day01笔记:

是

注释:

- 分为三种:

单行注释//

多行注释/**/

关键字:被Java语言赋予了特殊含义的单词

特点:

1.组成关键字的字母全是小写;

2.常见的代码编辑器对关键字的代码有特殊的颜色标记.

常量与变量

- 常量:在程序执行过程中,其值不可改变的量

分类:

1.字符串 双引号扩住

2.整数常量

3.小数常量

4.字符常量 单引号扩住

5.布尔常量 分为true与false两种

6.空常量

- 变量:是内存中的一小块区域,在程序的执行过程中,其值可以在一定范围内发生改变

1.对区域要有限定(用数据类型进行限定)

2.必须对区域有个名称(即变量名)

3.区域内必须有内容(初始化值)

4.定义格式:数据类型 变量名 = 初始化值

- 数据类型分为:

1.基本数据类型

A.整数

2.引用数据类型

- 变量只在它所属的范围有效(它所在的大括号)

- 变量未赋值不能使用

- 一行上可以定义多个变量,但是不建议

-

1. * 强制转换类型格式:目标类型 变量名 = (目标类型) (被转换的变量或表达式)
2. * 建议:数据做运算,结果应是什么类型,就有什么类型接受,不要随意转换类型,否则会损失精度!

##

---

day02笔记:

- Day02

-

* eclipse的基本使用

1. 解压文件既可直接使用
1. 选择工作空间，一般放到eclipse文件夹内
1. 软件面包分为三大板块：1.项目列表；2.编写代码板块；3.控制台
2. 进入软件，首先新建项目 File-new-project-java project-项目名-Finish
1. 项目建好后创建包，src中创建包，切换视图为java 右击src-new-package-包名-Finish
2. 在包中内创建类
3. 类中写代码，保存后自动编译

---

* 基本配置

1. 修改字体：window-Preferences-Basic-Test Font-Edit-修改字体大小-确定-Apply-ok
2. 单独打开控制台：window-show view-Console
3. 窗口乱了怎么重置：window-perspective-REST
4. 程序乱了怎么办：编程页面右键-Source-Format（shift+ctri+f）

---

* 运算符

对常量和变量进行操作的符合称为运算符

---

* 表达式

* 用运算符把常量或者变量连接起来的符合java语法的式子就可以称为表达式。
* 不同运算符连接的式子体现的是不同类型的表达式。举例：定义两个Int类型的变量

a，b，做加法即是（a+b）

---

* 常用运算符
    * 算数运算符
    * 赋值运算符
    * 关系运算符
    * 逻辑运算符
    * 三元运算符

---

* 常用算数运算符：*，+，—，/,%,++,--

    * 算数运算符取余和除法的区别 %：取余运算符。得到的是两个相除数据的余数。/:除法运算符。得到是两个相除数据的商。使用场景： %：判断两个数据是否整除。
    * 整数相除为整数，除非有浮点型参与运算

---

* 字符和字符串参与加法操作

    *字符参与运算：其对应的字符数值来操作
  * 'a' = 97 依次往后递增
  * 'A'= 65　依次往后递增
  * '0'= 48　依次往后递增到‘9’

* ++，--的用法要点

  * `++`: 自增, 用于对变量加一
* `--`: 自减, 用于对变量减一
* 运算符放在变量前和变量后的区别:
  * 单独使用时(即只有`i++;`或`++i;`), 没有区别, 都是自增
  * 参与其他操作时:
    * `int a = i++`:
      * 先使用i本身的值作为i++表达式的值
      * 然后i再自增(i自增后不影响之前i++表达式的值)
      * 最后将等号右边的值赋值给左边a

* `int a = ++i`:
* i先自增
* 然后将自增后的值作为++i表达式的值
* 最后将等号右边的值赋值给左边
* `--`同理

---

* 赋值运算

---
* 关系运算

---

* 逻辑运算

* &：与，两边同为true则为true，否则false
* |：或，有false则true
* ！：非，取反
* ^ :异或，不同则为true
* &&：短路与，具有短路效果，前面为false时则后面不执行
* ||：短路或，具有短路效果，前面为true时后面不执行

---
* 三元运算符

* 格式：（关系表达式）？表达式1：表达式2
* 执行流程：1.先计算关系表达式的布尔值；2.如果true，则表达式1是结果，否则表达式2结果，具有强制转换功能

* 键盘导入

* A:导包(位置放到class定义的上面)
import java.util.Scanner;
* B:创建对象
Scanner sc = new Scanner(System.in);
* C:接收数据

int x = sc.nextInt();

- # Day03：
- ## -流程控制语句

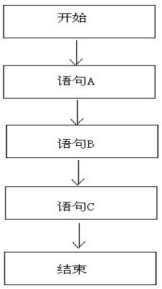
在一个程序执行过程中，各条语句的执行顺序对程序的结果是有直接影响的。也就是说程序的流程对运行结果有直接影响。

- **流程控制语句分类**

1. 顺序结构
2. 选择结构
3. 循环结构

- 顺序结构

是程序中最简单最基本的流程控制，没有特定的语法结构，按照代码的先后顺序，依次执行，程序中大多数的代码都是这样执行的



- 选择结构
  - 也被称为分支结构
  - 选择结构有特定的语法格式，我们必须按照它的基本格式来编写代码。
  - Java语言提供了两种选择结构语句
    - if语句
    - switch语句

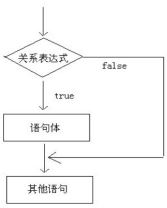
  - **if语句**
    - 三种格式

1. 一种情况判断　if（关系表达式）{

语句体

}

判断表达式结果，true则执行语句体，false则不执行。



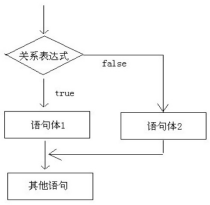
2. 两种情况判断　if（关系表达式）{

语句体1

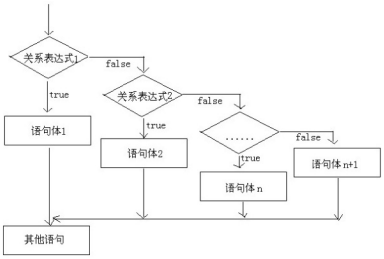
}else{

语句体2

}



3. 多种情况判断 if(关系表达式1) {
   语句体1;
}else if(关系表达式2) {
   语句体2;
}
   …
else {
   语句体n+1;
}



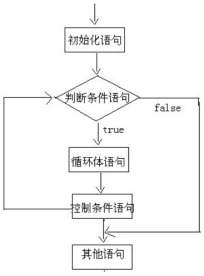
- switch语句
  - 格式：switch（表达式）{

case 值1；（值用来给表达式进行匹配的）
语句题1
break；（中断的意思）
.....
default：（所有值都不匹配）
语句体n+1;
break；

  - 执行流程：
    - A首先计算表达式值；
    - 拿着计算出的值，依次和case后面的值进行比较，一旦有对应的就执行相应语句体，执行过程中遇到break就结束；
    - 如果都不匹配，则执行语句体n+1

- 循环语句 **初始化语句在循环中只执行一次**
  - for循环，while循环，do while循环
    - for循环执行流程：

A:执行初始化语句
B:执行判断条件语句，看其结果是true还是false
如果是false，循环结束。
如果是true，继续执行。
C:执行循环体语句
D:执行控制条件语句
E:回到B继续



- for循环格式：
  - for(初始化语句;判断条件语句;控制条件语句) {

循环体语句;
}

- while循环格式：
  - 基本格式

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

- 基本格式

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

- for while do while三种循环的区别:
  - for与while循环如果条件不满足,则一次都不执行循环体
  - do while至少执行一次

- for 与 while的区别
  - for循环的初始化语句的作用域: for的大括号内
  - 但是for循环中的初始化语句可以定义到括号外面来,这样就可以使用 ;格式: 初始化语句

```
for(;关系表达式;控制语句){
循环体
}
```

- for循环
  - 适用于知道循环次数的场景
- while循环
  - 适用于不知道循环次数的场景
  - 至少执行0次循环体

- do...while循环
  - 适用于不知道循环次数的场景
  - 至少执行1次循环体
- 循环控制语句
  - break
    - 适用场景: switch语句, 循环
    - 作用: 结束switch语句或所在循环
  - continue
    - 适用场景: 循环
    - 作用: 结束本次循环, 继续下次循环
    - **switch语句的case穿透**
      - **如果switch的case语句中没有break, 则会继续执行下一个case的代码, 这称为case穿透**
      - **case穿透有时会导致代码出现问题, 但有时我们也利用case穿透简化代码**

- **if语句不写大括号的格式**
  - **当if语句不写大括号时, 只有if语句后的第一条语句算作if语句的语句体, 其余不算**
  - **while语句, for语句不写大括号的格式**
  - **当语句不写大括号时, 只有语句后的第一条语句算作语句的语句体, 其余不算**

- **for循环哪些语句可以省略:**
  - **如果变量在for循环外已经声明, 则初始化语句可以省略** **for(;判断条件语句;控制条件语句){**

    循环体

    }
  - **判断条件语句可以省略, 相当于不停止**
  - **控制条件语句可以省略, 相当于不改变for循环中的变量**

```
// 将for循环修改为死循环
for (;;) {
    // 死循环
}

for (;;);  // 连循环体都没有的死循环
```

Day05: 方法

- 方法的定义格式:
  - 方法:
    - 完成特定功能的代码块

- 作用

提高代码复用性和可维护性

定义格式:

修饰符: 目前记住public static

返回值类型: 用于限定返回值的数据类型

方法名: 方法的名字, 便于我们调用

参数类型: 用于限定调用方法时传入数据的类型

参数名: 用于接收调用方法时传入数据的变量,用于告诉方法的调用者, 调用该方法时需要传入何种数据

方法体: 完成我们需要的功能的代码

return语句: 结束方法, 并把返回值传递给调用者

  - 调用方法的3种格式

单独调用: sum(10, 20); 没有返回值时单独调用

输出调用: System.out.println(sum(10, 20)); 有返回值时用

赋值调用: int result = sum(10, 20);有返回值时用

  - 两个明确:
    - 明确返回值类型
    - 明确参数列表

  - 方法重载:
    - 概念: Overload, 在同一个类中, 多个方法有相同的方法名, 方法参数列表不同
    - 特点:

在同一个类中方法名相同

方法的参数列表不同

参数的个数不同

参数的数据类型不同(包括顺序不同)

    - 重载与以下几点无关

与方法的返回值类型无关

和参数的参数名无关

和方法体无关

- 方法的形参类型
    - 方法形参是基本数据类型

结论: 形参的值的改变不会影响实际参数

为什么? 形参是定义在调用方法中的一个单独变量, 实际参数是定义在main方法中的另一个单独变量, 两者内存区域不同, 所以互不影响

    - 方法形参是引用数据类型

结论:

- 如果形参是修改自身保存的引用, 不会影响实际参数
- 原因: 因为形参是调用方法中的一个单独变量, 实际参数是定义在main方法中的另一个单独变量, 形参改变其保存的引用指向, 并不影响实际参数的引用指向
- 如果形参是通过引用修改堆内存中保存的数据, 会影响实际参数获

取的值

- 原因: 因为形参是调用方法中的一个单独变量, 实际参数是定义在main方法中的另一个单独变量, 形参通过引用修改的是堆内存中的数据, 而实际参数也是指向该堆内存中的数据, 所以形参修改后, 实际参数获取的值也会改变

Day06: Eclipse断点调试 基础语法练习

- 断点:
  - breakpoint, 标记程序在这里暂停
- 断点调试的作用:
  - 可以在程序运行中查看程序的执行流程
  - 调试程序, 如查看变量的值
- 如何加断点:
  - 在行号左边双击
- 创建/取消断点
  - 单个断点: 双击

    - 多个断点: 在debug视图中的breakpoints窗口中点击Remove all breakpoints按钮
- 执行debug:
    - 右键 - debug as
- 窗口功能介绍
    - 开启debug窗口方式
        - 点击右上角创建窗口按钮
        - 右键debug as出现
    - 窗口区域
        - 代码区域: 看程序的执行流程
        - Debug区域: 看程序的执行流程
        - Variables: 看变量的值
        - Console: 看控制台输出
        - breakpoints: 查看所有断点
- 快捷键
    - F5 进入方法
    - F6 执行下一行代码
    - F7 跳出方法
    - F8 调到下一个断点

- 查找元素第一次出现的索引
- 分析
    - Scanner录入数组数据
    - 实现查找方法
        - 遍历数组, 依次比较元素, 如果相等, 就把该处索引返回
    - 异常情况处理

- 查不到返回-1

- 方法一:
  - public class Test {

```
    public static void main(String[] args) {
        // 定义一个数组
        int[] arr = {1,2,3,4,5};
        // 测试方法
        System.out.println(getIndex(arr, 3));  // 2,
找到了
        System.out.println(getIndex(arr, 10)); // -1,
找不到
    }

    // 查找指定元素
    public static int getIndex(int[] arr, int value) {
        // 遍历数组
        for (int i = 0; i < arr.length; i++) {
            // 获取数组中的每个元素值, 与传入的值
进行比较是否相等
            if (arr[i] == value) {
                // 如果相等, 则说明找到了, 返回当前
索引
```

```
                return i;
            }
        }
        // 如果能执行到这里, 说明没有找到, 返回负数
        return -1;
    }
```

- 方法二:

```
public class Test {

    public static void main(String[] args) {
        // 定义一个数组
        int[] arr = {1,2,3,4,5};
        // 测试方法
        System.out.println(getIndex(arr, 3));  // 2, 找到了
        System.out.println(getIndex(arr, 10)); // -1, 找不到
    }

    // 查找指定元素
    public static int getIndex(int[] arr, int value) {
        // 定义一个变量接收索引
```

```
                int index = -1;
    // 遍历数组
    for (int i = 0; i < arr.length; i++) {
        // 获取数组中的每个元素值, 与传入的值进行比较是否相等
        if (arr[i] == value) {
            // 如果相等, 则说明找到了, 将当前索引赋值给index
          index = i;
          // 结束循环
          break;
        }
    }
    // 如果能执行到这里, 说明没有找到, 返回index
    return  index;
}
```