**今天完成的事情：**

**第一部分：JAVA基础知识**

**学习预览：**

* JAVA循环结构
* JAVA条件语句
* JAVA方法

1. **JAVA循环语句**

JAVA循环语句与C类似

**Java中有三种主要的循环结构：**

* while循环
* do...while循环
* for循环

1. **while循环**

while是最基本的循环

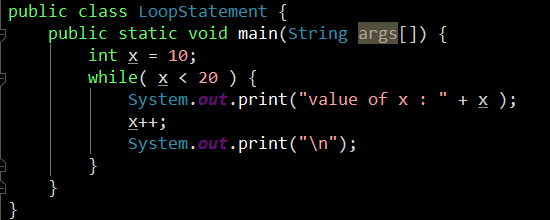
它的结构为：

**while（条件表达式（布尔型true/false））｛**

**//循环内容**

**｝**

**实例为**



1. **do...while循环**

条件循环语句，先执行do再循环while

结构为

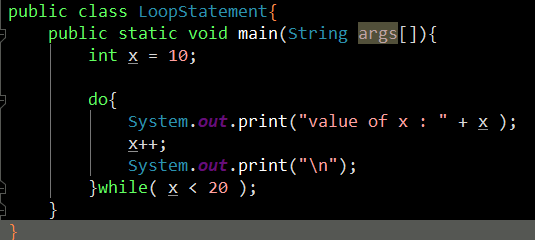
**do {**

**//代码语句**

**}**

**while(条件表达式（布尔型）);**

**实例为**



1. **for循环**

for循环与C同理

先初始化，再判定条件，再执行代码语句，最后循环变量运算

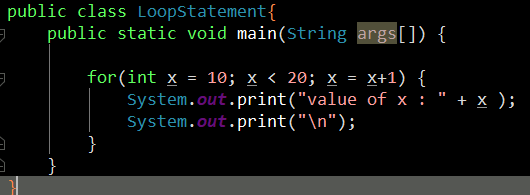
结构为

**for(初始化; 条件表达式; 循环变量算式) {**

**//代码语句**

**}**

**实例为**



1. **增强型for循环**

JAVA5引入的专门用于数组的for循环

**声明：**声明新的局部变量，该变量类型必须与数组元素的类型匹配，且相等

**表达式：**表达式是要访问的数组名，或是返回值为数组的方法

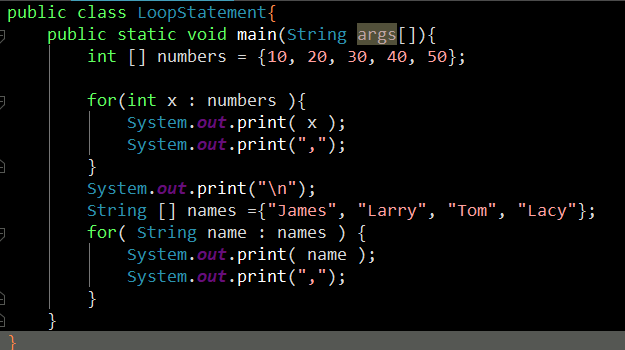
**结构为**

for(声明语句 : 表达式){

//代码句子

}

实例为



1. **break关键字**

跳出循环关键字

在循环中插入关键字，直接跳出循环

1. **continue关键字**

跳出循环，立刻进入下一次循环的关键字

在for循环中，立刻跳入循环变量更新表达式

在while或do...while循环中，立刻跳到条件表达式进行判断操作

1. **JAVA条件语句**

**JAVA条件语句与C语言类似**

1. **If条件语句**

先判断条件，如果符合条件执行if条件代码

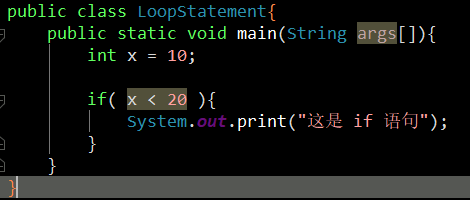
结构为

**if(布尔表达式){**

**//如果布尔表达式为true将执行的语句**

**}**

**实例为**



1. **If...else条件语句**

先判断条件，符合条件执行if条件代码，不符合执行else条件代码

结构为

**if(布尔表达式){**

**//如果布尔表达式的值为true**

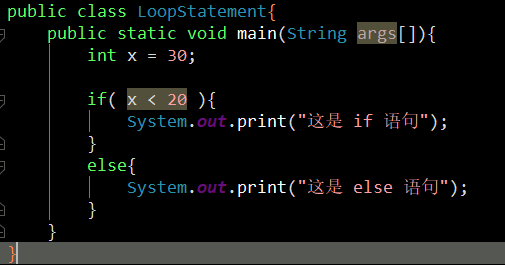
**}**

**else{**

**//如果布尔表达式的值为false**

**}**

**实例为**



1. **if...else if...else 语句**

多重条件语句

先判断条件，符合条件执行if条件代码

不符合条件，进行else if条件判定，通过则执行else if代码

不符合条件，执行else代码

可多重else if判定

其中有一个符合条件，执行完代码后，既跳出条件语句

结构为

**if(布尔表达式 1){**

**//如果布尔表达式 1的值为true执行代码**

**}else if(布尔表达式 2){**

**//如果布尔表达式 2的值为true执行代码**

**}else if(布尔表达式 3){**

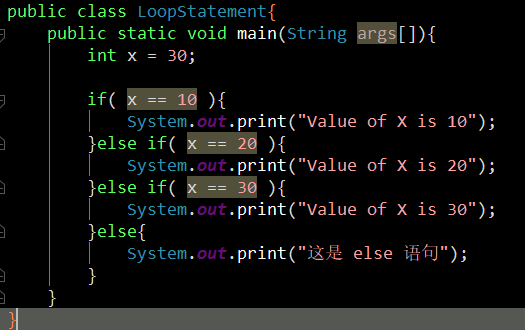
**//如果布尔表达式 3的值为true执行代码**

**}else {**

**//如果以上布尔表达式都不为true执行代码**

**}**

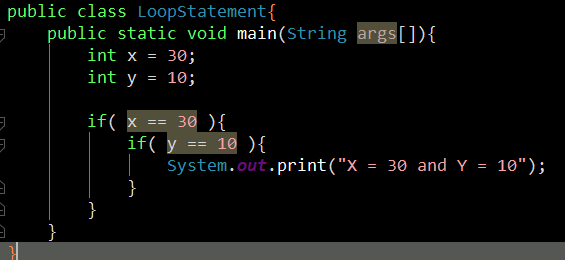
实例为



1. **嵌套if...else语句**

在if语句中嵌套if...else语句

**实例为**



1. **switch case语句**

switch case语句判断变量是否与目标值匹配，每个值拥有一个代码分支

可以有任意数量的case语句

**switch case执行时，一定会先进行匹配，匹配成功返回当前case的值**

**再根据是否有break，判断是否继续输出，或是跳出判断**

**如果case语句块中没有break语句时，JVM不会顺序输出每一个case对应的返回值，而是继续匹配，匹配不成功则返回默认case**

**如果case语句块中没有break语句时，匹配成功后，从当前case开始，后续所有 case的值都会输出**

结构为

**switch(变量名){**

**case value :**

**//代码语句**

**break;**

**case value :**

**//代码语句**

**break;**

**default :**

**//代码语句**

**}**

其中break关键字和default关键字可选，视情况而定

**switch case执行时，一定会先进行匹配，匹配成功返回当前case的值**

**再根据是否有break，判断是否继续输出，或是跳出判断**

**如果case语句块中没有break语句时，JVM不会顺序输出每一个case对应的返回值，而是继续匹配，匹配不成功则返回默认case**

**如果case语句块中没有break语句时，匹配成功后，从当前case开始，后续所有 case的值都会输出**

* switch语句中的变量类型可以是： byte、short、int、char、字符串string
* case参数必须与变量类型相同，字符串常量或字面量（既不同数据类型的值）
* case 语句不必须要包含break关键字，如果没有break关键字出现，程序会继续执行下一条case语句，直到出现break关键字，则立刻跳出switch语句
* switch语句可以包含一个default分支，default分支一般是switch语句最后一个分支（可以在任何位置，建议在末尾），default在没有case语句的值和变量值相等的时候执行。default分支不需要break语句。

**实例为**



1. **JAVA方法**

Java方法是语句的集合，它们在一起执行一个功能

* 方法是解决一类问题的步骤的有序组合
* 方法包含于类或对象中
* 方法在程序中被创建，在其他地方被引用

例如：

常用到的System.out.println()

* println()是一个方法
* System是系统类
* out是标准输出对象

这个方法调用了系统类System中的标准输出对象out中的方法println()

1. **方法的命名规则**

* 第一个单词以小写字母开头，后面的单词以大写字母开头，不适用连接符
* 下划线出现在JUnit测试方法的名称中，用以分割名称的逻辑组件

1. **方法的定义**

一般情况下，定义一个方法包含以下语法：

**修饰符 返回值类型 方法名(参数类型 参数名){**

**...**

**方法主体**

**...**

**return 返回值;**

**}**

* **修饰符**

修饰符可选

告诉编译器如何调用该方法，定义了该方法的访问类型

* **返回值类型**

方法可能会返回值

returnValueType 是方法返回值的数据类型

执行操作，没有返回值的情况下，returnValueType是关键字void

* **方法名**

方法的实际名称

方法名和参数表共同构成方法签名

* **参数类型**

参数列表是指方法的参数类型、顺序和参数个数

参数是可选的，方法可以不包含任何参数

参数是一个占位符，当方法被调用时，传递值给参数，参变成实参或变量

* **方法内容**

方法体包含具体的语句，定义该方法的功能

1. **实例**

单参数

**public static int age ( int birthday ) {...}**

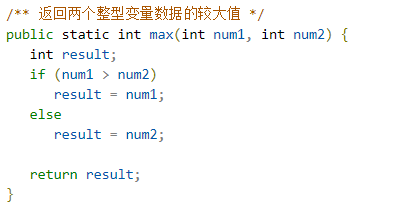
多参数

**static float interest ( float principal, int year ) {...}**

**在一些其它语言中方法指过程和函数**

* **一个返回非void类型返回值的方法称为函数**
* **一个返回void类型返回值的方法叫做过程**

多参数返回实例



1. **方法调用**

Java 支持两种调用方法的方式，根据方法是否返回值来选择

* **当方法返回一个值的时候，方法通常被当做一个值来调用**

例如：

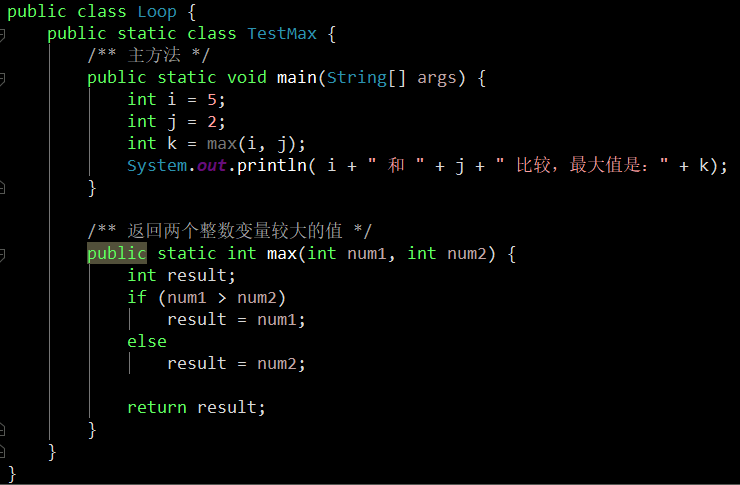
int larger = max(30, 40)

* **如果方法返回值是void，方法调用一定是一条语句**

例如：

System.out.println("基础真特么的多！")

实例：



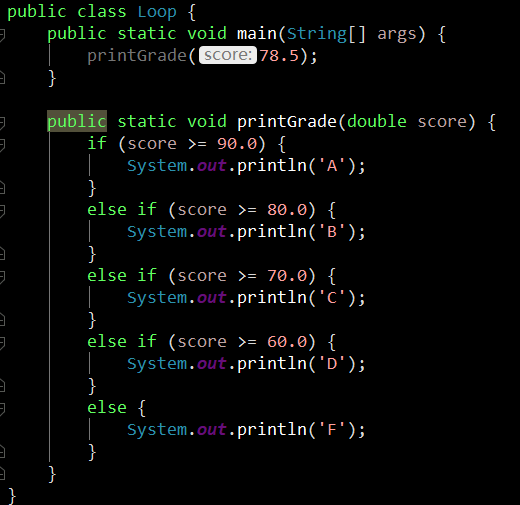
这个程序包含main方法和max方法

main方法是被JVM调用的，除此之外，main方法和其它方法没什么区别

main方法的头部是不变的，如例子所示，带修饰符public和static，返回void类型值，方法名字是main，此外带一个String[ ]字符串数组参数

1. **void关键字**

通过一个示例来说明如何声明和调用一个void方法



这里printGrade方法是一个void类型方法，它不返回值

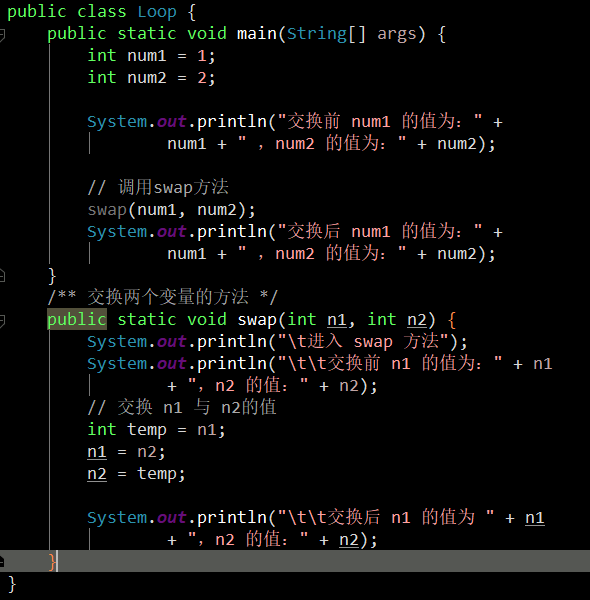
调用void方法的位置一定是在语句中

所以它被在main方法第三行以语句形式调用

1. **通过值传递参数**

调用一个方法时候需要提供参数，必须按照参数列表指定的顺序提供

说明

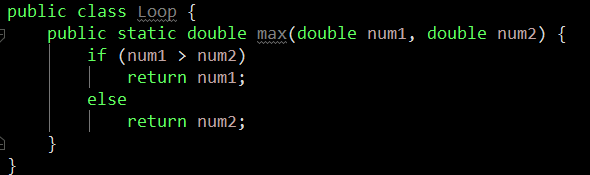


传递两个参数调用swap方法，但是方法被调用后，实参并没有改变

1. **方法的重载**

如果不清楚要传递什么类型的参数，可以建立多个同名但参数类型不同的方法

如下面代码所示：



如果调用max方法时：

* 传递的是int型参数，则int型参数的方法就会被调用；
* 如果传递的是double型参数，则double类型的max方法体会被调用

这种方法叫做重载

1. **构造方法**

当一个对象被创建时候，构造方法用来初始化该对象

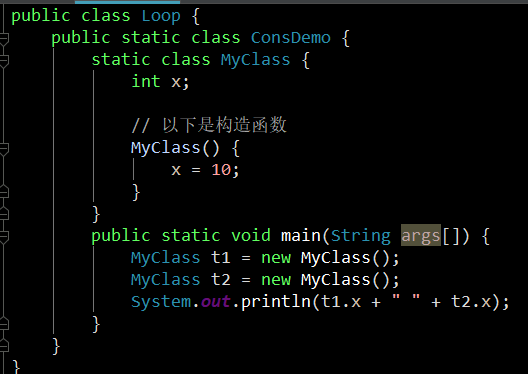
构造方法和它所在类的名字相同，但构造方法没有返回值

通常会使用构造方法给一个类的实例变量赋初值，或者执行其它必要的步骤来创建一个完整的对象

**默认构造方法的访问修改符和类的访问修改符相同：**

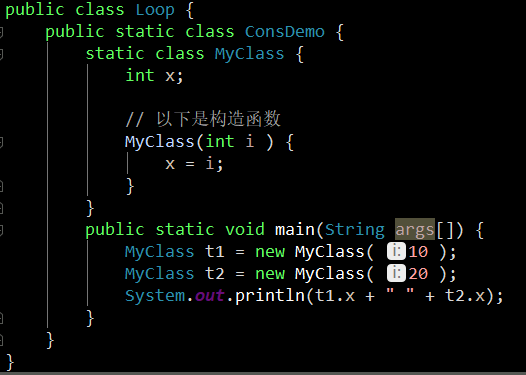
类为 public，构造函数也为 public；类改为 private，构造函数也改为 private

**使用构造方法来初始化一个对象：**



大多时候需要一个有参数的构造方法

**调用构造方法来初始化一个对象：**



初始化对象，并使对象实例化

**第二部分：修真任务**

**学习预览：**

* 重新认识IoC
* 依赖注入
* 设值注入
* 构造注入
* 接口注入

1. **重新认识IoC**
2. **IoC容器的概念**

IoC容器就是具有依赖注入功能的容器，IoC容器负责实例化、定位、配置应用程序中的对象及建立这些对象间的依赖。应用程序无需直接在代码中new相关的对象，应用程序由IoC容器进行组装。

1. **Spring IoC容器如何知道哪些是它管理的对象呢？**

Spring IoC容器通过读取配置文件中的配置元数据，通过元数据对应用中的各个对象进行实例化及装配。

一般使用基于xml配置文件进行配置元数据，而且Spring与配置文件完全解耦的，可以使用其他任何可能的方式进行配置元数据，比如注解、基于java文件的、基于属性文件的配置都可以。

1. **Bean的概念**

由IoC容器管理的那些组成你应用程序的对象我们就叫它Bean

Bean就是由Spring容器初始化、装配及管理的对象

1. **配置Spring环境**

**IDEA安装插件Spring Assistant，创建Spring的新web项目**

**然后依赖包下了很久，没来得及测试**

**明天开始操作吧**

1. **依赖注入**

**依赖注入是内置在IoC容器内的机制，不需要人为操作**

1. **设值注入**

通过这几天的学习，随着对基础知识理解的增加，也明白了设值注入的含义

在一个类中，我们传统地赋值方式，是直接用 变量+表达式 的方式赋值

而根据设值注入的思想，采用提前构建setter（）方法

随后调用setter（）方法给需要修改的对象属性进行修改

例如：



提前构建好setter（）方法，在随后的构建一个对象

想要给对象属性赋值，直接引用setter（）方法即可

1. 构造注入
2. 接口注入

**明天计划完成的事情：**

1. **JAVA基础知识的学习**

语法部分已经基本学完，该理解JAVA面向对象的概念和机制

1. **Spring框架的实操**

**遇到的问题：**

1. 之前对于JAVA核心语句，构建方法的概念很模糊

在今天学习完JAVA基本知识后，有了一定的认识

学会了规范的构建方法和引用方法的方式

**收获：**

基本学习完JAVA的基本语法知识

跟同期同学交流重新规划了学习spring的道路

**在理解了Spring核心机制的概念后，应该实操代码，通过代码间的逻辑来理解框架的运作机制，理论理解的部分可以暂时放缓。**