCheck(int index) {  *//索引是否合法的判断* if(index<0||index>=size) { *//0,size-1* try {  throw new Exception("索引必须位于0到size-1之间");  } catch (Exception e) {  e.printStackTrace();  }  }  }    } |

# 第51集 容器存取四方法：手写容器\_自定义实现ArrayList2

完成课堂上手写MyArrayList的代码实现！

|  |
| --- |
| import java.util.Arrays; import java.util.LinkedList;  public class MyArrayList<E> implements MyList<E> {   private int size;  private Object[] elementData;    @Override  public int size() {  return size;  }   @Override  public boolean isEmpty() {  return size==0;  }   public MyArrayList(){  this(10);  }   public MyArrayList(int initialCapacity){  if(initialCapacity<0) {  try {  throw new Exception("容器初始化大小不能为负数！");  } catch (Exception e) {  e.printStackTrace();  }  }  elementData = new Object[initialCapacity];  }   public static void main(String[] args) {  MyArrayList<String> list = new MyArrayList<>(10);  System.*out*.println(list);  list.add("aa");  list.add("bb");  list.add("cc");  list.add("dd");  list.add("ee");  list.add("ff");  list.add(2,"gaoqi");  list.set(3,"z");   String str = list.get(4);  System.*out*.println(str);  System.*out*.println(list.contains("eee")); *// list.remove(2);* list.remove("z"); *// list.clear();* System.*out*.println(list);  }   @Override  public String toString() {  StringBuilder sb = new StringBuilder();  sb.append("[");  for(int i=0;i<size;i++) {  sb.append(elementData[i]+"\t");  }  sb.append("]");  return sb.toString();  }   @Override  public void add(E obj) {  if(size==elementData.length) { *//需要数组扩容* Object[] newArray = new Object[elementData.length\*2+1];  System.*arraycopy*(elementData,0,newArray,0,elementData.length);  elementData = newArray;  }   elementData[size] = obj;  size++;   }   @Override  public void add(int index, E obj) { *//在index位置处插入一个元素；其余元素后移！* rangeCheck(index);   ensureCapacity();   System.*arraycopy*(elementData,index,elementData,index+1,size-index);  elementData[index] = obj;  size++;  }   private void ensureCapacity(){  *//如果已经放满了元素，则扩容* if(size==elementData.length) { *//需要数组扩容* Object[] newArray = new Object[elementData.length\*2+1];  System.*arraycopy*(elementData,0,newArray,0,elementData.length);  elementData = newArray;  }  }   private void rangeCheck(int index) {  *//索引是否合法的判断* if(index<0||index>=size) { *//0,size-1* try {  throw new Exception("索引必须位于0到size-1之间");  } catch (Exception e) {  e.printStackTrace();  }  }  }   @Override  public E set(int index, E obj) { *//在指定索引位置，替换元素;并且将旧的元素返回* rangeCheck(index);   *//a b c d e* Object oldValue = elementData[index];  elementData[index] = obj;   return (E)oldValue;  }   @Override  public E get(int index){  rangeCheck(index);  return (E)elementData[index];  }   @Override  public boolean contains(E obj) {  for(int i=0;i<size;i++) {  if(elementData[i].equals(obj)) {  return true;  }  }  return false;  }   @Override  public Object[] toArray() {  return elementData;  }   @Override  public boolean remove(E obj) { *//删除指定的对象* for(int i=0;i<size;i++) {  if(elementData[i].equals(obj)) {  remove(i);  return true;  }  }  return false;  }   @Override  public void remove(int index) { *//删除指定位置的元素* rangeCheck(index);   *//删除指定位置的对象：a b c d e f ==> a b d e f null size-（index+1)* int numMoved = size-(index+1);  if(numMoved>0) {  System.*arraycopy*(elementData,index+1,elementData,index,numMoved);  }   size--;  elementData[size]=null;   }   @Override  public void clear() {  for(int i=0;i<size;i++) {  elementData[i] = null;  }   size=0;  } } |

# 第52集 链表底层造容器：手写容器\_自定义实现LinkedList1

完成课堂上手写MyLinkedList的代码实现！

|  |
| --- |
| package collection3;  import java.util.LinkedList;  public class MyLinkedList<E> implements MyList<E> {   private int size;   private Node first; *//第一个节点* private Node last; *//最后一个节点* @Override  public int size() {  return size;  }   @Override  public boolean isEmpty() {  return size==0;  }   @Override  public void add(E obj) {  *//给容器中增加一个obj，对应实际上需要增加一个节点（使用这个节点存储obj）* Node n = new Node();  if(first==null) {  *//增加第一个节点* n.setPrevious(null);  n.setObj(obj);  n.setNext(null);   *//first指向第一个节点；由于只有一个，last也指向这个节点* first = n;  last = n;   }else{  n.setPrevious(last);  n.setObj(obj);  n.setNext(null);   last.setNext(n);   last = n;   }   size++;   }   @Override  public void add(int index, E obj) {    }   @Override  public E set(int index, E obj) {  return null;  }   @Override  public E get(int index) {  return null;  }   @Override  public boolean contains(E obj) {  return false;  }   @Override  public Object[] toArray() {  return new Object[0];  }   @Override  public boolean remove(E obj) {  return false;  }   @Override  public void remove(int index) {   }   @Override  public void clear() {   }   public static void main(String[] args) {  MyLinkedList<String> list = new MyLinkedList<>();  list.add("aa");  list.add("bb");  list.add("cc");  list.add("dd");   }  }  */\*\*  \* 链表中的节点类  \*/* class Node {  Node previous; *//上一个节点* Node next; *//下一个节点* Object obj; *//真正存储的信息* public Node(Node previous, Node next, Object obj) {  this.previous = previous;  this.next = next;  this.obj = obj;  }   public Node(){}   public Node getPrevious() {  return previous;  }   public void setPrevious(Node previous) {  this.previous = previous;  }   public Node getNext() {  return next;  }   public void setNext(Node next) {  this.next = next;  }   public Object getObj() {  return obj;  }   public void setObj(Object obj) {  this.obj = obj;  } } |

# 第53集 链表元素手拉手：手写容器\_自定义实现LinkedList2

完成课堂上手写MyLinkedList的代码！

|  |
| --- |
| package collection3;  import java.util.LinkedList;  public class MyLinkedList<E> implements MyList<E> {   private int size;   private Node first; *//第一个节点* private Node last; *//最后一个节点* @Override  public int size() {  return size;  }   @Override  public boolean isEmpty() {  return size==0;  }   @Override  public void add(E obj) {  *//给容器中增加一个obj，对应实际上需要增加一个节点（使用这个节点存储obj）* Node n = new Node();  if(first==null) {  *//增加第一个节点* n.setPrevious(null);  n.setObj(obj);  n.setNext(null);   *//first指向第一个节点；由于只有一个，last也指向这个节点* first = n;  last = n;   }else{  n.setPrevious(last);  n.setObj(obj);  n.setNext(null);   last.setNext(n);   last = n;   }   size++;   }   public static void main(String[] args) {  MyLinkedList<String> list = new MyLinkedList<>();  list.add("aa");  list.add("bb");  list.add("cc");  list.add("dd");   Node n = list.node(2);  list.add(2,"gaoqi");  Node n2 = list.node(2);   String str = list.get(2);  System.*out*.println(str);  System.*out*.println(list.toString());  }   @Override  public String toString() {  StringBuilder sb = new StringBuilder();  sb.append("[");  for(int i=0;i<size;i++) {  sb.append(node(i).obj+"\t");  }  sb.append("]");  return sb.toString();  }   private void rangeCheck(int index) {  *//索引是否合法的判断* if(index<0||index>=size) { *//0,size-1* try {  throw new Exception("索引必须位于0到size-1之间");  } catch (Exception e) {  e.printStackTrace();  }  }  }   private Node node(int index){   rangeCheck(index);   Node temp = null;  if(first!=null) {  if(index<(size>>1)) {  temp = first;   for(int i=0;i<index;i++) {  temp = temp.next;  }  }else{  temp = last;  for(int i=size-1;i>index;i--) {  temp = temp.previous;  }  }   }   return temp;  }   @Override  public void add(int index, E obj) {  Node temp = node(index);   Node newNode = new Node();  newNode.obj = obj;   if(temp!=null) {  Node up = temp.previous;  up.next = newNode;  newNode.previous = up;   newNode.next = temp;  temp.previous = newNode;  }   size++;   }   @Override  public E set(int index, E obj) {  rangeCheck(index);   Node temp = node(index);  temp.obj = obj;  return (E)temp.obj;  }   @Override  public E get(int index) {  return (E)node(index).obj;  }   @Override  public boolean contains(E obj) {  for(int i=0;i<size();i++) {  if(node(i).obj.equals(obj)) {  return true;  }  }   return false;  }   @Override  public Object[] toArray() {  Object[] objs = new Object[size];   for(int i=0;i<size;i++) {  objs[i] = node(i).obj;  }   return objs;  }   @Override  public boolean remove(E obj) {  for(int i=0;i<size;i++) {  if(node(i).obj.equals(obj)) {  remove(i);  return true;  }  }  return false;  }   @Override  public void remove(int index) {   rangeCheck(index);   Node temp = node(index);   if(temp!=null) {  Node up = temp.previous;  Node down = temp.next;   up.next = down;  down.previous = up;   size--;   }  }   @Override  public void clear() {  for(int i=0;i<size;i++) {  remove(i);  }  } }  */\*\*  \* 链表中的节点类  \*/* class Node {  Node previous; *//上一个节点* Node next; *//下一个节点* Object obj; *//真正存储的信息* public Node(Node previous, Node next, Object obj) {  this.previous = previous;  this.next = next;  this.obj = obj;  }   public Node(){}   public Node getPrevious() {  return previous;  }   public void setPrevious(Node previous) {  this.previous = previous;  }   public Node getNext() {  return next;  }   public void setNext(Node next) {  this.next = next;  }   public Object getObj() {  return obj;  }   public void setObj(Object obj) {  this.obj = obj;  } } |

# 第54集 数组链表成Hash：手写容器\_自定义实现HashMap1

完成本次课堂MyHashMap的代码实现

|  |
| --- |
| package collection3;  import java.util.Arrays; import java.util.HashMap;  */\*\*  \* 手工实现的HashMap  \* 仅用于学习！  \*/* public class MyHashMap<K,V> implements MyMap<K,V> {  *// HashMap map;* private static final int *INITIAL\_CAPACITY* = 16;   private int size;   private Entry[] table;    public MyHashMap(){  table = new Entry[*INITIAL\_CAPACITY*];  }   public static void main(String[] args) {  MyMap<String,String> map = new MyHashMap<>();  System.*out*.println(map);  }    @Override  public void put(K key, V value) {    }   @Override  public V get(K key) {  return null;  }   @Override  public boolean containsKey(K key) {  return false;  }   @Override  public boolean containsValue(V value) {  return false;  }   @Override  public void remove(K key) {   }   @Override  public int size() {  return 0;  }   @Override  public boolean isEmpty() {  return false;  } }  class Entry<K,V> {  K key;  V value;  int hash;  Entry next; *//下一个节点* public Entry(K key, V value, int hash, Entry next) {  this.key = key;  this.value = value;  this.hash = hash;  this.next = next;  }   public Entry(){}  } |

# 第55集 大厂面试问Hash：手写容器\_自定义实现HashMap2

完成本次课MyHashMap的代码实现(put和get方法)

|  |
| --- |
| package collection3;  import java.util.Arrays; import java.util.HashMap;  */\*\*  \* 手工实现的HashMap  \* 仅用于学习！  \*/* public class MyHashMap<K,V> implements MyMap<K,V> {  *// HashMap map;* private static final int *INITIAL\_CAPACITY* = 16;   private int size;   private Entry[] table;    public MyHashMap(){  table = new Entry[*INITIAL\_CAPACITY*];  }   public static void main(String[] args) {  MyMap<String,String> map = new MyHashMap<>();  System.*out*.println(map);  map.put("a","AAA");  map.put("b","BBB");  map.put("c","CCC");  map.put("d","DDD");  map.put("a","啊啦啊");   String val = map.get("a");  System.*out*.println(val);  }    @Override  public void put(K key, V value) {   int index = hash(key);   Entry entry = new Entry(key,value,index,null);   if(table[index]==null){  table[index] = entry;  }else{  Entry e = table[index];  Entry last = e;   while(e!=null) {  if(e.key.equals(key)){  e.value = value; *//如果key相等，则直接覆盖value！* return;  }  last = e; *//保存最后一个* e = e.next;  }   *//上面整个循环结束，没有return；则说明整个单向链表中没有Entry的key和传入的key相等。  //则添加到链表的最后* last.next = entry;  }   size++;   }    private int hash(Object key){  int hashcode = key.hashCode(); *//返回的是int，有可能是负数* hashcode = hashcode<0?-hashcode:hashcode; *//确保正数  // int index = hashcode/hashcode; //最差的算法。hashmap退化成链表。 // int index = hashcode%table.length; //早起的jdk就是这样的算法。* return hashcode&(table.length-1); *//位运算，效率较高。产生[0,length-1]范围的一个数字* }    @Override  public V get(K key) {   int index = hash(key);   if(table[index]!=null){  Entry e = table[index];   while(e!=null) {  if(e.key.equals(key)){  return (V)e.value; *//如果key相等，则返回对应的value* }   e = e.next;  }   }    return null;  }   @Override  public boolean containsKey(K key) {  return false;  }   @Override  public boolean containsValue(V value) {  return false;  }   @Override  public void remove(K key) {   }   @Override  public int size() {  return 0;  }   @Override  public boolean isEmpty() {  return false;  }   }  class Entry<K,V> {  K key;  V value;  int hash;  Entry next; *//下一个节点* public Entry(K key, V value, int hash, Entry next) {  this.key = key;  this.value = value;  this.hash = hash;  this.next = next;  }   public Entry(){} } |

# 第56集 Set底层是Hash：手写容器\_自定义实现HashMap3\_HashSet

1. 完成本次课堂MyHashMap最终版代码

|  |
| --- |
| package collection3;  import java.util.Arrays; import java.util.HashMap;  */\*\*  \* 手工实现的HashMap  \* 仅用于学习！  \*/* public class MyHashMap<K,V> implements MyMap<K,V> {  *// HashMap map;* private static final int *INITIAL\_CAPACITY* = 16;   private int size;   private Entry[] table;    public MyHashMap(){  table = new Entry[*INITIAL\_CAPACITY*];  }   public static void main(String[] args) {  MyMap<String,String> map = new MyHashMap<>();  System.*out*.println(map);  map.put("a","AAA");  map.put("b","BBB");  map.put("c","CCC");  map.put("d","DDD");  map.put("a","啊啦啊");   String val = map.get("a");  System.*out*.println(val);  System.*out*.println(map);  map.remove("a");  System.*out*.println(map);  map.remove("c");  System.*out*.println(map);   }   @Override  public String toString() {  StringBuilder sb = new StringBuilder();  *//[{a:123,b:456},{a:123,c:333}]* sb.append("[");  for(Entry e:table) {  while(e!=null){  sb.append("{"+e.key+":"+e.value+"},");  e = e.next;  }  }  sb.append("]");  return sb.toString();  }   @Override  public void put(K key, V value) {   int index = hash(key);   Entry entry = new Entry(key,value,index,null);   if(table[index]==null){  table[index] = entry;  }else{  Entry e = table[index];  Entry last = e;   while(e!=null) {  if(e.key.equals(key)){  e.value = value; *//如果key相等，则直接覆盖value！* return;  }  last = e; *//保存最后一个* e = e.next;  }   *//上面整个循环结束，没有return；则说明整个单向链表中没有Entry的key和传入的key相等。  //则添加到链表的最后* last.next = entry;  }   size++;   }    private int hash(Object key){  int hashcode = key.hashCode(); *//返回的是int，有可能是负数* hashcode = hashcode<0?-hashcode:hashcode; *//确保正数  // int index = hashcode/hashcode; //最差的算法。hashmap退化成链表。 // int index = hashcode%table.length; //早起的jdk就是这样的算法。* return hashcode&(table.length-1); *//位运算，效率较高。产生[0,length-1]范围的一个数字* }    @Override  public V get(K key) {   int index = hash(key);   if(table[index]!=null){  Entry e = table[index];   while(e!=null) {  if(e.key.equals(key)){  return (V)e.value; *//如果key相等，则返回对应的value* }   e = e.next;  }   }    return null;  }   @Override  public boolean containsKey(K key) {  return false;  }   @Override  public boolean containsValue(V value) {  return false;  }   @Override  public void remove(K key) {   int index = hash(key);   if(table[index]!=null) {  Entry e = table[index];  Entry previous = null; *//上一个节点。如果没有上一个节点，则为null* while(e!=null) {  if(e.key.equals(key)){  if(previous==null){  *//说明是第0个节点* table[index] = e.next; *//它的下一个节点前移* }else{  *//说明是第n多个节点* previous.next = e.next; *//相当于把当前节点跳过（从链表中移除了）* size--;  }   }   previous = e;  e = e.next;   }   }     }   @Override  public int size() {  return size;  }   @Override  public boolean isEmpty() {  return size==0;  } }  class Entry<K,V> {  K key;  V value;  int hash;  Entry next; *//下一个节点* public Entry(K key, V value, int hash, Entry next) {  this.key = key;  this.value = value;  this.hash = hash;  this.next = next;  }   public Entry(){}  } |

1. 完成本次课堂MyHashSet的代码实现

|  |
| --- |
| package collection3;  */\*\*  \* 自定义实现的HashSet  \*/* public class MyHashSet {  MyHashMap map;  private static final Object *PRESENT*= new Object();  public MyHashSet(){  map = new MyHashMap();  }   public int size(){  return map.size();  }   public void add(Object obj){  map.put(obj,*PRESENT*); *//set的不可重复就是利用了map里面键对象不可重复的特点！* }   public static void main(String[] args) {  MyHashSet set = new MyHashSet();  set.add("aa");  set.add("bb");  set.add("cc");   }  } |