

<#>

请到场的同学扫码签到




Java A互助课

堂

lecture1

导生：杨宗奇

- 互助课堂仅作为老师讲课内容的解释和补充，请不要以听互助课堂取代听大课
- 本课件主要内容来自于老师上课课件，并对其作了修改和补充。
- 互助课堂旨在帮助大家提升学业成绩，有困难之处欢迎讨论交流，但要**杜绝抄袭、代写等行为。不要以可存留的方式分享代码**
- 课件中标红或有 ★ 标志的个人认为是需要重点理解的内容。有
 标志的表示需要记忆但不用理解很深的内容。

<#>

自我介绍

杨宗奇

2021级导生，来自浙江台州

就读于计算机科学与技术专业

比我厉害的大佬有很多，希望与大家以交流学习的心态互相进步，讲解若有错误欢迎与我讨论并批评指正。



<#>

JavaA学生现状



第一周选



第二周睡



第三周摆烂



第四周崩溃



第五周听啥啥不会



第六周被代码灌醉



第七周bug多到跪



第八周终于把课退

<#>

互助课堂群号



群名称:2023秋JavaA互助课堂

群 号:703021043

Java自学网课

- 尚硅谷2023Java教程, Java入门, java基础, java面试题

Source: [https://www.bilibili.com/video/BV1PY411e7J6/?](https://www.bilibili.com/video/BV1PY411e7J6/?spm_id_from=333.999.0.0&vd_source=0be91be5d566db743f3ab2797403cf2b)

[spm_id_from=333.999.0.0&vd_source=0be91be5d566db743f3ab2797403cf2b](https://www.bilibili.com/video/BV1PY411e7J6/?spm_id_from=333.999.0.0&vd_source=0be91be5d566db743f3ab2797403cf2b)

- Java入门基础视频教程, java零基础自学就选黑马程序员Java入门教程
(含Java项目和Java真题)

[https://www.bilibili.com/video/BV1Cv411372m/?](https://www.bilibili.com/video/BV1Cv411372m/?spm_id_from=333.337.search-card.all.click&vd_source=0be91be5d566db743f3ab2797403cf2b)

[spm_id_from=333.337.search-](https://www.bilibili.com/video/BV1Cv411372m/?spm_id_from=333.337.search-card.all.click&vd_source=0be91be5d566db743f3ab2797403cf2b)

[card.all.click&vd_source=0be91be5d566db743f3ab2797403cf2b](https://www.bilibili.com/video/BV1Cv411372m/?spm_id_from=333.337.search-card.all.click&vd_source=0be91be5d566db743f3ab2797403cf2b)

尚硅谷Git入门到精通全套教程

- Source: https://www.bilibili.com/video/BV1vy4y1s7k6/?spm_id_from=333.1007.top_right_bar_window_custom_collection.content.click&vd_source=0be91be5d566db743f3ab2797403cf2b
- 做project很有用的工具

目录

CONTENTS

01

基础知识

02

OJ工作机制及常见问题

03

程序的主要构成

04

程序输出

05

变量与程序输入

06

运算符

基础知识

<#>

冯·诺依曼结构 ▲

I/O + ACM

Input unit



键盘、鼠标等

Output



扬声器、显示屏等

Memory



RAM运行内存
访问快，容量小
断电会丢失数据

<#>

冯·诺依曼结构

算术逻辑
AL 单元



四则运算、大小比较、分支判断等

CP
U



控制各单元的工作

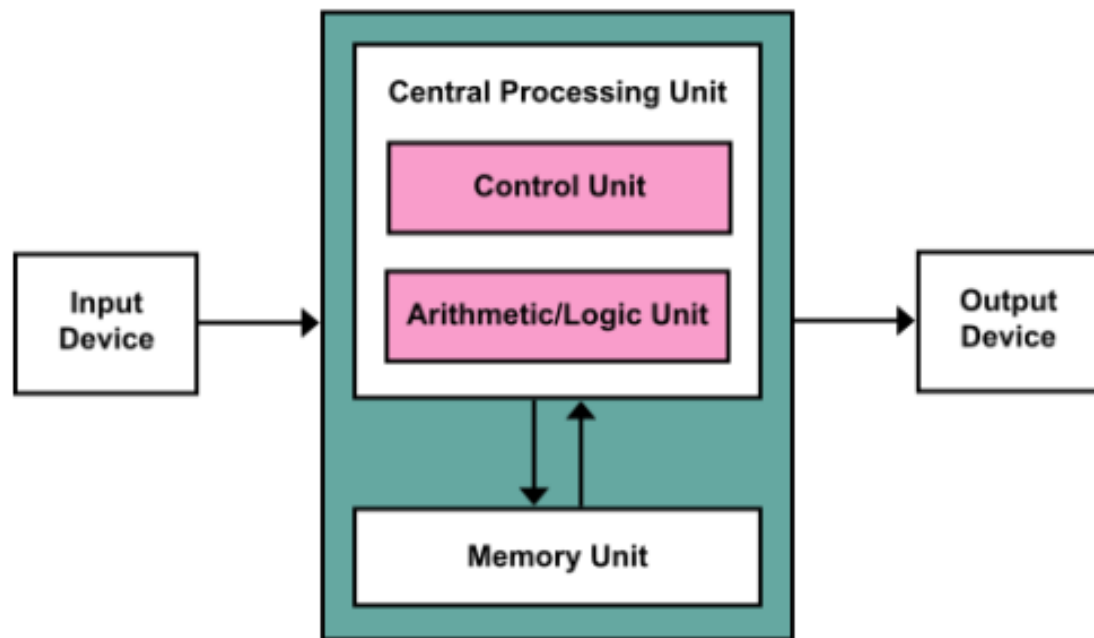
Secondary
Storage
Unit



U盘、固态硬盘、光盘等外部存储工具。容量大，长期存储，断电不丢失

<#>

冯·诺依曼结构



John von Neumann
(1903-1957)
Hungarian-American
mathematician, physicist

- 生活中我们会遇到一些问题，我们可以设计解决它们的方法，但由于其规模较大，人为地去进行操作会很难解决。遇到这一类问题，我们通常采用编程的方法，借助计算机的运算能力来解决。

我 + 高数题 + 答案解法 = 做对

电脑 + 运算逻辑 + 正确程序 = 解决问题

- 编程实际上就是指导程序完成某个特定目标的过程。我们现在学习的便是了解计算机的运算逻辑，以及如何用代码来指导其运算。

- 计算机里的所有数据都以二进制形式存储，人类无法阅读并理解。

Can You Understand This?

- 00001001001011100110011001101001011011000110010100001001001000100110110001100101011
00011011101000111010101110010011001010011000100101110011000110010001000001010011001
11011000110110001100110010010111110110001101101111011011010111000001101001011011000
11001010110010000101110001110100000101000101110011100110110010101100011011101000110
10010110111101101110000010010010001000101110011101000110010101111000011101000010001
00000101000001001001011100110000101101100011010010110011101101110001000000011010000
00101000001001001011100110011101101100011011110110001001100001011011000010000001101
10101100001011010010110111000001010000010010010111001110100011110010111000001100101
00001001001000000110110101100001011010010110111000101100001000110110011001110101011
01110011000110111010001101001011011110110111000001010000010010010111001110000011100
10011011110110001100001001001100000011010000001010011011010110000101101001011011100
01110100000101000001001001000010010001101010000010100100100111101001100010011110100
01110101010101000101001000110010000000110000000010100000100101110011011000010111011
001

- 从而产生了汇编语言，可以理解为汇编语言是二进制的一种机器能读懂的翻译。

How about This?

```
main:
    !#PROLOGUE# 0
    save %sp,-128,%sp
    !#PROLOGUE# 1
    mov 1,%o0
    st %o0,[%fp-20]
    mov 2,%o0
    st %o0,[%fp-24]
    ld [%fp-20],%o0
    ld [%fp-24],%o1
    add %o0,%o1,%o0
    st %o0,[%fp-28]
    mov 0,%i0
    nop
```


- 但汇编语言仍然过于复杂繁琐，因此衍生出了更多简单易懂的编程语言，例如Java, C++, Python等

Is It Better Now?

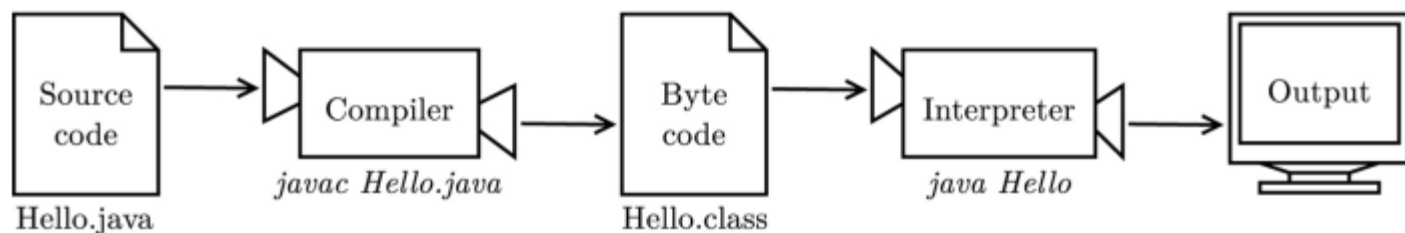
```
int valueofz( )  
{  
    int x, y, z;  
    x = 1;  
    y = 2;  
    z = x+y;  
    return z;  
}
```

- 编译器则是编程语言的翻译器，它可以将便于人类理解的代码转化为计算机可以理解的机器语言（二进制数据），从而实现指导计算机的运算。



编译执行和解释执行

- 解释执行：逐行将代码转化为机器语言并运行。就好像吃火锅，边涮边吃。
- 编译执行：将整个程序转化为机器语言，再直接运行转化好的机器语言。
类似于烧好了一大桌菜，一起开吃。
- ▲ Java与上述两种不同，是一种半解释半编译的语言。源文件(.java)通过编译器生成字节码(.class)，再通过解释器生成机器码





Java平台

- **JVM: Java Virtual Machine**
用于执行Java程序的虚拟机
 - **JRE: Java Runtime Environment**
提供执行Java程序所需的基本工具
 - **JDK: Java Development Kit**
用于开发Java程序的其他实用的工具包
- ▲ **JDK = JRE + Development tools, JRE = JVM + Library classes**

- **Syntax Errors: 语法错误**

程序不符合语法规则，**无法通过编译**，编译器会指出错误位置

- **Runtime Errors: 运行错误**

程序通过编译，但在运行的时候出现错误，常见原因有数组下标越界，除0错误等。

- **Logic Errors and Semantics: 逻辑或语义错误**

程序能正常编译运行，但运行结果跟所预期的不同。一般是因为算法设计不完善或者是编程时没有正确将算法转化为代码。

The slide features a minimalist design with two sets of overlapping geometric shapes in the corners. The top-left corner consists of a dark blue triangle and a light blue parallelogram. The bottom-right corner features a dark blue parallelogram and a light blue triangle. The central text is bold and black.

OJ工作机制及常见问题

- 对于某个问题，大多数情况下会有无穷多种不同的输入情况
- 对于某些特定的输入，我们可以先通过正确的程序，或者用人为的方法得到正确的输出。
- 再利用这些输入给到待检验的程序，并将程序的输出与正确答案比对。

- OJ(Online Judge)是基于上述方法来检验程序正确性的
- OJ后台会有很多组预先准备好的程序输入，以及对应的正确输出
- 它们以文件形式存储，输入和输出互不相干。
- 当你提交代码时，OJ会运行你的程序，输入数据得到输出，并与提前准备的标准答案输出进行比对，完全一致则认为在**该数据测试下**你的代码是正确的。（Accepted）
- 因为可能的输入是无穷多的，所以通过OJ并不代表你的算法一定是正确的，但没通过则说明你的算法一定有错误。

样例过了，为什么交上去是错的？

- 样例只是众多对数据中较为简单的一组
- 它用于初步检测你程序的正确性
- OJ后台还有很多更复杂，更特殊的数据
- 同理，通过样例不代表你就能过OJ，但是过不了样例八成过不了OJ（除非OJ数据过于简单）
- 遇到这种情况请用各种方法来检查修正你的代码

为什么在输入的时候会弹出输出，跟OJ输出不一样？

- IDE的输入输出是按时间顺序显示的，会交错在一起
- 而给的样例以及OJ内部的输入输出是分隔开的，互不影响
- 可以理解为是从一个文件输入，然后输出到另一个文件，比对输出的文件跟答案所在文件内容是否相同
- 因此只要观察输出的内容是否跟样例是否一致即可。（注意区分你在输入时自己按下的回车和输出的回车）

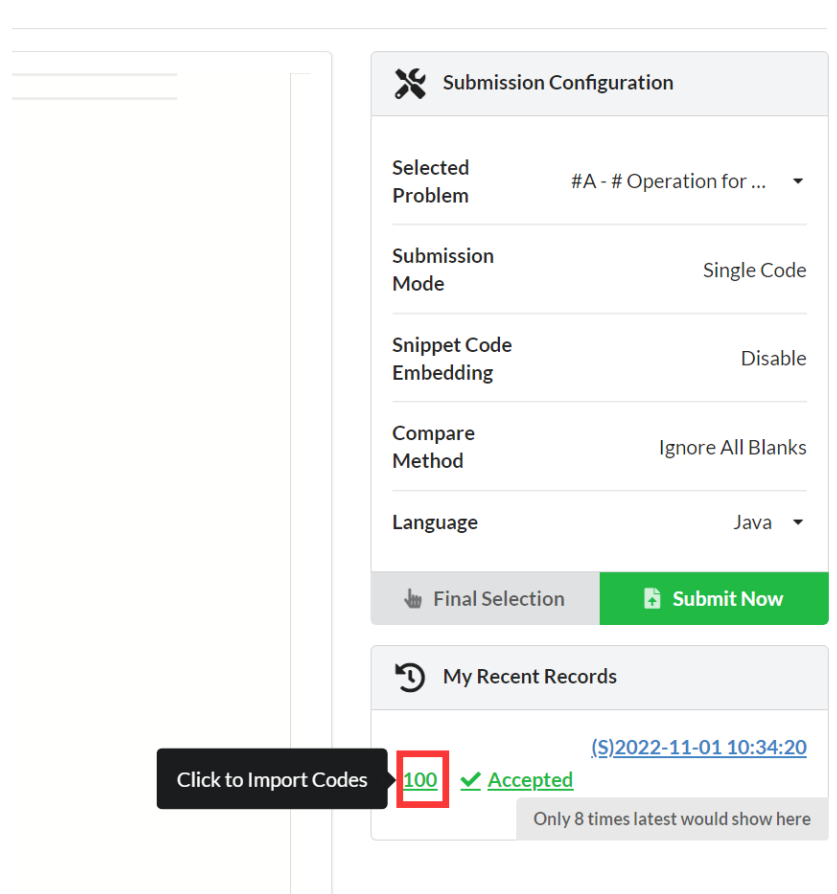
为什么会报CE或RE?

- 首先检查①主类必须是Main②文件开头不能加package
- 其它的CE一般编译器就会报错，如果提交了还报错一般是主类名不正确
- 对于RE有很多可能，我们下节课讲Debug时会根据不同错误类型介绍错误的可能以及Debug的方向

<#>

新手使用OJ时的问题

如何直接获取我提交过的代码？




- 在提交下方的页面点击某次提交的分数即可将之前的提交复制到当此界面中

为什么之前对的但是分数变少了？

- OJ以最后一次提交为准
- 需要选择记录分数的是哪一次提交
- 提交其他题时OJ经常会自动跳到A题导致交错，要注意这个情况

<#>

新手使用OJ时的问题

 Submission Configuration

Selected Problem

#B - Perimeter of Bla... ▾

Submission Mode

Single Code

Snippet Code Embedding

Disable

Compare Method


Ignore All Blanks



Language

Java ▾


Final Selection

Submit Now

 My Recent Records

0	 Compile Error	(S)2023-03-08 18:25:15
0	 Runtime Error	2023-03-08 18:24:46
100	✓ Accepted	2022-11-02 16:27:28
100	✓ Accepted	2022-11-01 11:22:33

Only 8 times latest would show here

 Select Record as Final Submission

☰ 177146

*

Cancel

Confirm

177146 - Submission Record

▶ compile	✓ Accepted	0 pts	0 ms	0 MB
▶ TestCase: 1	✓ Accepted	10 pts	305 ms	14.8 MB
▶ TestCase: 2	✓ Accepted	10 pts	237 ms	14.7 MB
▶ TestCase: 3	✓ Accepted	10 pts	237 ms	14.8 MB
▶ TestCase: 4	✓ Accepted	10 pts	506 ms	24.5 MB
▶ TestCase: 5	✓ Accepted	10 pts	261 ms	14.8 MB
▶ TestCase: 6	✓ Accepted	10 pts	365 ms	14.8 MB
▶ TestCase: 7	✓ Accepted	10 pts	353 ms	21.5 MB
▶ TestCase: 8	✓ Accepted	10 pts	301 ms	19.1 MB
▶ TestCase: 9	✓ Accepted	10 pts	345 ms	21.5 MB
▶ TestCase: 10	✓ Accepted	10 pts	353 ms	14.7 MB

 Result  Code  Analysis

Record Information


Problem [#CS10922F011 - Perimeter of Black Grid Cells](#)


Judging State ✓ Accepted

Score 100

Submission Time 2022-11-02 16:27:28

My Recent Records


0  Compile Error 2023-03-08 18:25:15

0  Runtime Error 2023-03-08 18:24:46

100 ✓ Accepted 2022-11-02 16:27:28

100 ✓ Accepted 2022-11-01 11:22:33


Only 8 times latest would show here

The slide features a minimalist design with two sets of overlapping geometric shapes in the corners. The top-left corner contains a dark blue parallelogram and a light blue parallelogram. The bottom-right corner contains a dark blue parallelogram and a light blue parallelogram. The central text is in a bold, black, sans-serif font.

程序的主要构成



程序的主要构成



```
1 // Text printing program
2 public class Welcome1 {
3     // main method begins execution of Java application
4     public static void main (String[] args) {
5         System.out.println("Welcome to Java Programming!");
6     } // end method main
7 } // end class Welcome1
```


<#>

程序的主要构成——注释

注释



```
1 // Text printing program
2 public class Welcome1 {
3 // main method begins execution of Java application
4     public static void main (String[] args) {
5         System.out.println("Welcome to Java Programming!");
6     } // end method main
7 } // end class Welcome1
```

```
1 /*
2     This is a traditional comment. It
3     can be split over multiple lines
4 */
5
6 /* This is a traditional comment. It
7 can be split over multiple lines */
```

- // ： 注释一行
- /* */ :注释整段，与最近的匹配

Traditional vs. End-of-Line Comments

- Traditional comments do not nest, the first `*/` after the first `/*` will terminate the comment.

```
11  /*
12  |  /* comment 1 */
13  |      comment 2 */
```

[点击固定](#)

- End-of-line comments can contain anything.

```
15  // /* This comment is okay */
```

- 多行注释不能嵌套，注释范围为第一个`/*`到第一个`*/`之间的位置
- 单行注释可以注释该行任何内容

```
1 // Text printing program
2 public class Welcome1 {
3     // main method begins execution of Java application
4     public static void main (String[] args) {
5         System.out.println("Welcome to Java Programming!");
6     } // end method main
7 } // end class Welcome1
```

- Java的代码是有层次关系的，花括号内的内容表示从属于某一结构
- 基本的层次是 类——方法——循环、分支等嵌套

- 在java中作用域是有花括号的位置决定的,它决定了其变量名的可见性与生命周期
- 简单来说,就是能在哪里使用,什么时候消失。
- 对于在方法中出现的变量,我们称为local variable (局部变量), 它们有一个很重要的性质: 它们的作用域为它们出现的花括号所包括的范围
- 也就是说局部变量只能在其出现的花括号内被访问和使用, 离开了花括号的作用域, 该变量会从内存空间中被删除。
- 使用 (Tab) 缩进来区分作用域和语句之间的从属关系, 是个必须要养成的习惯

```
1 // Text printing program
2 public class Welcome1 {
3     // main method begins execution of Java application
4     public static void main (String[] args) {
5         System.out.println("Welcome to Java Programming!");
6     } // end method main
7 } // end class Welcome1
```

- Java要求每个文件必须有public class，才能生成可执行文件
- 这里暂时不多作介绍，随着后续学习再继续深入

```
1 // Text printing program
2 public class Welcome1 {
3 // main method begins execution of Java application
4     public static void main (String[] args) {
5         System.out.println("Welcome to Java Programming!");
6     } // end method main
7 } // end class Welcome1
```

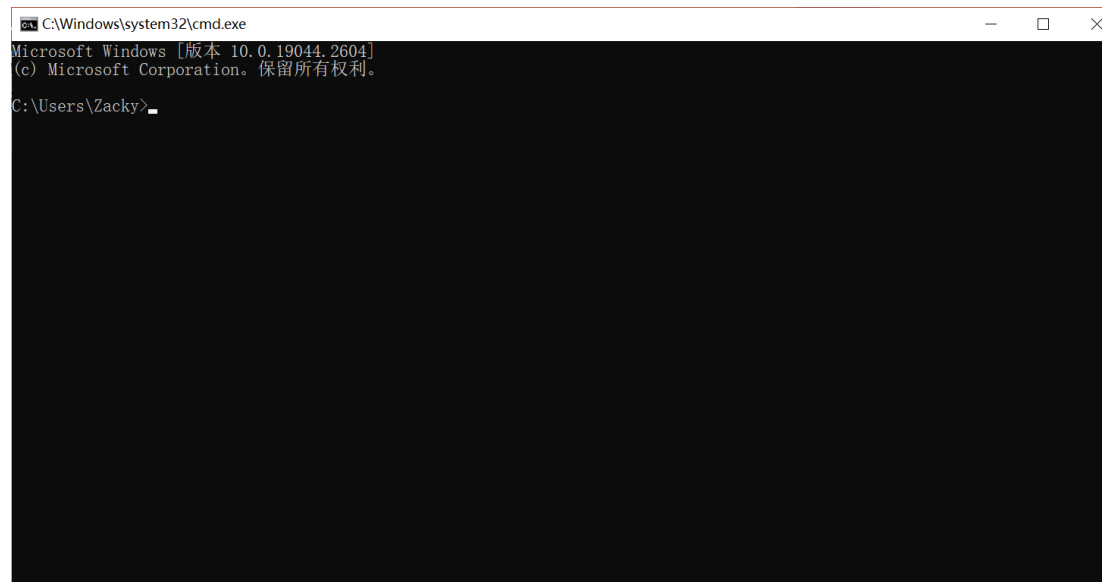
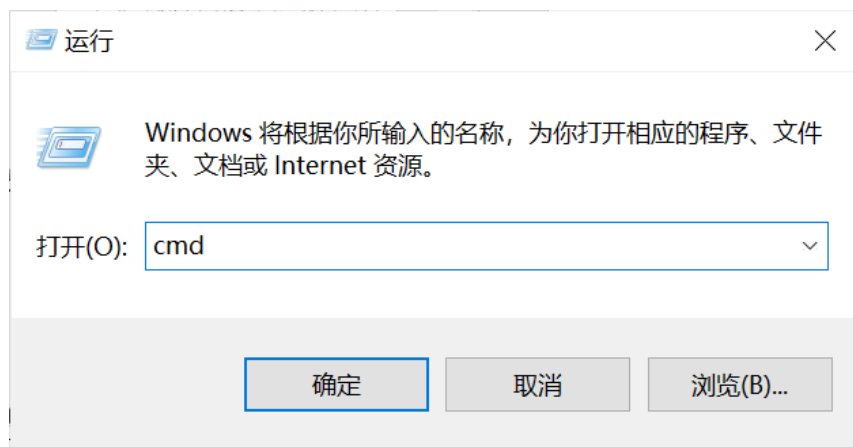
- 方法是我们描述计算机应该做些什么的地方
- 主方法是程序的入口，计算机找到叫做main的方法开始按顺序执行代码

```
1 // Text printing program
2 public class Welcome1 {
3 // main method begins execution of Java application
4     public static void main (String[] args) {
5         System.out.println("Welcome to Java Programming!");
6     } // end method main
7 } // end class Welcome1
```

- 方法中的语句即为我们如何指导计算机做一些事情
- 方法中的语句可以有多种嵌套、组合，从上到下依照语法逻辑执行

程序输出

〈#〉 命令行编译运行



- 计算机中的命令行可以通过键盘键入指令来实现一切我们用鼠标的能完成的操作，比如打开文件夹、创建删除文件等等。
- 这些方便可可视化的操作都是我们的操作系统提供的，如Windows、linux、macOS
- 同样的，我们在IDE中一件编译运行的操作也是可以在命令行实现的

〈#〉 命令行编译运行

```
C:\Users\Zacky\Desktop>javac test.java  
C:\Users\Zacky\Desktop>java test  
Hello world  
C:\Users\Zacky\Desktop>
```



- **javac**命令用于编译文件，后面的文件要加.java，编译后生成.class文件
- **java**用于运行生成的.class文件，输入的时候不需要.class
- **IDE**会将输入参数以及输出结果的命令行显示在其界面下方

First Program in Java: Printing a Line of Text (Cont.)

`System.out` object

5

```
System.out.println("Welcome to Java Programming!");
```

- Standard output object
- Allows Java applications to display strings in the **command window** from which the Java application executes
- `System.out.println` method
 - Displays (or prints) a line of text in the command window.
 - The string in the parentheses is the **argument** to the method.
 - **Positions the output cursor at the beginning of the next line in the command window**
- 可以将System类比作为一个工具包，out类比作为一个工具，out.println则为此工具能做到输出字符串到命令窗口中这么一件事情。

〈#〉 输出方式

System.out.print()

- 直接输出括号中的内容，不换行

System.out.println()

- ln是line的缩写，表示输出括号中的内容后换行

System.out.printf()

```
System.out.printf("%s%n%s\n", "Welcome to", "Java Programming!");
```

- 第一个位置的字符串称为正则表达式，表示了输出由什么构成，暂时不确定内容但知道类型的地方用一个占位符代替（可以类比占座）。而后面逗号跟的内容则按顺序填入我们曾占了位置的地方。

Common Escape Sequences

Escape Sequence	Description
<code>\n</code>	Newline. Position the cursor at the beginning of the next line.
<code>\t</code>	Horizontal tab. Move the cursor to the next tab stop.
<code>\r</code>	Carriage return. Position the cursor at the beginning of the current line (do not advance to the next line). Any characters output after the carriage return overwrite the characters previously output on that line.
<code>\\</code>	Use to print a backslash character.
<code>\"</code>	Used to print a double-quote character. <pre>System.out.println("\"in quotes\");</pre> displays "in quotes"

- 右斜杠(\)称为转义符(**escape character**), 它用于和其它字符组合表达一些不方便表达的输出内容。
- 转义符和另一个字符组成escape sequence, 表达某个特殊含义

The image features a light gray background with decorative geometric elements. In the top-left corner, there are overlapping triangles in dark blue and light blue. In the bottom-right corner, there are overlapping triangles in dark blue and light blue, mirroring the top-left design.

变量与程序输入



什么是变量？

- 变量在程序中是一种数据的载体。
- 可以将变量类比为装液体的容器
- 容器的种类不同（变量类型），可以装的液体也不同。
- 容器可以贴标签（变量名），用来表示其用途

<#>

为什么要使用变量?

Can you understand this?

$a*b$

Or this?

$5*5$

What about this?

$\text{price}*\text{num}$

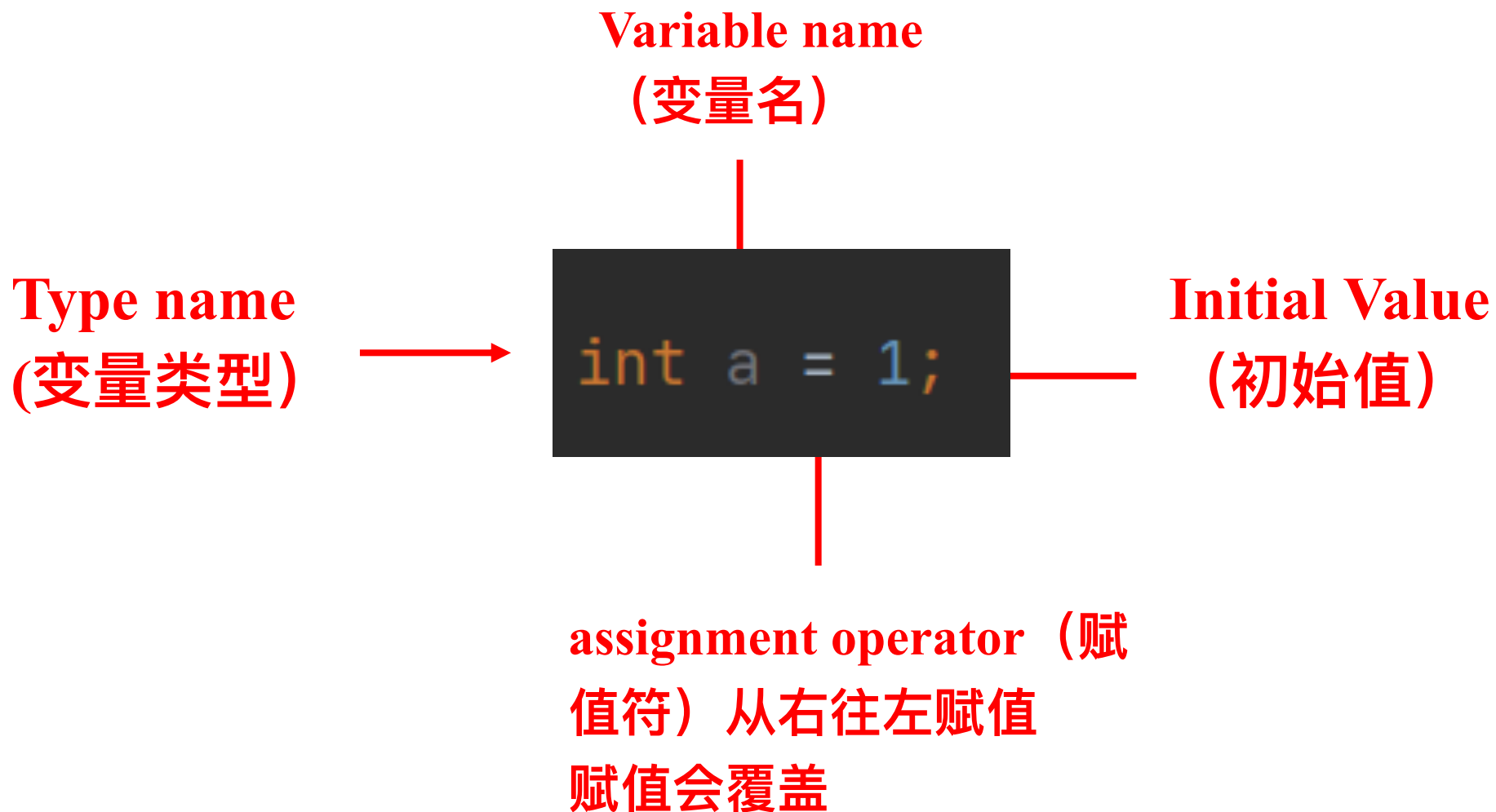
〈#〉

为什么要使用变量？

- 为了存储不同的输入值
- 为了让我们的程序更易理解和修改

<#>

变量的定义





变量的定义

- 变量在赋值和调用前必须先被定义
- 变量在调用前必须先被赋值
- 变量只能在其作用域内使用，对于局部变量，即为其出现的花括号内
- 同一个作用域不能有变量名相同的变量

<#>

命名

- 可以由①英文字符②数字③下划线④\$ 组成
- 不能以数字开头
- 不能使用java保留字作为变量名
- 建议变量名与实际用途相关

<#>

赋值



- 赋值时从右往左
- 变量会被新赋的值给覆盖

练习：若有相同类型的变量a, b,如何用第三个变量t实现a, b值的交换？

```
int a=1,b=2,t;  
t=a;  
a=b;  
b=t;  
System.out.printf("a:%d b:%d",a,b);
```

```
a:2 b:1  
Process finished with exit code 0
```




变量类型

Type	Size	Range
byte	8 bits	-128 to +127
short	16 bits	-32,768 to +32,767
int	32 bits	(about) -2 billion to +2 billion
long	64 bits	(about) -10E18 to +10E18

整型

Type	Size	Range
float	32 bits	-3.4E+38 to +3.4E+38
double	64 bits	-1.7E+308 to 1.7E+308

浮点数
(不精确)



```
public class test{  
    public static void main(String[] args){  
        System.out.println(1.1f*3.0f==3.3f);  
        System.out.printf("%.16f",1.1f);  
    }  
}
```

```
false  
1.1000000238418580  
Process finished with exit code 0
```

- 连续区间的数是无穷的，计算机只能存储离散的数值
- 对一个浮点数进行赋值时，会以最接近的值去代表。
- 因此在浮点数计算时会有精度损失
- **浮点数不要直接用“==”比较相等**，这样的操作是很不安全的

- **char** 字符类型变量，本质是一个16bit的整型，以某种编码方式对应到各个字符。
- **boolean** 布尔类型变量，值只有true 或 false ,用于表示逻辑表达式的值。 各类逻辑表达式的值都是布尔类型的，例如 $a==b$, $a+b \geq c$ 等，可以用boolean类型变量存储这些表达式的值，即对其赋值。


```
1 import java.util.Scanner;
```

- 可以理解为import获取了工具库中叫Scanner的这类工具，该工具用于读入数据。
- 请记住Scanner所在的库
- ▲ • 同时System所在的库为**java.lang**，该库默认被程序引用

<#>

程序输入

类名

变量名

新建Scanner类的实例

```
Scanner input = new Scanner(System.in);
```

- Scanner是类，也就是我们所说的工具类别。
- 我们要使用这类工具，我们就要创建一个实际的工具来使用。比如我们要用剪刀，那我们就得买一个特定的剪刀来使用。这个特定的工具就称为这个类的实例。
- 变量（reference type）用来指向这个实际的工具。需要的时候就通过变量存的地址找到它。相当于给这个剪刀一个称呼，用来指定这个剪刀。

- 对于一段特定的输入，我们可以**根据实际需要以不同的变量类型**输入
- 例如，输入“1”，我们可以将其存为整型或字符型变量。
- **常用的Scanner输入方法**
- `next()` 读入一个字符串，遇到回车、换行、制表符停止。
- `nextLine()` 读入一整行字符串，包括换行、制表符等，遇到回车停止。
- `next+变量类型`（首字母大写） 读入对应变量的值，遇到回车、换行、制表符停止。

例如，`nextInt()` 读入数据作为`int`类型数值。

运算符

Operator	Description	Algebra Expression	Java Expression
+	Additive operator (also used for String concatenation)	$f + 7$	<code>f + 7</code>
-	Subtraction operator	$p - c$	<code>p - c</code>
*	Multiplication operator	bm	<code>b * m</code>
/	Division operator	x/y or $\frac{x}{y}$ or $x \div y$	<code>x / y</code>
%	Remainder operator	$r \bmod s$	<code>r % s</code>

- 按优先级运算，优先级相同从左到右，有括号先算括号
- *,/,%优先级相同，大于+,-

<#>

转换和提升



```
1 int chinese = 92;  
2 int english = 90;  
3 int math = 89;  
4  
5 double average = (chinese + english + math) / 3;
```

- average的值是多少?

```
90.0
```

```
Process finished with exit code 0
```



- 在java中，小数默认为double类型，整数默认为int类型
- 同一个类型的值进行运算，只能得到该类型值的结果。
- 两个不同类型的值进行运算，会将精度低的自动转变为精度高的类型，再进行运算
- 高转低可以强制转换，会失去精度。方法为在要转换的变量前加括号，括号内写转换类型
- Primitive type精度排序（由低到高）
- (char, byte, short) □ int □ long □ float □ double
- char, byte, short 计算前都先转化为int

- 解决方案

(double) ()

```
double average=(chinese + english + math) /3.0;
```

```
double average=(double)(chinese + english + math) /3;
```

```
double average=((double)chinese + english + math) /3;
```

- 比较一下三者之间的区别



优先级	运算符
1	()
2	! +(正) -(负) ++ --
3	* / %
4	+(加) -(减)
5	< <= > >=
6	== !=
7	^
8	&&
9	
10	?:
11	= += -= *= /= %=

Source:

https://blog.csdn.net/weixin_49667274/article/details/123207671

<#>

复合赋值运算符



- *, /, %, +, -都可以写成如下的表达式

variable operator= expression;

变量名 \longrightarrow 等 c $\xrightarrow{\text{运算符}}$ += 3;

c = c + 3;

<#>

自增自减运算



- 单独成句时，`a++`; 和 `++a`; 都表示变量`a`的值+1
- 但当要使用到`a`变量时，两者略有不同

```
int a=1,b=0;  
b=a++;  
System.out.println(a);  
System.out.println(b);
```

等价



```
int a=1,b=0;  
b=a;  
a=a+1;  
System.out.println(a);  
System.out.println(b);
```



a:

2

b:

1

```
int a=1,b=0;  
b=++a;  
System.out.println(a);  
System.out.println(b);
```

等价



```
int a=1,b=0;  
a=a+1;  
b=a;  
System.out.println(a);  
System.out.println(b);
```



a:

2

b:

2

- `==` 用于判断两个相同类型的值是否相等，结果为一个布尔类型的值，若相等则表达式的值为true，**注意与赋值符=区分**
- `!=` 表示不等于，若两个值不相等则表达式的值为true
- 不应用这类运算符直接比较浮点数

```
double f1=1.1f;  
double f2=110.0/100;  
System.out.println(f1==f2);
```

```
false
```

```
Process finished with exit code 0
```



&& (conditional AND)

|| (conditional OR)

& (boolean logical AND)

| (boolean logical inclusive OR)

^ (boolean logical exclusive OR)

! (logical NOT)

- &&, ||, !, 只用来表达布尔类型值之间的关系
- &, |, ^ 可以作用于整型的每个二进制位
- && 有0出0, 全1出1
- || 有1出1, 全0出0

- 涉及 `&&` 或 `||` 的表达式中，若在运算符左边的表达式已经能确定表达式的值，则不会对其余的表达式进行计算。

```
boolean b;  
int a=1,c=1;  
b=(a==1 || a==c);  
b=(a==0 && a==c);
```

- 以下两个表达式都不会执行对 `a==c` 的判断，因为第一个表达式已经可以确定这个运算的结果



**Thanks for
watching**