# 第一章 基础常识

## 计算机组成 :

1. 硬件 : 可以看得见, 摸的着的电子原器件

CPU 执行指令mov, add

外存(disk)持久 程序(Program)

内存(ram) 进程(process)

程序员关注的焦点只有一个 :　内存, 内存中全部都是一个个的存储单元

1. 软件 : 必须依赖硬件的存在而存在, 一系列的指令和数据的集合.
2. 应用软件 : 为了完成某个特定的功能的软件

运行在OS上的普通软件

1. 系统软件 : 操作系统OS, 对硬件的直接使用和管理.

通过OS才能使用到硬件.

**人机交互方式 : 人通过OS使用电脑.**

1. 图形化界面 使用简单, 直观. (GUI)
2. 命令行方式 使用复杂, 需要记住大量的命令(CLI), 优点 :　高效直接

启动控制台 就可以执行命令(操作系统可以识别的)

Win + r -> cmd -> 回车

DOS 磁盘操作系统(disk operating system), 基于文件系统工作的. 文件用目录来保存管理的.

命令行必须在某个目录下工作.

C:\Users\pauliuyou这是一个路径(path) 一条路, 一条路的尽头是一个文件或目录

前导符’>’左面的路径表示的目录就称为当前目录, 工作目录, 当前工作目录.

命令 :

dir (directory) 列出当前目录(文件夹)下的内容, 包括子目录和子文件

md (make directory) 在当前目录下创建新的子目录. 必须后面加上参数

命令行参数 : 跟在命令后面, 作用是给命令传递数据.

切换盘符

盘符 :

D:

切换工作目录

cd(change directory) 目标目录, 切换到目标目录为工作目录

练习 : 创建目录 D:\MyWork\JavaSE\day01\code

两个特殊目录

. 目录 代表的是当前目录

.. 目录 代表的是当前目录的父目录(上级目录)

Cd \ 作用是直接回到根目录

“mywork\javase\day01\code” 这是路径, 是一个相对路径, 相对当前目录而言

“.\mywork\javase\day01\code” 和上面的写法等价

D:\MyWork\JavaSE\day01\code 这也是路径, 是一个绝对路径. 特点是以根目录为开始的.

不用 cd \ 如何回到根目录

命令分类 :

1. 内部命令 : 操作系统内置的, 直接执行.
2. 外部命令 : 保存在硬盘上的可执行文件. 在执行时 操作系统必须能够找得到, 需要依赖path环境变量. 在命令行输入 path命令即可查看配置.

语言的发展历程

第一代 打孔机 面向硬件开发

第二代 汇编 面向CPU开发

第三代 c/c++ 面向OS, 通过系统调用完成功能.

第四代 java 面向VM, VM是用软件模拟的一台机器, 此机器的所有硬件规格都是不变的统一的.

Java的8 个特点

1. 简单 : 相对于C/C++简单
2. 面向对象 : 纯面向对象语言, 和面向过程相对. 面向过程关注的是问题解决的步骤. 面向对象关注的是谁能解决问题, 找谁.
3. 分布式 : 基于网络的多主机协作.
4. 健壮 : 强类型,所有数据都必须有特定的数据类型. 异常处理, 垃圾自动收集机制

垃圾 : 内存中不再使用的空间但是仍然被占着, 清理垃圾就是标记内存为可用

1. 安全 : 所有程序必须经过类加载器(ClassLoader) 来检查
2. 跨平台 : 因为java程序依赖的JVM是跨平台.
3. 性能好 : java程序是编译型, 不是解释型.
4. 多线程 : 提高吞吐量, 最大化利用CPU

* Java两种核心机制
  + Java虚拟机（Java Virtal Machine）
  + 垃圾收集机制（Garbage Collection）

JRE (运行环境) = JVM + 核心类库

JDK(开发工具) = JRE + 工具

javac 是编译器

java 是运行器

* 步骤：
  1. 将 Java 代码**编写**到扩展名为 .java 的文件中。

在D:\MyWork\JavaSE\day01\code目录下新建一个文本文件, 并改名为Hello.java然后用NotePad++打开编辑此文件

* 1. 通过 javac 命令对该 java 文件进行**编译**。

启动命令行控制台, 把工作目录切换到刚才的目录

” D:\MyWork\JavaSE\day01\code” 以这个目录为工作目录, 执行命令

javac Hello.java

编译的结果会产生一个Hello.class字节码文件.

* 1. 通过 java 命令对生成的 class 文件进行**运行**。

仍然是在刚才的命令行窗口中, 还是以刚才的目录为当前工作目录, 执行命令

java Hello(类必须是主类)

改变源文件的编码 :

全选所有内容并复制

菜单”格式”->”ANSI” 就会出现乱码

全选所有内容, 删除, 再重新粘贴刚才复制好的内容.

类 {

方法1 {

语句1;

语句2;

…..

}

方法2 {

}

}

主方法 : 入口方法, 写法固定

public static void main(String[] args) {}

java Test

java Hello.

程序执行的过程 :

Java 的执行作用是创建JVM

Hello 在创建好的JVM中通过类加载器加载主类, 找到主类中的主方法, 然后执行它.

主方法一旦结束, 进程也结束, 同时JVM销毁.

|  |
| --- |
| // 注释 : 源码中一些帮助性提醒性的文字, 它并参与程序执行, 在编译时被编译器丢弃掉. 这个注释称为单行注释, 只对当前行有用  /\*  多行注释, 有开始有结束  还是注释, 多行注释不可以嵌套  \*/  /\*\*  这个是文档注释, 它有特殊作用.  \*/  /\*  public 是一个修饰符  class 是类, 定义声明一个类  Test 是类名, 类名后面的一对{} 及其中的内容 称为类体(body)  类 = 类头(类签名) + 类体  类是java程序的最基本单位.  类中可以包含若干方法.    主类 : 包含主方法(入口)的类就称为主类  \*/  public class Test {    /\*  这是一个方法, 方法也称为函数, 是某个独立的功能体单位  public 是公共的, static 静态的 都是修饰符  void main(String[] args) 方法的签名  void 返回值类型, 是空类型, 表示方法没有返回值  main 是方法名(标识符) main是主要的意思, 这个方法称为主方法  (String[] args) 称为参数列表  方法名后面的一对{} 及其中的内容 称为方法体  方法 = 方法头(修饰符+方法签名) + 方法体;  方法和方法必须是平行关系, 方法中不可以嵌套方法.  main方法称为主方法, 程序的入口方法, 程序总是从main方法开始执行...  \*/  public static void main(String[] args) {  // 语句 : java程序的最小执行单位, 语句必须以;结尾  // 语句必须隶属于某个方法.  System.out.println("这是字符串内容, 可以随便写");  System.out.println("这也是一个语句");  }    /\*  public static void main(String[] args) {  System.out.println("main2");  }\*/    // 这是一个普通方法, 普通方法不是首先执行  public static void test() {  System.out.println("test()...");  }    }  // 此类和上面的类不冲突, 可以在源文件中共存.  // 非主类  class Test2 {    public static void test2() {  System.out.println("test2()...");  }  }  // 主类 : 包含了主方法  class Test3 {    public static void main(String[] args) {  System.out.println("Test3 main()....");  }  } |

主类 : 包含主方法(入口方法)的类.

公共类 : 被public修饰的类就称为公共类

公共类的类名必须和文件名一致, 所以一个源文件中, 只能有一个公共类. 非公共类不限.

注意点 :　对程序的任何修改, 都必须即时保存文件并必须重新编译才能生效.

* Java源文件以“java”为扩展名。源文件的基本组成部分是类（class），如本类中的HelloWorld类。
* 一个源文件中最多只能有一个public类。其它类的个数不限，如果源文件包含一个public类，则文件名必须按该类名命名。
* Java应用程序的执行入口是main()方法。它有固定的书写格式：**public static void main(String[] args) {...}**
* Java语言严格区分大小写。
* Java方法由一条条语句构成，每个语句以“;”结束。
* 大括号都是成对出现的，缺一不可。

# NotePad++ 快捷键

Ctrl + c 复制

Ctrl + a 全选

Ctrl + v 粘贴

Ctrl + s 保存

Ctrl + x 剪切

Ctrl + z 撤销

Ctrl + y重做

Shift + tab 把所有选中的内容整体左移

Ctrl + d快速复制当前行

Ctrl + l 快速删除当前行

# 第二章 基本语法

## 关键字

关键字 : 在java程序中有特殊含义作用的单词, 字母全部小写

**保留字 : 从C++中保留下来的关键字, 在java中目前还不是关键字, 将来有可能是.**

标识符 : 标识某个东西的符号, 简单的来讲就是名字.

定义合法标识符规则

1. 由52个字母, 0~9 10个数字字符, \_和$(尽量不要使用)
2. 数字不可以开头, 标识符3C 就是非法, C3是合法
3. 不可以使用关键字和保留字, 但是可以包含关键字和保留字
4. 大小写敏感的, 长度有限制(65535)
5. 标识符中不能包含空格, 因为空格也有特殊含义.

命名规范 :

1. 包名 : 所有字母都小写 mypackage
2. 类名 : 首字母大写, 后面的每个单词的首字母也大写 : MyClassName
3. 变量名或方法名 : 首字母必须小写, 后面的单词的首字母大写 : myVarName
4. 常量名 : 所有字母都大写, 单词之间用\_隔开 : MY\_CONST\_NAME

变量 : 内存中的一块被命名并且要被特定的数据类型约束的区域, 此区域中可以保存数据.

要有变量名 : 为了是正确的定位内存空间.

要有数据类型 : 数据类型决定了空间大小, 此空间中可以保存什么数据,这个数据能做什么.

变量的声明 : 数据类型 变量名;

变量的使用 : 通过变量名来使用.

如 : int n1;

* **变量的概念：**
  + 内存中的一个存储区域
  + 该区域有自己的名称（变量名）和类型（数据类型）
  + Java中每个变量必须先声明，后使用
  + 该区域的数据可以在同一类型范围内不断变化

|  |
| --- |
| public class VariableTest {    public static void main(String[] args) {  // 变量声明 : 数据类型 变量名;  int n1; // 作用是在内存中开辟一块4字节空间, 并用n1符号与此空间联系起来  n1 = 10; // 把10这个值写入n1符号指向的空间中. 这是一个赋值操作, 把右侧的值写入左面的变量  System.out.println(n1); // 把n1符号指向的空间中的值取出来并打印输出.    int n2 = n1; // 把n1变量中的值复制出来, 再写入n2变量内存空间中  System.out.println(n2);    n1 = n1 \* n2; // 一个变量在某个时刻只能保存一个值, 一旦被重新赋值, 之前的老值永远无法找回  System.out.println(n1); // 变量有时间性, 不同时刻变量中的值有可能不一样.    }  } |

## 每日一考\_day02

1. 写出java语言的8个特性

1) 简单 相对于C++

2) 面向对象 和面向过程相对, 关注的是具有功能的对象(角色)

3) 分布式 基于网络的多主机协作

4) 健壮 强类型(所有数据都必须要有唯一的确定的数据类型)

异常处理, GC(垃圾收集), 指针安全化(引用reference)

5) 安全 java程序必须在VM中运行, 所以要想执行,必须通过类加载器的检查

6) 跨平台 VM跨平台, 不同平台有不同的VM.

7) 性能好 编译型语言比解释型要快. HotSpot热点

8) 多线程 CPU硬件支持的.

2. 描述一下语句,类,和方法之间的关系.

类 { // 类是java程序的基本单位

方法 { // 一个功能的单位体, 方法不可以嵌套.

语句1; // 语句是java程序最小执行单位.

语句2;

}

}

3. 什么是主类, 什么是公共类, 公共类有什么注意点?

包含主方法的类就是主类

被public修饰的类

公共类的类名必须和源文件名一致.

4. 变量是什么?

内存中的一块被命名的被特定的数据类型约束区域, 此区域中可以保存数据类型范围内的数据.

变量的声明 :

数据类型 变量名;

5. 使用变量有哪些注意事项(至少3点)?

1) 必须先声明后使用

2) 必须要有数据类型和变量名

3) 变量有其作用范围, 变量的声明语句所在的一对{}内.

{

int n = 10;

}

{

System.out.println(n); // 变量不能使用, 原因是不在这个范围内

}

变量声明(declare) : 是一个宣告, 宣告某个变量名和某块空间产生关联

int n = 10;

4 同一范围内, 变量不允许重复声明( 重婚罪)

5 变量内存空间中保存的数据只能在其范围内变化.

变量 : 变化的量,在内存中.

变量按照数据类型分类 :

1. 基本数据类型(8种) : 内存空间中保存的是数据本身

整数 : 是实打实存储, 精准

byte 1字节 -128~127

short 2字节 -32768~32767

int 4字节 -20多亿~20多亿

long 8字节 -巨大~巨大(900多亿亿)

浮点数 : 是近似值不精准

float 4字节 -10^38~10^38

double 8字节 -10^308~10^308

字符

char 2字节

布尔

boolean 1字节 true, false

1. 引用数据类型 : 内存空间中保存的是其他数据的地址

字符串是引用类型

代码

|  |
| --- |
| class VariableTest4 {    public static void main(String[] args) {  byte b1 = 10;  short s1 = 20;  int i1 = 30;    // 任意非long整数变量作运算, 结果总是升级成int型  //s1 = b1 + s1; // 任意非long整数作运算, 结果总是升级成int型  //s1 = (short)b1 + (short)s1;  s1 = (short)(b1 + s1); // 变量中的值对于编译器来说, 是不确定.  s1 = 20 + 50; // 右侧如果是常量, 编译器很确定 翻译成 s1 = 70;    long l1 = 40L;    // 各种类型的变量混合运算, 结果总是升级成范围最大的那个类型  // 右侧相当是3个long型做运算, 结果肯定是long型.  //i1 = l1 \* b1 + s1;  i1 = (int)l1 \* b1 + s1;    float f1 = 0.22F;  double d1 = 0.33;    //l1 = f1 + l1 + b1; // 不可以  l1 = (long)(f1 + l1 + b1);  d1 = d1 \* s1 + l1; // 可以    }  }  class VariableTest3 {    public static void main(String[] args) {  int n1 = 200; // n1是变量, 200是值 也有数据类型是int类型    // 常量 : 内容不可改变的量, 包括字面量和final修饰的量.  //300 = 300; // 这个语句不可以, 原因是左面的常量不允许被写入.  // 赋值符号 左侧 永远必须是变量    //int 3 = 3; 左面的3的变量名不合法.  //'a' = 'b'; "abc", 3.22 这些都是字面量      // 整数字面量默认是int型  byte b1 = 10; // 右侧是整数字面量  short s1 = 20; // 编译器优化, 20这个常量是在short的范围内的, 如果超出范围就无法优化  //short s2 = n1; // 编译器无法测量变量中的值  int i1 = 30; // 右侧是整数字面量, 是用int型4字节来保存  long l1 = 40; // 右侧是整数字面量, 也是用int型4字节来保存  //long l2 = 3000000000; // 右面的字面量尝试用4字节int来保存, 存不下, 所 以报错  long l2 = 3000000000L; // 后缀L的作用是告诉编译器,此字面量用8字节的long型来保存.    // 浮点数字面量默认数据类型是double型  double d1 = 3.22D;  //float f1 = 0.23; // 浮点数字面量默认数据类型是double型  float f1 = (float)0.23; // 浮点数不进行优化.  float f2 = 2.33F; // F的作用也是告诉编译器, 此字面量用4字节的float型来保存    f1 = i1; // 范围小的量值给范围大的变量写入. 可以写入  i1 = (int)f1; // 范围大的量值给范围小的变量写入, 不可以直接完成, 可以强转    l1 = (short)f1; // 可以!!    // 基本数据类型的数值范围大小, 从小到大排序 :  // byte < short < int < long < float < double    //d1 = ?; // 右面可以是什么类型, 任意类型, 兼容性最好.  }  }  class VariableTest2 {    public static void main(String[] args) {  // 数据类型 数值型 : byte, short, int, long, float, double  byte b1 = 10;  byte b2 = 11;  short s1 = 20;  short s2 = 2100;    // 看到赋值操作, 内心要紧张.  b2 = b1; // 这是赋值操作, 特点 : 反人类, 从右向左, 如果右面没有得出确定的值, 不能向左走.    s1 = b2; // 可以完成, 右面的数据类型范围小, 左面的数据类型的范围大    //b1 = s2; 直接操作不允许  // (目标类型)数据;  // 强制类型转换有风险, 风险自负, 程序员必须知道自已在做什么.  b1 = (byte)s2; // 如果要把范围大的变量中的值写入范围小的变量中, 必须强制类型转换  System.out.println(b1);    s2 = (short)b1; // 范围小的量值写入范围大的变量中时 不需要强制类型转换  }  }  public class VariableTest {    public static void main(String[] args) {  // 变量的使用事项    //n = 200; 变量应该先声明, 后使用  //int n;  //System.out.println(n);    {  int n = 10;  }  {  //System.out.println(n); // n的使用范围只在它声明语句 的{} 内  }    int n2 = 20;  //long n2 = 30; 同一个范围内, 变量只能声明一次, 不允许重复声明    //short s1 = 50000; 变量不可以超范围保存数据    int n3;  //System.out.println(n3); // ? 变量没有初始化不可以读取  n3 = 30; // 初始化赋值, 变量声明好后的第一次赋值.  n3 = 50; // 普通赋值  n3 = 60;    int n4 = 200; // 最好这样写, 声明和初始化同时进行, 防止忘记初始化.    }  } |

一些不可见的特殊控制字符的转义



字符串 , 由一系列字符串起来

// 字符串可以和任意数据作拼接, 拼接的结果是原内容加上新内容的新字符串

字符串是内容不可以改变的对象, 所以任何的字符串连接都会产生新对象.

String s2 = ""; // 空串, 含义是有字符串对象, 但是里面没有内容

String s3 = null; // s3是空地址, 含义是什么也没有, 当然也不可能有内容\

// 在程序中能用变量的地方绝不用常量, 因为变量灵活

## 进制

293486

2 \* 10 ^ 5 +

9 \* 10 ^ 4 +

3 \* 10 ^ 3 +

4 \* 10 ^ 2 +

8 \* 10 ^ 1 +

6 \* 10 ^ 0

权值 : 也就是位分和位相关 越在左越大, 10进制求权值是以10为底的n次幂

这样的数称为10进制, 只有10个数字表示大小 ‘0’ ~ ‘9’ 逢10进1

计算机底层用二进制, 计算权值是以2为底的n次幂, 逢2进1

还需要掌握的有16进制, 计算权值是以16为底的n次幂, 逢16进1

0x32

3 \* 16 ^ 1 +

2 \* 16 ^ 0 = 50

0011 0010

0 \* 2 ^ 7 + // 128

0 \* 2 ^ 6 + // 64

1 \* 2 ^ 5 + // 32

1 \* 2 ^ 4 + // 16

0 \* 2 ^ 3 + // 8

0 \* 2 ^ 2 + // 4

1 \* 2 ^ 1 + // 2

0 \* 2 ^ 0 // 1

= 50

十进制的50和0x32, 和二进制0011 0010 之间的关系, 是同一个东西.

练习 :

0x6211

6 \* 16 ^ 3 // 4096 \* 6

2 \* 16 ^ 2 // 256 \* 2

1 \* 16 // 16

1 \* 1 // 1 +

= 25105 这是’我’的Unicode码值

char ch = ‘\u6211’;

0010 1010 1010 0111, 一些2的n次幂的和

0 \* 2 ^ 15 + // 32768

0 \* 2 ^ 14 + // 16384

1 \* 2 ^ 13 + // 8192

0 \* 2 ^ 12 + // 4096

1 \* 2 ^ 11 + // 2048

0 \* 2 ^ 10 + // 1024

1 \* 2 ^ 9 + // 512

0 \* 2 ^ 8 // 256

1 \* 2 ^ 7 + // 128

0 \* 2 ^ 6 + // 64

1 \* 2 ^ 5 + // 32

0 \* 2 ^ 4 + // 16

0 \* 2 ^ 3 + // 8

1 \* 2 ^ 2 + // 4

1 \* 2 ^ 1 + // 2

1 \* 2 ^ 0 // 1

1 \* 2 ^ 13 + // 8192

1 \* 2 ^ 11 + // 2048

1 \* 2 ^ 9 + // 512

1 \* 2 ^ 7 + // 128

1 \* 2 ^ 5 + // 32

1 \* 2 ^ 2 + // 4

1 \* 2 ^ 1 + // 2

1 \* 2 ^ 0 // 1

= 10919

十进制 二进制 十六进制

0 0000 0

1 0001 1

2 0010 2

3 0011 3

4 0100 4

5 0101 5

6 0110 6

7 0111 7

8 1000 8

9 1001 9

10 1010 A

11 1011 B

12 1100 C

13 1101 D

14 1110 E

15 1111 F

一个十六进制正好可以对应4个bit, 4个位

两个十六进制正好可以对应8个bit, 正好一个字节.

0xA3 -> 1010 0011

0xC2B5E7A9 ->

1100 0010 1011 0101 1110 0111 1010 1001

0101 0110 1110 1010 0101 0110 0101 0100 -> 32bit

0x56EA5654

0xD752C9BD ->

1101 0111 0101 0010 1100 1001 1011 1101

0101 1010 1101 0101 0101 1010 1110 1110 ->

0x5AD55AEE

计算底层全部都是二进制, 并且都是以二进制补码的形式保存数据

有符号二进制数的符号位永远在最高位, 如果符号位为0, 表示这个数是正数

如果符号位为1 表示这个数是负数

正数的补码就是自身

负数的补码是它的相反数全部取反再加1

byte b1 = 0b0010 1101; 这是正数

0x2D -> 2 \* 16 + 13 => 45 所以0010 1101是+45

负数的补码是它的相反数全部取反再加1

X 全部取反 + 1 =

Byte b2 = 0b1101 1001; 这是一个负数, 负多少, 找相反数

-1 -> 1101 1000

取反 -> 0010 0111 -> 0x27 -> 39

所以 1101 1001 是-39的真实存储.

分析 :

0011 0110是多少, 是正数

0x36 -> 3 \* 16 + 6 = 54 所以0011 0110 就是+54

1110 1010是多少, 是负数, 负多少?

-1 -> 1110 1001

取反 -> 0001 0110 => 0x16 => 22

所以1110 1010 就是-22在计算机中的真实存储.

byte型最大值

二进制表示

0111 1111 => 0x7F => 7\*16 + 15 = 127

1111 1111 这是负数, 负多少? 符号位为1, 后面全是1, 表示负数最大

-1 -> 1111 1110

取反 -> 0000 0001 => 1, 所以1111 1111 是-1

1000 0000 这是负数, 负多少? 符号位为1, 后面全是0表示负数最小

-1 -> 0111 1111

取反 -> 1000 0000 => 0x80 => 16\*8 = 128, 所以1000 0000是-128

正数最大 + 1 => 最小负数

最小负数 -1 => 最大正数

Byte最小值~最大值 : 1000 0000 ~ 0111 1111 (0x80 ~ 0x7F)

Short型最大值

0111 1111 1111 1111 => 0x7FFF => 32767

Short型最小值

1000 0000 0000 0000 => 0x8000 => -32768

Int型最大值

0111 1111 1111 1111 1111 1111 1111 1111 => 0x7FFFFFFF

Int型最小值

1000 0000 0000 0000 0000 0000 0000 0000 => 0x80000000

Long型最大值

0x7FFFFFFFFFFFFFFF

Long型最小值

0x8000000000000000

Char型最大值 因为没有符号位, 所以最高位也是数据部分

1111 1111 1111 1111 => 0xFFFF

Char型最小值

0000 0000 0000 0000 -> 0x0000

Int n = 09; // 八进制以0开始.

## 每日一考\_day03

1. 列出变量使用注意事项(至少5点), 变量 : 内存中的一块区域, 此区域中可以保存数据

1) 变量必须要变量名, 数据类型

2) 先声明, 后使用

3) 有其作用范围 : 声明语句所隶属的一对{}内有效.

4) 在同一个范围内, 变量不能重复声明.

5) 变量必须初始化后才能读取. 通常做法是 : 在声明变量的同时进行初始化

6) 变量内存区域中保存的数据只能在其数据类型范围内变化.

2. 变量按照数据类型来分, 分为基本数据类型变量和引用数据类型变量, 请写出基本数据类型变量和引用数据类型变量的区别.

数据类型来分 :

1. 基本数据类型 : 内存区域中保存的数据本身
2. 引用数据类型 : 内存区域中保存的对象地址.

3基本数据类型有8种, 写出8种基本数据类型, 并用的16进制形式写出所有整数数据类型的最小值和最大值

byte 1 1000 0000 ~ 0111 1111 (0x80~0x7F)

short 2 1000 0000 0000 0000 ~ 0111 1111 1111 1111(0x8000~0x7FFF)

int 4 1(31个0)~0(31个1) 0x80000000 ~ 0x7FFFFFFF

long 8 0x8000000000000000 ~ 0x7FFFFFFFFFFFFFFF

float 4

double 8

char 2 本质是整数 0000 0000 0000 0000 ~ 1111 1111 1111 1111 (0x0000 ~0xFFFF)

boolean 1

4. 判断下列的带( )行的对错

int i1 = 20;

short s1 = i1; (F)

char c1 = 97; (T)

char c2 = '我' - '你';(T) // 编译器会把常量优化了.

char c3 = (char)(c1 - 32); //可以达到小写变大写的效果(T)

float f1 = i1;(T)

long l1 = 234234239933;(F)

f1 = l1 \* 20;(T)

double d1 = .342;

d1 = i1 \* f1 \* l1;(T)

l1 = f1 / 10000;(F)

boolean b1 = (boolean)1;(F)

5. 相同字面量十六进制表示的数比十进制要大, 对或错? 为什么?

错, 不一定

0x5和5 是一样大

0x10和10 是十六进制大

0x80和80 如果0x80是byte型, 十六进制小

## 变量 : 内存中被命名被数据类型约束的一块区域, 此区域中可以保存数据

数据类型作用是 决定空间大小, 空间可以保存什么数据, 这个数据能做什么.

变量声明 : 数据类型 变量名;

变量分类 :

1. 变量按照数据类型来分 :

1) 基本数据类型 : 内存区域中保存数据本身 不同的类型, 空间大小不同

2) 引用数据类型 : 内存区域中保存内存地址 不同的类型, 空间大小相同, 和JDK位数相关

内存地址 : 内存的每个字节都有一个顺序的编号, 这个编号就是了.

本质上来讲就是无符号正整数. 0表示null 空地址.

1. 变量按照声明位置来分 : 声明位置决定了变量的作用范围
2. 局部变量 : 声明在方法中的变量.

范围小, 寿命短.

1. 成员变量 : 声明在类中方法外的变量.

范围大, 寿命长.

1. 实例变量 没有static修饰 隶属于对象 保存在对象所在的GC区
2. 类变量 有static修饰 隶属于类, 保存在类所在的方法区

## 运算符 :

// &是只要有0结果就是0

// 0000 0000 0000 0000 0000 0000 0101 1010 &

// 0000 0000 0000 0000 0000 0000 1100 0011 =

// 0000 0000 0000 0000 0000 0000 0100 0010 => 0x42 -> 66

System.out.println(n1 & n2);

// |是只要有1结果就是1

// 0000 0000 0000 0000 0000 0000 0101 1010 |

// 0000 0000 0000 0000 0000 0000 1100 0011 =

// 0000 0000 0000 0000 0000 0000 1101 1011 => 0xDB -> 13 \* 16 + 11

System.out.println(n1 | n2);

// ^是只要不同就是1

// 0000 0000 0000 0000 0000 0000 0101 1010 |

// 0000 0000 0000 0000 0000 0000 1100 0011 =

// 0000 0000 0000 0000 0000 0000 1001 1001 => 0x99 -> 9 \* 16 + 9

System.out.println(n1 ^ n2);

+ 是取一个数本身

* 取相反数

/ 如果是两个整数相除, 结果会直接丢弃小数部分, 有可能会丢失精度.

% 取余

1. 如果n % m == 0 说明n能被m整除.
2. 如果n % 2 == 0 说明n是偶数, 如果n % 2 != 0, 说明n是奇数
3. N % M 的结果总是小于M, 让一个完全未知的数落在一个已知的M范围内.

如果对负数取模，可以把模数负号忽略不记，如：5%-2=1。 但被模数是负数则不可忽略。此外，取模运算的结果不一定总是整数。

++n 前加加, 先加后用, 不需要临时空间 效率高

n++ 后加加, 先用后加, 需要一个临时空间保存老值(也就是用的值), 效率低

int n = 10; n += 30;

效果相当于 n = n + 30, 但是不会引起数据类型的变化, 更安全.

比较运算的结果总是boolean

比较大小的运算只适用于基本数据类型中的数值型

==, != 可以适用于任意数据类型.

在Java中不可以写成3 < x < 6. 为什么? 因为 3 < x结果是boolean, 让boolean再和6比大小出问题.

**逻辑运算符 只适用于布尔之间.**

**逻辑与 : 只要有假就是假**

**& 和 && 的区别 就是双& 有短路效果. 在实际使用中必须使用这个.**

**逻辑或 : 只要有真就是真**

**| 和 || 的区别 双|| 有短路效果, 在实际使用必须使用这个.**

变量 = (布尔表达式) ? 表达式1 : 表达式2; //表达式1和表达2的值类型要一致.

运算符优先级, 先记住最高的.() 最低的赋值和累操作

结合性 由高到低

|  |  |
| --- | --- |
|  | . () {} ; , |
| R—>L | ++ -- ~ !(data type) |
| L—>R | \* / % |
| L—>R | + - |
| L—>R | << >> >>> |
| L—>R | < > <= >= instanceof |
| L—>R | == != |
| L—>R | & |
| L—>R | ^ |
| L—>R | | |
| L—>R | && |
| L—>R | || |
| R—>L | ? : |
| R—>L | = \*= /= %= |
|  | += -= <<= >>= |
|  | >>>= &= ^= |= |

## 流程控制

* **顺序结构**
* 程序从上到下逐行地执行，中间没有任何判断和跳转。
* **分支结构**
* 根据条件，选择性地执行某段代码。
* 有if…else和switch两种分支语句。

If

// if (条件布尔) {

//语句块; 如果条件布尔为真时此语句块就执行.

//}

if (n == 5) {

System.out.println("n == 5"); // 有条件执行语句

}

If else

If else if else if…

switch

* **循环结构**
* 根据循环条件，重复性的执行某段代码。
* 有while、do…while、for三种循环语句。

## 每日一考\_day04

1. 变量分类有两种分法, 第一种是按数据类型来分,另外一种是按照声明位置来分, 每一种又各分为哪些种类型. 各有什么特点?

变量分类 :

1. 按照数据类型来分
2. 基本数据类型 : 内存区域中保存的是数据本身
3. 数值型
4. 整数
   1. byte
   2. short
   3. int
   4. long
   5. char
5. 浮点数
6. float
7. double

只有数值型之间可以强制互相转换, 只有数值型之间可以比大小.

在赋值时 :

如果右侧的量值范围大于左侧的变量类型的范围, 必须强制类型转换(强转有风险)

如果右侧的量值范围小于左侧的变量类型的范围, 可以直接自动完成.

double > float > long > int > short > byte

> char

3)布尔型 boolean : true, false

1. 引用数据类型 : 内存区域中保存的是对象地址 : 对象在内存中的某个字节的编号. 本质上是一个正整数, 所有的引用变量都8字节(64位JDK)

String s = “abc”;

1. 按照声明位置来分
2. 局部变量 范围小, 寿命短.

声明在方法中的变量

1. 成员变量 范围大, 寿命长

声明在类中方法外的变量

2. 计算下列结果, 分析过程, 只需要计算到十六进制形式即可.

int a = 0x6B;

int b = 0x5D;

// 0110 1011 &

// 0101 1101 =

// 0100 1001 => 0x49

System.out.println(a & b);

// 0110 1011 |

// 0101 1101 =

// 0111 1111 => 0x7F

System.out.println(a | b);

// 0110 1011 ^

// 0101 1101 =

// 0011 0110 => 0x36

System.out.println(a ^ b);

3. 运算符%的作用是什么? 有什么实际的应用?

1) n % m 如果结果是0, 说明n能被m整除, 如果结果不为0, 说明n不能被m整除

2) n % 2 如果结果是0, 说明n是偶数, 如果结果是非0, 说明n是奇数

3) n % m 结果总是0~m-1,

判断一个数是否是奇数

If (n & 0x01 == 1) {

说明它一定是奇数

}

4. 判断:

1) if else if else if else 语句中, 如果同时有多个条件都为true, 那么将会有多个语句块被执行, 错误, 因为if else 是绝对的分支, 多语句块之间互斥

2) switch case case default 语句中, 如果同时有多个条件都为true, 那么将会有多个语句块被执行

如果是switch, case, case之间绝对不允许相同

Switch(n) {

Case 1 :

Case 2 :

Case 1 : // 此行直接报错, 因为case不允许重复

}

5. 在switch结构中, switch()括号中的要求具体是什么? case后面的要求又是什么?

Switch()中的要求是必须是变量或表达式, 数据类型是非long整数.字符串,枚举.

Case 后面要求必须是常量, 包括字面量和final修饰的量. 不允许表达式和变量

switch (n \* 2 / 5) {

}

## 每日一考\_day05

1. 列出变量的使用注意事项(至少6点)(尝试解释一下原因)

1) 必须要有数据类型和变量名

2) 先声明, 后使用

3) 变量有其作用范围

4) 变量有其数据范围

5) 变量必须初始化值才能读取.

6) 同一范围内变量不允许重复声明.

2. 变量分类有两种分法, 第一种是按数据类型来分,另外一种是按照声明位置来分, 每一种又各分为哪些种类型. 各有什么特点?

分类 :

1. 按照数据类型来分 :
2. 基本数据类型 : 内存区域中保存的是数据本身
3. 数值型 : 可以互相转换, 可以互相比大小

1) 整数 : byte, short, int, char, long

2) 浮点数 : float, double

2) 布尔型 : 都不行, 只允许两个常量值, true, false

2) 引用数据类型 : 内存区域中保存的是对象地址, 本质上就是内存的某个字节的编号.

1. 按照声明位置来分 :
2. 局部变量 : 声明在方法中

范围小, 寿命短

2) 成员变量 : 声明在类中方法外

范围大, 寿命长

3. for循环的结构是什么? 执行流程是如何?

for (初始语句A; 条件语句B; 迭代语句C) { // 和while很像

循环体语句D;

}

A B D C B D C B D C B D …… B

迭代语句C 是每次循环的开始语句

写程序, 打印一个倒直角三角形.

int n = ?;

for (int i = 0; i < n; i++) {

for (int j = 0; j < -i + n; j++) {

System.out.print(“\*”);

}

System.out.println();

}

## 循环总结 :

循环 : 在某些条件满足的情况下, 反复执行特定的代码的功能.

循环的组成 :

1) 初始化语句 : 作准备工作, 通常是int i = 0;

2) 循环条件 : 控制循环的生死, 如果条件为真就一直循环, 直接条件为假. i < n

3) 循环体 : 被多次执行的语句

4) 迭代语句 : 让i向n靠近, 每次i++, 使用循环趋于结束, 如果没有迭代, 循环不能自然结束

while : 适用于循环次数不确定的循环, 结束取决于外部条件

while (布尔条件) { // 如果布尔条件为假, 循环次数是0~N次

循环体; 在循环中控制布尔

}

do while : 适用于循环次数不确定的循环

do { // 如果布尔条件为假, 至少也要循环1次. 循环次数是1~N次.

循环体; 在循环体中控制布尔

} while (布尔条件);

for : 适用于循环次数确定的循环

for (初始语句int i = 0; 循环条件 i < n; 迭代语句i++) {

//循环体;

}

## 方法

方法 : java程序中某个功能的封装, 一个独立的功能体. 也称为函数

方法声明 :

修饰符 返回值类型 方法名(数据类型1 形式参数1, 数据类型2 形式参数2, ....) {

语句, 方法体

return 返回值;

}

返回值类型 : 方法的功能的最终体现 , 成果是什么, 数据类型是什么.

方法名 : 标识方法的.

形式参数 : 形式上需要的数据, 但是实际数据是多少不影响功能, 没有这个数据也不行. 在功能的完成时需要的数据

return 返回值, 最终的功能的结果数据要返回给调用者

调用者 : 使用这个方法的角色. 方法在调用时必须要由调用者传递实际参数.

方法 = 方法签名 + 方法体(body)

方法签名 : 方法的使用说明书. API

方法体 : 实际执行的代码.

注意 : 方法不可以嵌套, 必须是并行的.

方法调用(method invoke) : 是一个语句

方法名(实参列表); // 实参列表必须完全遵照形参的要求来写. 类型和个数及顺序要完全匹配.

方法的返回值? 方法调用本身

方法的返回值的接收只有一次机会, 错过了, 就错过了....

如果方法没有返回值, 必须使用void作为返回值类型

如果方法不需要参数, 参数列表空着, 但是()必须要有

* 注 意：
* 没有具体返回值的情况，返回值类型用关键字void表示，那么该函数中的return语句如果在最后一行可以省略不写。
* **定义方法时，方法的结果应该返回给调用者，交由调用者处理。**
* **方法中只能调用方法，不可以在方法内部定义方法。**
* 方法的返回值只有一次机会接收, 就是在调用时
* 如果在方法中又调用了方法本身, 称为递归调用

## 每日一考\_day06

1. 写出while循环和do while循环的区别, 三个循环分别在什么场景下使用?

While至少执行0次

Do while 至少执行1次

While和dowhile 适用于循环次数不确定的循环

For循环适用于循环次数确定的循环.

2. 简述break语句和continue语句的作用.

break 中断某个语句块的执行

|  |
| --- |
| public class BreakTest {    public static void main(String[] args) {  boolean flag = true;  l1 : {    System.out.println("hello");  if (flag) {  break l1;  }  System.out.println("world"); // 它不执行  }  System.out.println("three");  }  } |

continue 中断循环的某次循环, 进入下一次.

3. 声明定义一个方法的语法格式是什么? 解释每部分的作用.

修饰符 返回值类型 方法名(形参列表) {

方法体;

return 返回值;

}

方法 = 方法签名(方法的使用说明书) + 方法体(真正执行的)

修饰符 : 修饰public, private 访问控制修饰符, static 静态, synchronized, abstract, final….

返回值类型 : 方法执行完毕后的结果数据的类型 ( 输出 ), 返回值只能返回一个.

方法名 : 标识符

形参列表 : 形式参数, 形式上的数据, 虽然没有值, 在方法体中可以直接使用. 在方法调用时 会由调用者传入实际的值. 此方法如果需要外部传入的数据, 就可以使用参数.( 输入 )

方法体 : 由一个个的语句构成, 是将来在方法时真正的执行部分.

返回值 : 方法在结束后, 产生的结果值, 值必须被返回值类型兼容.

4. 方法的返回值是什么? 如何使用这个返回值?

方法调用本身 :

变量 = 方法名(实参列表);

5. 什么是方法重载, 为什么要方法重载?

同一个类中, 方法名相同, 参数不同. 参数不同体现在类型不同, 个数不同, 顺序不同.

功能相近的一组方法, 便于使用者记忆. 调用简单.

注意点 :

1)方法在调用时实参是通过值传递给形参数据, 本质上就是一个赋值操作. 值传递更安全

2)方法永远无法修改另一个方法的局部变量.

# 第三章 面向对象

学习面向对象三条主线 :

1. 类和类的成员的研究
2. 三大特性 : 封装Encapsulation, 继承Inheritance, 多态Polymorphism
3. 其他关键字 : this, package, import

完成需求时：

* + 先去找具有所需功能的对象来用。
  + 如果该对象不存在，那么创建一个具有所需功能的对象。
  + 这样简化开发并提高复用。

类(class)和对象(object)是面向对象的核心概念。

面向对象程序设计的重点是**类的设计**

类 : 对现实世界事物的描述, 是抽象的, 是概念上的.

用成员变量来描述事物的特征

用成员方法来描述事物的行为

对象 : 类的实体, 也称为实例.

类的本质也是一种新的数据类型, 也称为复合数据类型, Teacher类类型, 和int, char一样.

|  |
| --- |
| class Teacher1 {    // 成员(member)变量来描述事物的特征  String name; // 对象属性或实例变量  int age;// 对象属性或实例变量  String gender;// 对象属性或实例变量    // 成员(member)方法来描述事物的行为  // 对象方法或实例方法  public void lesson() { //对象方法  System.out.println(name + "老师在上课");// 成员互访  }    public String eat(String some) { //对象方法  System.out.println(name + "老师在吃" + some);// 成员互访  return "真好吃";  }    // 自我介绍, 返回一个字符串, 内容是我的所有属性值的拼接, 加上一些提醒会更清晰  public String say() { //对象方法  String str = "姓名 : " + name + ", 年龄 : " + age + ", 性别 : " + gender; // 成员互访  return str;  }    } |

|  |
| --- |
| public class TeacherTest {    public static void main(String[] args) {  // 类也备好了, 我们可以创建对象了.  Teacher t = new Teacher(); //依据类模板创建对象, 使用关键字new ;  t.name = "李明"; // 给对象的姓名赋值  t.age = 30;  t.gender = "男";    // 对象成员, 必须通过对象.成员的方法来访问  System.out.println(t.name); // 打印对象的姓名  System.out.println(t.age);  System.out.println(t.gender);    // 对象方法  t.lesson();  t.eat("包子"); // 方法调用 : 方法名(实参)  String s = t.say();  System.out.println(s);  }  } |

## 每日一考\_day07

1. 什么是类, 什么是对象, 什么是实例?

类 : 对现实世界事物的描述, 抽象定义

对象 : 类的实体, 也称为实例.

实例 : 就是对象, 对象就是实例, 是一回事. .,./

1. 类中有哪些成员? 各有什么作用? 成员意味着什么?

属性, 方法.

属性描述事物的特征, 通常都是名词.

方法描述事物的行为, 动作. 通常都是动词.

成员意味着同属一个类, 成员可以直接互访.

1. Teacher t = new Teacher(); 这个语句中的t究竟是什么?

t是一个Teacher类型的引用类型的变量, t变量内存空间中保存的是地址(右面对象在堆内存中的地址—某个字节的编号), t指向对象实体. 通过t操作数据, 就相当于操作对象.

4. Teacher t1 = new Teacher();

Teacher t3 = t1;

Teacher t2 = new Teacher();

t1 = t2;

t2 = t3;

以上代码中创建了几个对象? B, 因为有两个new操作

A. 1个 B.2个 C.3个 D.4个

5. 什么是垃圾对象? 垃圾对象会被立刻清理吗? 如何清理垃圾?

Teacher t1 = new Teacher();

t1 = null;

不再有引用指向的对象, 称为垃圾对象. GC内部测试对象是否可达, 如果不可达, 就是垃圾对象, GC会清理它, GC什么时候清理取决于GC内部.

如何清理? 把垃圾对象占用的所有内存空间标识为可用状态.

栈 : 正在运行中的方法

堆 : 类数据, 和对象数据, 常量池.

## 构造器(也称为构造方法) – constructor, 本质是一个方法, 但是有其特殊性

1. 方法名和类名一致, 唯一允许首字母大写的方法名
2. 没有返回值声明, 甚至连void也不允许
3. 不能被关键字 : static, final, synchronize, native, abstract等修饰
4. 不能像普通方法一样随意调用, 只能调用一次. 某个对象的生命周期中只能调用一次.

总结 :

* + **Java语言中，每个类都至少有一个构造器**
  + **默认构造器的修饰符与所属类的修饰符一致**
  + **一旦显式定义了构造器，则系统不再提供默认构造器**
  + **一个类可以创建多个重载的构造器**
  + **父类的构造器不可被子类继承**

构造器重载, 还可以连环调用, 但是一定要保证有一个构造器中是没有this(…). 防止无限递归.

this(...) 特殊调用必须是构造器中的第一行. 对其他构造器的调用必须要先于此构造器的特有代码, 最终要体现在继承中子类对象的初始化要晚一些.

* 在java中，this关键字比较难理解，它的作用和其词义很接近。
  + 它在方法内部使用，即这个方法所属对象的引用；谁调用this就是谁.
  + 它在构造器内部使用，表示该构造器正在初始化的对象。
* **注意：**
* **1.使用this()必须放在构造器的首行！**
* **2.使用this调用本类中其他的构造器，保证至少有一个构造器是不用this的。**

|  |
| --- |
| * class TeacherV0 { * // 成员变量(member), 成员可以使用修饰符修饰, 注意 : 局部变量无法使用访问控制修饰符 * // private表示私有的意思, 一旦私有, 外部无法访问, 只能在类内部访问 * // 这样的处理称为 "封装", 成员私有化. * private String name = "某老师"; // 属性如果没有显式赋值, 它的值就是默认值0 * private int age; // 缺省值是0 * private String gender = "未知"; * // 构造器重载, 还可以连环调用, 但是一定要保证有一个构造器中是没有this(). 防止无限递归. * // this(...) 特殊调用必须是构造器中的第一行. 对其他构造器的调用必须要先于此构造器的特有代码 * public TeacherV0() { // 无参, 调用简单, 功能单一, 有其特殊性, 创建对象简单, 无参构造器将来要为子类专供. * /\* * this.name = "佟刚"; * this.age = 40; * this.gender = "男"; * \*/ * this("佟刚", 40); // 对其他构造器的调用 * System.out.println("Teacher()..."); // 此构造器的特有代码 * } * public TeacherV0(String name, int age) { * /\* * this.name = name; * this.age = age; * this.gender = "男"; * \*/ * this(name, age, "男"); // 重载的构造器的调用, 必须使用this(); * System.out.println("Teacher(String, int)..."); * } * public TeacherV0(String name, int age, String gender) { // 全参, 功能强大, 调用复杂 * //this(); * this.name = name; * //this.age = age; * setAge(age); // 统一的参数合法性判断. * this.gender = gender; * System.out.println("Teacher(Stintg, int, String)..."); * } * /\* 这种方法也是允许存在的, 但是不要这样写, 容易引起误会.\*/ * public void TeacherV0(String name) { * System.out.println("我是普通方法, 不是构造方法, 我是捣乱的, 忽略我"); * } * // set方法有参无返回 * public void setName(String name) { // 间接设置属性的方法,称为set方法 * //name = name; // 变量访问的就近原则, 会导致左面的name不是属性, 而是局部变量. * this.name = name; * // this表示当前对象. 当局部变量参数和成员变量冲突时, 成员变量必须加上限定 * } * // get方法无参有返回 * public String getName() { // 间接获取name属性值的方法, 称为get方法 * return this.name; * } * // 使用方法间接给属性赋值的好处是 可以加上 逻辑判断, 使得数据得到保护... * public void setAge(int age) { * if (age < 0 || age > 150) { // 如果数据非法, 直接弹栈 * System.out.println("参数年龄太大"); * return; // 提前弹栈, 会导致this.age没有刷新值. 保持了原有的0 * } * this.age = age; * } * public int getAge() { * return this.age; * } * public void setGender(String gender) { * this.gender = gender; * } * public String getGender() { * return this.gender; * } * // 成员方法(member) * public void lesson() { * System.out.println(this.say() + "老师在上课"); // 成员互访 * } * public void eat(String some) { * System.out.println(name + "老师在吃" + some); * } * // 返回对象的详细信息字符串, 所有属性值的拼接 * public String say() { * return "姓名 : " + name + ", 年龄 : " + age + ", 性别 : " + gender; * } * } |

## 每日一考\_day08

1. 什么是封装? 如何封装?

成员私有化, 防止外部直接访问. 成员包括属性, 方法, 构造器, 内部类.

使用private修饰成员, 就可以封装.

2. 什么是构造器,作用是什么? 有什么特点(4点)

也称为构造方法, 在对象的创建过程中会由JVM调用它, 作用是对对象进行初始化工作

1) 方法名必须和类名一致, 唯一允许使用首字母大写的方法.

2) 不声明返回值类型, 甚至连void也没有.

3) 不能被关键字:abstract, static ,final, synchronized, native等 修饰

4) 不能像普通方法一样随意调用, 只能在对象创建时调用仅有的一次.

1. 创建一个对象的步骤(7步)

1) 在方法区要检查要创建的对象所属的类模板信息是否存在(类是否已经加载)

2) 如果不存在, 必须由类加载器加载类模板, 加载到方法区中, 永久保存它.

3) 如果存在, 则不加载, 保证类模板在方法区中只有一份.

4) 依据类模板中的所有属性定义信息, 在GC区中开辟空间. 获取到对象的地址.

5) 把此空间全部写0. 底层操作 : memset(内存地址, 0, 长度); 效果是所有属性拥有0

6) 检查属性是否有显式赋值, 如果有, 则执行显式赋值.

7) 执行构造方法, 进一步对属性进行赋值等初始化工作.

8) 把对象地址返回给创建者.

4 this关键字表示什么含义?

this表示的是对象, 当前对象

1. 如果是在构造器中, this表示正在创建中的对象, this.成员.
2. 如果是在构造器中, this(…) 表示对其他重载的构造器的调用.

要求 :

1) this(…) 必须在构造器的首行,

2) 一定要保证至少有一个构造器中没有this(…);

1. 如果是在方法中或属性的使用中, 表示的是正在访问这个属性或方法的对象的引用

Xxx ref1 = new Xxx();

ref1.method(); // 对象方法的调用必须要有对象.

Xxx ref2 = new Xxx();

ref2.method();

ref1.field = “值”;

class Xxx {

String field = this.xxx();

public void method() {

// ref1.method() 调用中, this 代表就是ref1

// ref2.method() 调用中, this 代表就是ref2

}

}

this 就是指调用者对象

对象 包括属性, 或者说是拥有属性

对象地址 : 对象在GC区中的第一个字节的编号. 也称为首地址

## 对象关联

对象关联 : 一个对象完全拥有另外一个类的对象.

关联目的 : 随时随时地方便地使用另一个对象.

如何关联 : 在前者的类中把另外一个类的对象作为属性即可. 属性是成员, 成员意味着它们之间可以直接互访

一旦把对象作为我的属性, 需要处理全参构造器, 因为是属性也要提供get/set方法, 改造say()方法

|  |
| --- |
| public class Teacher {    private String name;  private int age;  private String gender;  private Computer myComputer; // 对象关联    public Teacher() {}    public Teacher(String name, int age, String gender, Computer myComputer) {  this.name = name;  this.age = age;  this.gender = gender;  this.myComputer = myComputer;  }    public void setMyComputer(Computer myComputer) {  this.myComputer = myComputer;  }    public Computer getMyComputer() {  return myComputer;  }    public void setName(String name) {  this.name = name;  }    public String getName() {  return name;  }    public void setAge(int age) {  this.age = age;  }    public int getAge() {  return age;  }    public void setGender(String gender) {  this.gender = gender;  }    public String getGender() {  return gender;  }    public String say() {  return "姓名 : " + name + ", 年龄 : " + age + ", 性别 : " + gender + ", 我的电脑 : " + myComputer.say();  }    //public void lesson(Computer computer) {  public void lesson() {  System.out.println(name + "老师在使用电脑[" + myComputer.say() + "]上课");  }    public void film() {  System.out.println("使用电脑[" + myComputer.say() + "]看电影");  }    } |

## Package

包 : 用于管理功能类似的多个类.

package 包名.子包名.子子包名.子子子包名; // 这个语句总是在源文件中第一行

作用 : 告诉编译器, 当前源文件中的类, 编译后的.class文件放入指定的包目录中.

通常情况下包有4层

package 机构类型.机构名称.项目名称.模块名称; // 包名全部小写

package com.atguigu.javase.javabean;

一旦使用了package, 会带来两个麻烦

1) 编译时必须加上选项-d : javac -d 目标目录路径 源文件名

-d表示目标目录

常见写法 : javac 空格 -d 空格 . 空格 源文件名

2) 使用时, 如果是跨包使用本类, 在别的包的类中必须使用全限定名称来使用本类,

全限定名称就是 所有包名.类名 : 包名.子包名.子子包名.子子子名包.类名;

比如在测试类中, 要想使用本类. com.atguigu.javase.javabean.Teacher

练习 : 把Person类和Phone类都打包在com.atguigu.javase.javabean包中

把PersonTest类打包在com.atguigu.javase.test包中

在测试类中, 使用全限定类名来使用Person类和Phone类, 编译所有程序, 运行测试.

# 第三章 2数组

数组 : 一组相同任意类型的数据的组合, 便于统一管理.

数组 : 由多个元素组成, 每个元素相当于是数组对象的属性

数组属于引用类型, 数组的数据则是对象.

数组声明 :

元素数据类型[] 数组名;

数组创建 :

数组名 = new 元素数据类型[元素个数];

|  |
| --- |
| public class ArrayTest {    public static void main(String[] args) {  // 声明 : 元素数据类型[] 数组名;  int[] arr; // 此时没有数组对象存在的.  // 创建 : new 元素数据类型[元素个数];  arr = new int[5]; // 在GC区中创建一个元素有5个的int型数组. 数组对象创建好以后, 所有元素都有缺省值0  // 数组元素的定位(访问), 通过首地址(数组名)[偏移量(也称为下标, 索引, 脚标)]  // 每个元素都是一个小变量.  arr[2] = 4;  arr[0] = 3;  arr[arr[0]] = 1;  arr[arr[2]] = 5;  // 分析每个元素的值  System.out.println(arr[0]); // 3  System.out.println(arr[1]); // 0  System.out.println(arr[2]); // 4  System.out.println(arr[3]); // 1  System.out.println(arr[4]); // 5  }  } |

## 每日一考\_day09

1. 类模板信息保存在内存中的什么位置, 里面包含了哪些信息, 类模板信息的特点和作用是什么?

堆内存中的永久区(方法区)

类模板中包含的信息有 :

1) 属性的定义信息(修饰符, 数据类型, 属性名, 显示值)

2) 所有方法的代码. (不是在运行中状态), 如果要运行, 必须把它压入栈中执行.

3) 构造器, 类名, 父类, 接口, 注解……

2. 数组是什么? 什么类型的数据可以创建数组?

一组相同任意类型的数据的组合, 便于统一管理.

任意数据类型, 包括基本型和引用型.

3. 如何声明并创建数组? 有几种方式(3)?

声明 : 元素数据类型[] 数组名;

创建 : new 元素数据类型[长度];

动态方式声明和创建 : 元素数据类型[] 数组名 = new 元素数据类型[长度];

静态方式1 : 适用于数据量小, 元素值确定的情况.

元素数据类型[] 数组名 = new 元素数据类型[] {元素值1, 元素值2, 元素值3……};

静态方式2 : // 使用受限 : 声明和初始化必须在同一条语句上才可以这样.

元素数据类型[] 数组名 = {元素值1, 元素值2, 元素值3……};

int n = Integer.parseInt(args[0]); // 命令行参数的第一个字符串

4. 判断:

1) 数组的长度可以随意变化. F

2) 数组的元素类型和数组类型是一回事. F

3) 数组的访问是通过数组名.下标实现的. F

4) 数组的访问方式是数组名(首地址) + 偏移算出来的. T. arr[index].

5) 数组的元素类型可以随意变化. F

5. Student[] stuArr = new Student[3]; // 1 : 1 object

stuArr[0] = new Student(20, "小明","男");

stuArr[1] = new Student(30, "小丽","女");

for (int i = 0; i < stuArr.length; i++) {

System.out.println(stuArr[i].say());

}

在1位置处有几个对象被创建, 什么对象? 1, Student[]

以上的代码有没有问题? 如何改进? Yes,

1)

for (int i = 0; i < stuArr.length; i++) {

if (stuArr[i] != null) {

System.out.println(stuArr[i].say());

} else {

System.out.println(stuArr[i]);

}

}

2)

stuArr[2] = new Student(22, “Mike”, “male”);

# IDEA快捷使用

不需要特别保存, 也不用编译.

Ctrl + p 提醒方法调用的参数列表

Alt + 回车, 快速帮助 比如定义变量接收数据

Shift + 回车, 任意位置进入新行

Ctrl + d 复制当前行

Ctrl + y 删除当前行

在dubug模式下

F7 是进入细节

Alt+shift+F7 进入最细节

F8 一行一行的执行

Shift+F8 当前方法执行返回, 如果后面有断点, 会直接跳到断点处

Alt+F9 直接运行到下一断点或结束.

写普通类的时候, 先写属性

Alt + insert -> constructor -> alt+n 创建无参构造

Alt + insert -> constructor -> ctrl + a 回车 创建全参构造

Alt + insert -> getter and setter -> ctrl+a 回车, 生成所有get/set方法

# 第四章 面向对象高级1

## 每日一考\_day13

1. 什么是继承? 为什么要继承? 如何继承?

从现有类创建子类, 现有类称为父类, 基类, 超类.

为什么要继承 :

1. 代码复用, 在子类中不需要重复一些成员的定义.
2. 让类与类之间更加的具有分类一致性.

如何继承 ?

Class 子类 extends父类 {

}

2. 子类能继承父类的私有成员吗? 如何处理?

子类会继承父类的所有成员(构造器除外), 包括私有成员.

虽然能继承父类的私有成员, 但是不可以直接访问. 必须通过从父类继承的公共的

方法间接访问.

3. 为什么父类又叫基类或超类?

基类 : 子类在父类的基础之上进行扩展(extends)

超类 : 因为在子类中使用关键字super标识从父类继承的成员.

4. 什么是方法覆盖? 方法覆盖有什么条件?

子类根据需求对父类继承的方法进行重写, 改造, 重置. 对父类的方法不满.

要求 :

1. 方法的签名一致. 返回值类型, 方法名, 参数列表(参数类型, 顺序, 个数)
2. 子类的覆盖方法的访问控制修饰符要大于等于父类的.

5. 如果A类被B类继承,B类又被C类继承, 在所有类中都有一个方法test(),

创建C类对象,调用test()方法,执行的是哪个类的方法? 在C类中有几个test()方法?

执行的是C类中的test()方法.

1个方法 : 从测试类的角度去看, C类中只有一个test方法.

2个方法 : 从C类的内部来看, C类中有2个test方法, 一个是super.test(), 一个是this.test()

3个方法 : 从继承的概念来看. C类继承了B类中的2个test, 自已也有一个, 所以3个.

## 关于构造器 :

1. 所有类都必须有构造器

如果在类中没有提供任何构造器, 编译器会自动添加一个缺省构造器

如果在类中提供了构造器, 编译器就不添加构造器了.

1. 所有类的构造器中都必须要有先对父类构造的调用. 如何体现 “先” 强制第一行
2. 所有类的构造器中的第一行默认是super()
3. 所有类构造器的第一行要么是this(…), 要么是super(…)

super(…) 作用是直接调用父类构造

this(…) 作用是调用其他重载构造器, 其他构造器也是必须要满足这2点, 效果是间接调用了父类构造.

思考：

1).为什么super(…)和this(…)调用语句不能同时在一个构造器中出现？

这两个语句都要求在第一行.

Super()如果和this() 同时出现, 会导致子类中两个调用到父类构造, 出现混乱.

2).为什么super(…)或this(…)调用语句只能作为构造器中的第一句出现？

所有类的构造器中都必须要有先对父类构造的调用. 如何体现 “先” 强制第一行

## 每日一考\_day14

1. 什么是多态? 如何在java中体现多态,如何使用?

子类对象的多种父类形态, 父类引用指向的多种不同子类对象.

多态引用 : 子类对象赋值于父类类型的引用变量

内在逻辑 : 把子类对象”看作是”, “当成是” 父类对象来使用. 因为子类对象中包含了所有父类成员.

2. 什么是多态副作用?

当多态引用时, 子类对象的个性抹杀(隐藏起来), 特有成员无法访问.

3. 什么是虚方法调用? 有什么特点?

虚拟方法调用 : 通过多态引用调用覆盖方法.

class A {

void t(){} // 在父类中此方法的唯一作用是骗过编译器, 运行时不执行

}

class B extends A {

void t() {} // 子类中此方法在运行时真正执行的

}

// invokevirtual – 动态绑定.

A a = new B();

t(); // 编译时, 只知道a是A类型, 只能到A类型中找这个方法.

// 运行时, JVM会动态定位对象实体(实体由new操作符)

4. instanceof 操作的符的作用是什么?

引用 intstanceof 类型

判断引用指向的对象实体是否是右侧指定类型的对象

ref instanceof Object 永远为真

// 在造型时, 使用它作判断依据, 造型有风险, 必须先判断

// 如果不兼容的类型之间造型会引发异常 : ClassCastException

if (a instanceof A) {

A a2 = (A)a;

}

1. 描述equals方法和hashCode方法的含义和作用.

equals方法的作用是判断两个对象的内容是否相等

equals(b) 如果a对象和b对象内容相等, 返回true, 否则返回false

a引用隶属的类中必须重写此方法.

hashCode方法是返回当前调用者对象的哈希码 :

散列码 : 对象要在内存中尽量散列.

特征码 : 内容相同的对象特征码要一致.

如果两个对象的equals为true, 说明两个对象的内容相等, 内容相等,特征码必须相等 一定能做到

如果两个对象的equals为false, 说明两个对象的内容不等,内容不等, 散列码必须不等

不一定能做到

是否要重写这两个方法, 取决于是否关心它的所有对象之间能比较相等.

# 第五章 面向对象高级2

## 每日一考\_day16

1. 什么成员应该声明成静态的, 什么成员应该声明为非静态的?

成员应该隶属于类还是隶属于对象, 如果隶属于类(或者说是可以被所有对象共享) 这个的成员, 声明静态的.

成员必须要依赖于对象的存在才能意义, 这样的成员应该声明为非静态成员.

2. 静态环境中可以直接访问非静态成员吗? 非静态环境中可以访问静态成员吗? 各自的原因是什么?

静态环境中访问非静态成员 : 不可以, 原因是静态环境不是通过对象来调用的, 而非静态成员必须要依赖对象的存在, 可以通过创建新对象, 再通过对象来间接访问.

非静态环境中访问静态成员 : 可以的, 原因是非静态环境中一定是已经有了对象, 对象都有了, 类也早有了.

3. 写出饿汉式单例代码

public class Single {

private int id;

private String desc;

// 1) 封装构造器

private Single() {}

//2) 声明私有的静态的指向本类类型的对象的引用, 并同时创建唯一对象

private static Single only = new Single();

//3) 声明公有的静态方法, 用于获取唯一对象的地址

public static Single getInstance() {

return only;

}

}

加载类 : 分析类,及继承体系. 由父到子加载类, 先分析有哪些静态成员, 有哪些非静态成员. 再加载所有非静态成员的定义信息, 再加载静态成员的定义信息. 执行静态语句块.

4. final可以修饰什么? 被修饰的元素具有什么特点?

final可以修饰类, 方法, 变量

final修饰类表示这个类是终极类, 不允许子类扩展

final修饰方法表示这个方法是终极方法, 被子类继承后, 不允许再被子类重写,覆盖

final修饰变量表示这个量是最终量(final量), 它的特点 : 只能必须赋值一次.

5. 抽象类是什么? 和具体类的区别是什么?

具体类 : 某种事物的抽象定义

抽象类 : 某类不同种事物的抽象定义, 抽象可以包含抽象方法, 正因为如此, 不能创建对象

6. 判断 :

1) 抽象类中必须包含抽象方法 f

2) 抽象类中不能包含普通方法 f

3) 抽象类中可以包含抽象方法 t

4) 抽象类不能创建对象, 所以可以省略构造器 f

5) 具体类最多允许包含1个抽象方法 f

6) 抽象类主要用于被具体子类继承,具体子类可以不必理会抽象父类中的抽象方法 f

7) 具体类中如果包含抽象方法, 编译出错 t

8) 一个类中如果包含抽象方法, 这个类必须是抽象类 t

9) 抽象类中必须包含属性,构造器和普通方法和抽象方法. f

10) 抽象类中可以不包含抽象方法,具体子类必须实现全部的父类的抽象方法. t

7. 什么是接口? 作用是什么?

接口 : 不同类的不同种事物的共同的行为的抽象定义

作用 : 用于表达某种能力, 或表达某种标准规范.

## 每日一考\_day17

1. 判断:

1 类可以继承接口, F

2 接口也可以继承类, F

3 接口可以继承接口, 只能单继承 F

4 接口可以继承接口, 并且可以多继承 T

5 类可以实现接口, 并对其中的抽象方法不予处理 F

6 类可以实现接口, 但是只能实现一个接口 F

7 具体类可以实现接口, 并对其中的抽象方法不予处理 F

8 类可以实现接口, 并可以实现多个接口 T

9 具体类可以实现接口, 并要实现其接口中的方法 T

10具体类可以实现接口, 并可以实现多个接口, 并要实现所有接口中的所有方法. T

2. 抽象类和接口的比较

比较项目 抽象类 接口

定义 某类不同种事物抽象定义 不同类不同种事物共同行为抽象定义

组成 属性,方法,构造器,语句块,抽象方法 全局常量,公共抽象方法

如何使用 被子类继承 被子类实现(implements)

两者关系 抽象类可以实现接口 被实现

常见设计模式 模板 代理, 工厂模式

创建对象 通过多态使用子类对象 通过多态指向子类对象

局限性 单继承 没有

实际应用 模板 能力, 标准规范

选择 优先选择接口

3. 描述什么是代理设计模式. 在什么样的场景中使用

把代理对象当成被代理对象来使用

1. 使用者无法获取被代理类和对象
2. 对被代理类的方法升级, 但是不能修改被代理类.

4. 如何在其他测试类中创建内部类对象?

class Outer{

class Inner{}

}

main() {

Outer.Inner oi = (new Outer()).new Inner();

}

5. 什么是匿名内部类? 如何使用?

声明在方法中, 没有类名的类. 匿名内部类在声明时就必须创建对象

多态引用 = new 父类|接口() {

类体部分相当于 父类或接口的实现子类的类体

};

应用场景 : 需要临时用一个实现某接口的对象.

## 每日一考\_day18

1 定义一个注解MyAnn, 包含1个String型value属性, 并且此注解只能修饰类和属性, 此注解也可以通过反射处理.

@Target({ElementType.TYPE, ElementType.FIELD})

@Retention(RetentionPolicy.RUNTIME)

public @interface MyAnn {

public String value(); // 公共抽象方法

}

2 异常按照处理方式分为几种? 各包含哪些类?

异常 ：程序在运行中出现的非正常状况，会导致程序崩溃

分为2种

* 1. 受检异常(checked) 在程序中必须对其进行处理的异常, 不处理的话, 编译器出错, 也称为编译时异常

Exception及其子类(RuntimeExceptoin除外), 程度是必须要引起调用者的注意.

2）非受检异常(unchecked) 在程序中不是必须对其进行处理的异常, 不处理的话, 编译器不报错, 但是在运行时会出问题. 也称为运行时异常

包括 Error及其子类 太过严重

包括RuntimeException及其子类 太过常见和轻微

3 判断

1) 非受检异常就是必须不要处理的异常 F

2) 受检异常就是可以处理的异常 F

3) 非受检异常是不必须处理的异常 T

4) 受检异常可以对其忽略 F

5) 无论是什么异常,都必须对其进行处理 F

6) 只有受检异常会引起程序中断 F

7) 受检异常是必须对其进行处理的异常 T

8) 只有非受检异常会引起程序中断 F

9) 异常处理只适用于受检异常 F

10) 异常处理适用于所有异常, 包括Error T

4 异常的处理有几种方式, 各是什么, 如何处理?

1) 捕获 : 异常一旦在方法中捕获, 就不会自动地再向调用者抛出.

在方法中使用try catch finally结构

try {

可能抛出异常的语句

} catch (可能的异常类型1 引用) {

处理异常类型1的异常状况

} catch (可能的异常类型2 引用) {

处理异常类型2的异常状况

} ….. {

} finally {

无论前面(try catch)发生什么, 我都要执行.

}

2) 抛出 : 异常一旦抛出, 导致当前方法提前出栈, 带着具有破坏性的异常给调用者, 引起调用者的注意, 是一种特别的消息传递机制.

在方法的声明中使用throws 可能的异常类型的列表, 作用是警告调用者.

在方法体中在条件满足的情况下, 使用throw真的抛出异常对象.

5方法覆盖条件中对于异常的描述是什么?

子类方法抛出的受检异常的类型要小于等于父类方法抛出的受检异常类型.