1. 程序的基本功能是处理数据

程序用变量来表示数据；

程序中必须先定义变量才能使用；

定义变量是指设定变量的数据类型和变量的名字，Java语言要求变量遵循先定义，再初始化，然后使用的规则。

变量的使用有一个作用域的问题，作用域是指它的存在范围，只有在这个范围内，程序代码才能访问它。

其次，作用域决定了变量的生命周期。变量的生命周期是指从一个变量被创建并分配内存空间开始，到这个变

量被销毁并清除其所占用内存空间的过程。当一个变量被定义时，它的作用域就被确定了。按照作用域的不同，

变量可分为以下类型：

. 成员变量：在类中声明，它的作用域是整个类；

. 局部变量：在一个方法的内部或方法的一个代码块的内部声明。如果在一个方法内部声明，它的作用域是整个

方法；如果在一个方法的某个代码块的内部声明，它的作用域是这个代码块。代码块是指位于一对

大括号"{}"以内的代码。

. 方法参数：方法或者构造方法的参数，它的作用域是整个方法或者构造方法。

. 异常处理参数：和方法参数很相似，差别在于前者是传递参数给异常处理代码块，而后者是传递参数给方法或者

构造方法。异常处理参数是指catch(Exception e)语句中的异常参数"e"，它的作用域是紧跟着

catch(Exception e)语句后的代码块。

目的：1. 局部变量

1) 定义在方法的内部或方法的一个代码块的内部或方法参数列表；

public void method1(int c) { //c局部变量

int a = 0; //局部变量，作用域为整个method01方法；

{

int b = 0; //局部变量，作用域为所处的代码块；

b = a;

}

b = 20; //编译出错，b不能被访问；

}

2) 局部变量没有默认值，使用之前必须先初始化；

按照课件上讲解的内容进行介绍；

3) 生命周期(创建分配内存空间到销毁清除内存，空间的过程)

public class Sample {

public int add() {

int addResult = 1;

addResult = addResult+2;

return addResult;

}

public int subtract() {

int subResult = 1;

subResult = subResult-2;

return subResult;

}

public static void main(String[] args) {

Sample s = new Sample();

s.add();//开始局部变量addResult的生命周期，位于Java栈区；

结束局部变量addResult的生命周期，退回到main方法；

int a=s.substract();//开始局部变量subResult的生命周期，位于Java栈区；

结束局部变量subResult的生命周期，退回到main方法；

}

}

调用Sample实例的add方法，开始局部变量addResult的生命周期，addResult位于Java栈区。

执行完毕Sample实例的add方法，结束局部变量addResult的生命周期，退回到main方法；

画内存图分析其中的过程；

2. 实例变量/成员变量

1) 在类中声明，它的作用域是整个类；

class Test {

private int n1=0;

private int n2=0;

public int add() {

int result = n2 + n2;

return result;

}

}

2) 实例变量有默认值，使用之前可无须初始化；

实例变量可以在当前类的所有非静态代码块/方法中使用

按照课件上讲解的内容进行介绍；

int byte short long 默认赋值 0

double float 默认赋值0.0

boolean 默认赋值 false

char 默认赋值 ''空字符

引用数据类型 默认赋值为 null

3) 生命周期(创建分配内存空间到销毁清除内存空间的过程)

class Test {

private int n1=0;

private int n2=0;

public int add() {

int result = n2 + n2;

n1 = n1+1;

n2 = n2+2;

return result;

}

public static void main(String[] args) {

Test t1 = new Test();

Test t2 = new Test();

t1.add();

t1.add();

t2.add();

}

}

创建Test实例，开始实例变量n1,n2的生命周期，n1,n2位于堆区。

执行完毕Test类的main方法，结束Test实例及它的实例变量n1，n2的生命周期，卸载Sample类，Java虚拟机运行结束。

画内存图分析其中的过程；

4）判断举例

class Hello{

int age = 10;

public static void main(String[] args){

System.out.println(age);//1

}

public void show(){

System.out.println(age);//2

}

public static void eat(){

System.out.println(age);//3

}

}

在1 2 3 哪个位置访问是正确的？为什么？

1号位置不可以访问，main方法为static方法不能访问非静态的成员变量。

2号位置可以访问，普通方法可以访问普通的成员变量。

3 号位置不可以访问，eat是静态方法。

3. 操作符

程序的基本功能就是处理数据，程序用变量来表示数据。任何编程语言都有自己的操作符，Java语言也不例外。操作符

能与相应类型的数据组成表达式，来完成相应的运算。

一般情况下，不用去刻意记住操作符的优先级，当不能确定操作符的执行顺序时，可以使用圆括号来显示指定运算顺序。

优先级、可读性：

1.不要把一个表达式写的过于复杂，如果一个表达式过于复杂，则把它分成几步来完成

2.不要过多的依赖运算符的优先级来控制表达式的执行顺序，这样可读性太差，尽量使用()来控制表达式的执行顺序

1) 赋值操作符：

= : 赋值 把等号右边的值赋值给等号左边的变量 int x=0,i=1,j=1;

+= : 加等 a+=b 等价于 a = a + b;

-= : 减等 a-=b 等价于 a = a - b;

\*= : 乘等 a\*=b 等价于 a = a \* b;

这里的"\*="由操作符"\*"和"="复合而成，它等价于 a=a\*b;

这种复合操作符能使程序变得更加简洁。

/= : 除等 a/=b 等价于 a = a / b;

%= : 余等 a%=b 等价于 a = a% b;

<<= : 左移等 a<<=b 等价于 a = a << b;

>>= : 右移等 a>>=b 等价于 a = a >> b;

&= : 与等 a&=b 等价于 a = a & b;

^= : 异或等 a^=b 等价于 a = a ^ b;

|= : 或等 a|=b 等价于 a = a | b;

...

2) 比较操作符

> : 大于

>= : 大于等于

< : 小于

<= : 小于等于

以上操作符只适用于整数类型和浮点数类型；最后的返回结果为boolean类型。

int a=1,b=1;

double d=1.0;

boolean result1 = a>b; //result1的值为false;

boolean result2 = a<b; //result2的值为false;

boolean result3 = a>=d; //result3的值为true;

boolean result4 = a<=b; //result4的值为true;

instanceof: 判断一个引用类型所引用的对象是否是一个类的实例。该操作符左边是一个引用类型，右边是一个类

名或接口名。形式如下：

obj instanceof ClassName

或者

obj instanceof InterfaceName

例如：

String a = "zs";

System.out.println(a instanceof String); //输出true;

3) 相等操作符

== : 等于

!= : 不等于

既可以是基本类型，也可以是引用类型：

a. 基本类型：

int a=1,b=1;

float c=1.0f;

double d=1.0;

System.out.println(a==b); //输出true;

System.out.println(a==c); //输出true;

System.out.println(a==d); //输出true;

System.out.println(c==d); //输出true;

b. 引用类型：

这两个引用变量必须都引用同一个对象，结果才为true.

Student s1 = new Student("zs",25,100);

Student s2 = new Student("zs",25,100);

Student s3 = s1;

System.out.println(s1 == s2); //输出false;

System.out.println(s1 == s3); //输出true;

System.out.println(s2 == s3); //输出false;

判断值是否相等String.equals()

4) 算术运算操作符

+ : 数据类型值相加或字符串连接;

a. 数据类型值相加；

int a=1+2; //a值为3;

double b=1+2; //b值为3.0;

double b=1+2.0; //c值为3.0;

b. 字符串连接；

System.out.println(1+2+"a"); //输出3a

System.out.println(1+2.0+"a"); //输出3.0a

System.out.println(1+2.0+"a"+true); //输出3.0atrue

System.out.println("a"+1+2); //输出a12

System.out.println(1+"a"+2); //输出1a2

- : 减

\* : 乘

/ : 整除, 如操作数均为整数，运算结果为商的整数部分

int a1=12/5; //a1变量的取值为2

int a2=13/5; //a2变量的取值为2

int a3=-12/5; //a3变量的取值为-2

int a4=-13/5; //a4变量的取值为-2

int a5=1/5; //a5变量的取值为0

double a6=12/5; //a6变量的取值为-2.0

double a7=12/5.0; //a7变量的取值为-2.4

% : 取模操作符, 如操作数均为整数，运算结果为商的整数部分

int a1=1%5; //a1变量的取值为1

int a2=13%5; //a2变量的取值为3

double a3=1%5; //a3变量的取值为1.0

double a4=12%5.1; //a4变量的取值为1.8000000000000007

数学运算，比如乘方等，使用java.lang.Math

5) 移位操作符

>> : 算术右移位运算，也称做有符号右移位运算。

int a1 = 12 >> 1; //a1变量的取值为6;

int a2 = 128 >> 2; //a2变量的取值为32;

int a3 = 129 >> 2; //a3变量的取值为32;

int a3 = 12 >> 33; //a3变量的取值为6;

注：a. 对12右移一位的过程为：舍弃二进制数的最后一位，在二进制数的开头增加一位符号位，由于12是正整数，因此增加的符号位为0;

b. 对-12右移一位的过程为：舍弃二进制数的最后二位，在二进制数的开头增加二位符号位，由于-12是负整数，因此增加的符号位为1;

c. 表达式" a>>b " 等价于： a/2^(b%32)

对于低于int类型(如byte/short/char)的操作数总是先自动类型转换为int类型再位移；

对于int类型的整数移位a>>b，当b>32时，先用b对32求余，得到的结果才是真正移位的位数，例如a>>33和a>>1的结果完全一样；

对于long类型的整数移位a>>b，当b>64时，先用b对64求余，得到的结果才是真正移位的位数，因为long类型是64位。