第二章 标示符、关键字、变量

1 注释

> 程序中的注释，用来说明某段代码的作用，或者说明某个类的用途、某个属性的含义、某个方法的功能，方法参数和返回值的数据类型、意义等。

注释可以增强代码的**可读性**，让自己或者他人快速的理解代码含义和设计思路，同时可以便于后期的对系统中代码的维护和迭代升级等。

Java源码代码中的注释，不会出现在字节码文件中，因为在编译的时候，编译器会忽略掉源代码中的注释部分。因此，可以在源代码中根据需要添加任意多的注释，而不必担心字节码文件会膨胀。

源代码文件：Xxxx.java

字节码文件：Xxxx.class

> 源代码中，被注释掉的部分（代码、文字等），不会对代码的编译、运行产生任何影响

java代码中的注释，分为三种：

1. 单行注释
2. 多行注释
3. 文档注释

1.1 单行注释

最常用的注释方式，其注释内容从 "//"开始到本行末尾。

例如，`Comment01.java`

```
//类名：Comment01
//作用：单行注释测试类
//作者：briup
public class Comment01{ //Comment01类的大括号开始

    //main方法，固定写法，程序入口
    public static void main(String[] args){
        //向控制台中输出指定字符串内容
        System.out.println("hello java comment");
        //下面一句代码被注释掉了，不会被执行
        //int a = 1;
    }
    //main方法结束

```

```
15    }//Comment01类的大括号结束
```

> 编译、运行一切正常，注释不会对此产生任何影响

1.2 多行注释

注释从 "/*" 开始，到 "*/" 结束。

它可以注释一行，也可以注释多行

例如，`Comment02.java`

```
1   package com.briup.day03;
2
3   /*
4       类名：Comment02
5       作用：多行注释测试类
6       作者：briup
7    */
8   public class Comment02{ /* Comment01类的大括号开始 */
9
10      /* main方法，固定写法，程序入口 */
11      public static void main(String[] args){
12          /* 向控制台中输出 */
13          System.out.println("hello java comment");
14          /* 下面一句代码被注释掉了，不会被执行 */
15          /* int a = 1; */
16      }
17      /* main方法结束 */
18
19  }/* Comment02类的大括号结束 */
20
```

多行注释不能嵌套，否则会报错

例如：下面是**错误**的写法，编译是会报错的。

```
1   /*
2       /*多行注释里面又嵌套了多行注释，编译报错*/
3   */
4   public void test(){
5
6   }
```

1.3 文档注释

可以注释单行，也可以注释多行，以 "/**" 开始，以 "*/"结束的。

同时在Java中，文档注释也被用于生成API文档。

如果在生成API文档的时候，希望得到更新详细的信息，例如方法参数、返回值、异常的详细说明，可以使用javadoc标记，常用的javadoc标记有：

- @author：作者
- @version：版本
- @deprecated：不推荐使用的方法、过时的方法。
- @param：方法的参数类型。
- @return：方法的返回类型。
- @see：用于指定参考的内容。
- @exception：抛出的异常。
- @throws：抛出的异常，和exception同义

例如：`Comment03.java`

```
1  package com.briup.day03;
2
3  /**
4   * 文档注释测试类,利用文档注释，结合javadoc命令，可以生成API说明文档
5   * @author briup
6   * @version 1.0
7   * @since JDK1.8
8   */
9  public class Comment03 {
10
11     /**
12      * main方法，程序入口，写法固定
13      * @param args  程序入口参数，如果需要，运行时可以给main方法进行传参
14      */
15     public static void main(String[] args){
16         System.out.println("hello java comment");
17     }
18
19 }
```

使用javadoc命令，根据Comment03中的文档注释和标记，生成API说明文档，并将生成的文档存放在API目录中：

```
javadoc -d api src/Comment03.java
```

如果要显示作者和版本的话，需要加上-author和-version选项，前提是代码中使用@author @version 俩个标记

```
javadoc -d api -author -version src/Comment03.java
```

在是Windows中，如果出现了中文乱码，可以加入-encoding UTF-8

```
javadoc -d api -author -version -encoding UTF-8 src/Comment03.java
```

> Windows中，命令窗口默认使用编码是GBK（中文版Windows）

> Linux中，命令窗口默认使用编码是UTF-8

例如，`Comment04.java`

```java
package com.briup.day03;

/**
 * 文档注释测试类,利用文档注释，结合javadoc命令，可以生成API说明文档
 * @author briup
 * @version 1.0
 * @since JDK1.8
 * @see com.briup.day03.Comment03
 */
public class Comment04 {

    /**
     *  类中的属性，表示问好语句的模式
     *  0 - hi xxx
     *  1 - hello xxx
     *  默认是模式为 1
     */
    public int mode = 1;

    /**
     * 返回向指定人问好的语句
     * @param name  用户姓名，向其问好
     * @return  问候的语句
     * @exception RuntimeException 运行时，mode值不是0或者1，会出现异常
     */
    public String sayHello(String name)throws RuntimeException{
        if (mode == 0){
            return "hi! "+name;
        }
        else if(mode == 1){
            return "hello! "+name;
        }
        else{
            throw new RuntimeException("error mode");
        }
    }

    /**
     * @deprecated 测试方式，已经废弃
     */
    public void test(){}

}

```

> 注意，这里@see标记还关联了com.briup.day03.Comment03

可以把src下面的多个java文件都生成对应的doc文件

```
javadoc -d api -author -version src/*.java
```

如果将来源代码分别放在不同的文件夹中：

例如，

com.briup.test1.Hello类对应的java文件中，存放在src/com/briup/test1/Hello.java

com.briup.test2.World类对应的java文件中，存放在src/com/briup/test2/Hello.java

注意，编译生成的class文件分别存放在bin/com/briup/test1/Hello.class 和 bin/com/briup/test2/World.class

这是，如果要生成所有类的doc文档，可以执行以下命令：这样可以把src下面com包下面所有的子包里面存放的java文件全都查找到并生成对应的doc文档

```
javadoc -d api -sourcepath src -subpackages com -author -version
```

> -sourcepath 指定源代码存放的位置
> -subpackages 指定要递归查找的包的名字

2 符号

2.1 分号

java中，一句代码都是以分号（;）来结束的。

需要使用分号的代码语句有：

- 包的声明语句
- 类的引入语句
- 属性的声明语句
- 方法中要执行的代码

包的声明语句：

例如，`package com.briup.test;`

类的引入语句：

例如，`import java.util.Date;`

属性的声明语句：

例如，

```
1   public class Student{
2       public String name;//声明name属性
3       public int age;//声明age属性
4   }
```

方法中要执行的代码：

例如，方法中几乎每行要执行的代码都要加分号（;）

```
public class test(){
    int a = 1;
    int b = 2;
    int c = a+b;
    System.out.println(c);
}
```

> java中进行解析执行代码的时候，就是根据代码后面的分号来确定是一句代码还是俩句代码；

例如，

```
public void test(){
    //虽然是在同一行，但是这里是俩句代码。
    int a = 1;int b = 2;
}
```

不需要加分号的代码有些哪些？

- 类的声明，最后【不需要】加分号

```
public class Student{}
```

- 方法的声明，最后【不需要】加分号

```
public class Student{
    public void test(){}
}
```

- 代码块的声明，最后【不需要】加分号

```
public class Student{
    {
        //这里是代码块
        //没有名字，就是一对大括号给括起来
    }
}
```

2.2 空白

> 在代码中，可以使用空格、tab、换行（\n）、回车（\r），并且对代码是没有影响的

例如，

```
public          class
    Student{

```

```
    public void test(){
        //这四句代码的效果作用是一样的
        int a1=1;
        int a2 = 1;
        int a3  =   1;
        int a4
        =
        1;
        System.
            out.
                println("hello");
        a.b()
            .c()
            .d()
            .e()
            .f();

    }

}
```

> 注意，不能使用这些空白来分割单词或关键字

例如，这个代码编译是报错的

```
pu  blic cla    ss  Hel
    lo{}
```

3 标示符

> 在java中，给类、方法、变量起的名字，就是标示符，因为它可以用来标识这个类、方法、变量

3.1 命名规则

- 标示符可以由字母、数字、下划线_ 、美元符号$组成
- 标示符开头不能是数字
- 标识符中的字符大小写敏感
- 标识符的长度没有限制
- 标示符不能使用java中的关键字或保留字

合法标示符	非法标示符
try1	try#
GROUP_1	1GROUP
helloworld	hello-world
_int	int
$int	$-int

3.2 推荐规则

- 类和接口，首字母大写，如果是俩个单词，第二个单词的首字母大写
  例如，public class Account{} ，public interface AccountBase{}
- 方法和变量，首字母小写，如果是俩个单词，第二个单词的首字母大写
  例如，public void getStudentName(){} ， int personNum = 1；
- 常量，全部字母大写，如果是俩个单词，使用下划线分隔
  例如，public static final int MAX_NUM = 10;
- 尽量使用有意义的名字，尽量做到见名知义。
  例如，int numOfStudnet = 10; String userName = "tom";

4 关键字

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
enum	extends	final	finally	float
for	goto	if	implements	import
instanceof	int	interface	long	native
new	package	private	protected	public
return	strictfp	short	static	super
switch	synchronized	this	throw	throws
transient	try	void	volatile	while

> 注意，const 和 goto 是java中的保留字

> 注意，true 和 false 不是关键字，是boolean类型的字面值，但是也**不能**直接使用true和false来做标示符

5 基本类型

> java中有八种基本数据类型，它们是java语言中，可以表示出来的最基本数据的结构。

这八种基本数据类型是：

- byte
- short
- int
- long
- float
- double
- char
- boolean

5.1 字节

计算机中，数据传输大多是以“位”（bit，比特）为单位，一位就代表一个0或1（二进制），每8个位（bit）组成一个字节（Byte），所以，1个字节=8位0101代码，例如 0000 0001

例如，0000 0001 ，表示二进制的数字1，它是1个字节，共8位0101代码组成

十六进制有0 1 2 3 4 5 6 7 8 9 A B C D E F ，它的范围是0~15
每4位0101代码可以表示一个十六进制的数字，因为4位表示的最小值是 0000，最大值1111 ，刚好范围是0~15
所以8位0101代码，刚好可以使用2位十六进制的数字来表示

例如，二进制的1111 1111 就可以使用 十六进制的 FF 来表示

5.2 boolean

> 布尔类型占1个字节（8位），它的的值，必须是true或者false，在JVM中会转换为1（true）或者0（false）

例如，

```
1   public void test(){
2       boolean f1 = true;
3       boolean f2 = false;
4   }
```

5.3 char

> char类型占2个字节（16位），用来表示字符，是基本数据类型，String表示字符串，是类类型。一个String是由0~n个char组成

例如，字符使用**单引号**表示，字符串使用**双引号**表示。

```
public void test(){
    char c1 = 'a';
    char c2 = 'b';
    char c3 = 'c';
    String str = "abc";
}
```

5.3.1 字符编码

Java语言对文本字符采用Unicode编码。由于计算机内存只能存取二进制数据，因此必须为各个字符进行编码。

例如:a --编码--> 0000 0000 0110 0001

5.3.2 常见编码

- ASCII
  ASCII--Amecian Standard Code for Information Interchange，美国信息交换标准代码。主用于表达现代英语和其他西欧语言中的字符。它是现今最通用的**单字节**编码系统，它只用一个字节的7位，一共表示128个字符。
- ISO-8859-1
  又称为Latin-1, 是国际标准化组织(ISO)为西欧语言中的字符制定的编码，用一个字节(8位)来为字符编码，与ASCII字符编码兼容。所谓兼容，是指对于相同的字符，它的ASCII字符编码和ISO-8859-1字符编码相同。
- GB2312
  包括对简体中文字符的编码，一共收录了7445个字符(6763个汉字+682个其他字符). 它与ASCII字符编码兼容。
- GBK
  对GB2312字符编码的扩展，收录了21886个字符(21003个字符+其它字符), 它与GB2312字符编码兼容。
- Unicode
  由国际Unicode协会编制，收录了全世界所有语言文字中的字符，是一种跨平台的字符编码。
  Unicode具有两种编码方案：
  - 用2个字节(16位)编码，被称为UCS-2, Java语言采用;
  - 用4个字节(32位)编码，被称为UCS-4;

  UCS(Universal Character Set)是指采用Unicode字符编码的通用字符集。
- UTF
  有些操作系统不完全支持16位或32位的Unicode编码，UTF(UCS Transformation Format)字符编码能够把Unicode编码转换为操作系统支持的编码，常见的UTF字符编码包括UTF-8、UTF-16、UTF-32。
  - UTF-8，使用一至四个字节为每个字符编码，其中大部分汉字采用三个字节编码，少量不常用汉字采用四个字节编码。因为 UTF-8 是可变长度的编码方式，相对于 Unicode 编码可以减少存

储占用的空间，**所以被广泛使用。**

- UTF-16，使用二或四个字节为每个字符编码，其中大部分汉字采用两个字节编码，少量不常用汉字采用四个字节编码。
- UTF-32，使用四个字节为每个字符编码，使得 UTF-32 占用空间通常会是其它编码的二到四倍。

5.3.3 编码表

每一种字符编码都有一个与之对应字符编码表。

例如，在Unicode编码表中十六进制的数字6136对应的汉字是憎，

```
char c = '\u6136';
System.out.println(c);
```

> 更多的Unicode编码，可以参考Unicode编码表

5.3.4 char值

例如，字符'a'，的表示形式：

二进制数据形式为　0000 0000 0110 0001
十六进制数据形式为　0x0061
十进制数据形式为　97

```
//使用具体字符来表示a
char c = 'a';
//使用Unicode编码值表示字符a
char c = '\u0061';
//0x开头的数字位十六进制，使用十六进制表示字符a
char c = 0x0061;
//使用十进制数字表示字符a
char c = 97;
//0开头的数字为八进制，使用八进制表示字符a
char c = 0141;

//注意:一个中文汉字就是一个字符
char c = '中';
```

5.3.4 转义字符

在给字符变量赋值的时候，通常直接从键盘输入特定的字符，而不会使用Unicode字符编码，因为很难记住各种字符的Unicode字符编码值。

但是对于有些特殊字符，比如一个单引号（'），如不知道它的Unicode字符编码，直接从键盘输入编译错误：

```java
//编译出错
char c = ''';

//为了解决这个问题，可采用转义字符（\）来表示单引号和其他特殊字符：
char c = '\'';
char c = '\\';
```

常用转义字符	效果
\n	换行符，将光标定位到下一行的开头
\r	回车，把光标移动到行首(和环境有关)
\t	垂直制表符，将光标移到下一个制表符的位置
\\	反斜杠字符
\'	单引号字符
\"	双引号字符

5.4 整型

> byte, short, int和long都是整数类型，并且都是有符号整数（正负）

- byte 8位、 1字节 范围:负2的7次方~2的7次方减1
- short 16位、2字节 范围:负2的15次方~2的15次方减1
- int 32位、4字节 范围:负2的31次方~2的31次方减1
- long 64位、8字节 范围:负2的63次方~2的63次方减1

有符号整数把二进制数的首位作为**符号数**，当首位是0时，对应十进制的正整数，当首位是1时，对应十进制的负整数。

在Java语言中，为了区分不同进制的数据，八进制数以“0”开头，十六制以“0x”开头,二进制以“0b”开头

```java
byte b1 = 97;           十进制
byte b2 = 0141;         八进制
byte b3 = 0x61;         十六进制
byte b4 = 0b01100001;   二进制

//都是97打印出来
System.out.println(b1);
System.out.println(b2);
System.out.println(b3);
System.out.println(b4);
```

整数类型的默认类型是int，对于给出一个字面值是99的数据，在没有指明这个数据是什么具体的类型的情况下，那么java默认认为是int类型。

```java
byte a = 1;
//编译报错
//a+1中a是byte类型,字面值1没有声明类型,那么默认是int
//byte是8位,int是32位,那么结果是32位的数字
//b只是一个byte类型的变量,那么只能接收8位的数字
//修改为int b = a+1;就可以编译通过
byte b = a+1;


//编译通过
//虽然1+1中的1都是默认的int类型
//但是这个俩个1都是固定的字面值
//编译器可以判断出其结果是否超出了byte表示的范围
//上面例子中a+1不能判断出结果的原因是:
//a是变量,是有可能发生变化的
byte c = 1+1;

//编译报错
//编译器判断出其结果超出了byte的表示范围(-128~127)
byte d = 1+127;

//编译报错
//原因:32位的数据赋值给byte类型的变量
//因为使用的1这些都是字面值,默认是int，注意关键点在在最左边的1，编译器不认为是符合位
//所以它默认是在前面补了24个0
byte e = 0b11111111;

//编译通过
//输出结果为255
//因为1的前面补了24个0
int e = 0b11111111;

//编译通过
//输出结果为-1
//因为这里做了类型强制转换
byte f = (byte)0b11111111;
```

> 注意，java中，等号（=）为赋值操作，表示把等号右边的值或计算结果，赋值给等号左边的变量

> 注意，java中，正数取反在加1，就是对应的负数，负数取反再加1，就是对应的正数

四种整型类型的声明：

```java
byte  a1 = 1;   (内存中占8位) 1字节
short a2 = 1;   (内存中占16位)2字节
int   a3 = 1;   (内存中占32位)4字节
long  a4 = 1L;  (内存中占64位)8字节
```

> 使用long类型数据的时候，后面要加大写L或者小写l,建议加上大写的L,因为小写的l和数字1很像似。

5.5 浮点型

> float和double都是java中的浮点型,浮点型可以用来表示小数，它们的二进制表示方式和整型不同

float是32位， 1符号位+8指数位+23尾数位
double是64位 1符号位+11指数位+52尾数位

float和double的精度是由尾数的位数来决定的。浮点数在内存中是按科学计数法来存储的.

- float的精度为7位左右有效数字
- double的精度为16位左右有效数字

俩种浮点型数据的声明：

```
//后面加f或者F
float f = 10.5f;
//后面加d或者D
double d = 10.5d;
```

浮点型的二进制形式：可以使用API提供的方法获取浮点数的二进制形式

```
float f = 10.5f;
int b = Float.floatToIntBits(f);
System.out.println(Integer.toBinaryString(b));
```

浮点型的默认类型是double,对于给出一个字面值是10.5的数据,在没有指明这个数据是什么具体的类型的情况下,那么java中默认认为是double类型。

例如，

```
//编译通过
//字面值1.5默认类型是double
double d = 1.5;
double d = 1.5D;

//编译报错
//字面值1.5默认类型是double
//double和float的精确度不一样
float f = 1.5;

//f2编译通过,因为字面值1的类型是int
//f3编译报错,因为字面值1.5的类型是double
float f1 = 10.5f;
float f2 = f1+1;
float f3 = f1+1.5;
```

浮点型的精度丢失：

例如，

```
System.out.println(1.0-0.66);
//输出结果: 0.33999999999999997
```

Java中的浮点数类型float和double不能够进行精确运算，虽然大多数情况下是运行是正常的，但是偶尔会出现如上所示的问题。

这个问题其实不是java语言的bug，而是因为计算机存储数据是二进制的，而浮点数实际上只是个近似值，所以从二进制转化为十进制浮点数时，精度容易丢失，导致精度下降。

要保证运行结果的精度，可以使用BigDecimal类：

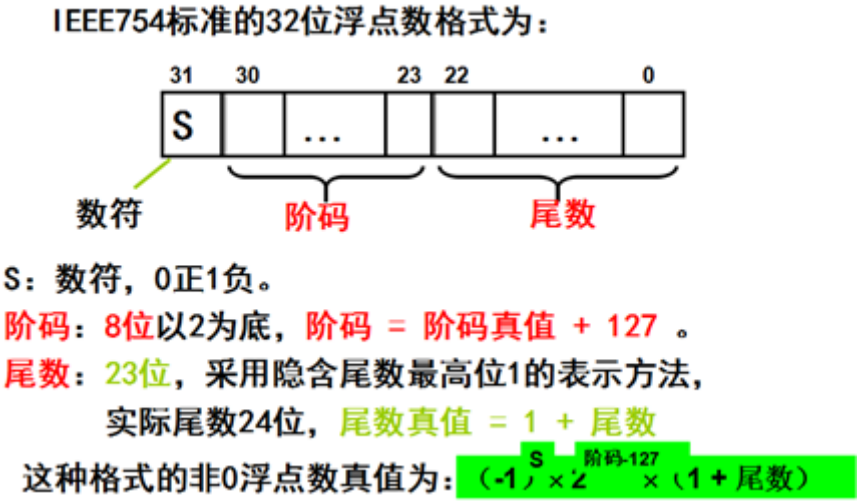
```
//add方法 +
//subtract方法 -
//multiply方法 *
//divide方法  /
BigDecimal d1 = BigDecimal.valueOf(1.0);
BigDecimal d2 = BigDecimal.valueOf(0.66);
double result = d1.subtract(d2).doubleValue();
System.out.println(result);
//输出结果:0.34
```

> BigDecimal是java.math包中的类，使用时需要import导入

整型、浮点型的位数、字节和表示范围：

Type	Bits	Bytes	Minimum Range	Maximum Range
byte	8	1	-2^7	$2^7 - 1$
short	16	2	-2^{15}	$2^{15} - 1$
int	32	4	-2^{31}	$2^{31} - 1$
long	64	8	-2^{63}	$2^{63} - 1$
float	32	4	not needed	not needed
double	64	8	not needed	not needed

浮点型数字的二进制结构（了解即可）



思考，32位的int和32位的float，谁表示的最大值更大？

6 变量

> java中的数据类型有八种基本类型，三种引用类型

八种基本类型：byte、short、int、long、float、double、boolean、char

三种引用类型：类类型、接口类型、数组类型

每一种类型都可以用来声明变量，而变量的作用就是用来接收、保存、传递、操作对应的数据的。

例如，有一个数字1，在程序中该如何接收、保存、操作、传递？

```
1   public void test(){
2       //声明int类型的变量a
3       int a;
4
5       //使用变量a接收数据1，其实也是把1赋值给变量a
6       a = 1;
7
8       //使用变量a进行操作，并把操作结果传给（赋值给）变量b
9       int b = a+a;
10
11  }
```

> 体会程序中，变量的作用

变量一定是要求先声明、再赋值、再使用

```java
int a;//声明变量
a = 1;//给变量赋值
System.out.println(a);//使用变量的值

int b = 1;//声明变量的同时就进行赋值
System.out.println(b);//使用变量的值

```

java是强类型编程语言，要求数据的类型和变量的类型保持一致，才能使用 = 号进行赋值，就是把=号右边的数据，赋值给=号左边的变量。

例如，`long a = 1L;`

如果=号右边的数据，和=号左边的变量类型不一致，那么只能做类型转换，把数据的类型转为变量的类型，然后才能完成=号赋值操作。

只是类型转换的过程，分为**手动转换**和**自动转换**：

1. 手动转换

```java
int a = 1;
byte b = (byte)a;
```

2. 自动转换

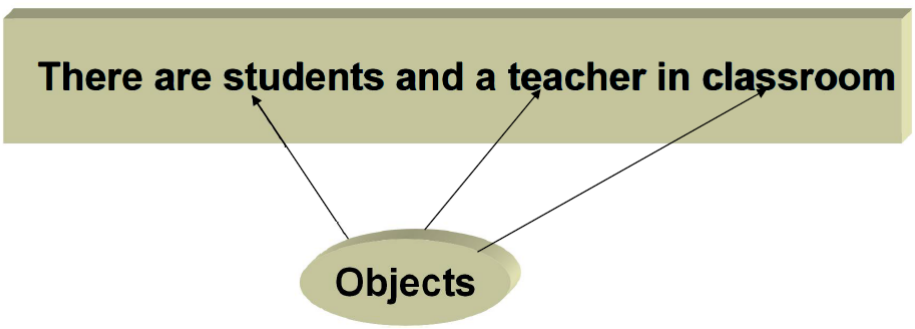
```java
int a = 1;
long b = a;
```

> 思考，手动类型转换和自动类型转换分别有什么特点？

7 理解对象

> java是面向对象的编程语言，在面向对象的思想中，万事万物皆为对象。

对象无处不在，在描述一个场景的时候，大多数情况下，里面的名词表示的就是对象：



对象一般都会具有一些属性和方法

- 属性表示对象本身的一些特点
- 方法表示对象本身的一些行为

例如，

- 学生有身高、体重等属性，学生还有学习、运动等方法
- 老师有年龄、专业等属性，老师还有上课、布置作业等方法
- 教室有位置、大小等属性，教室还有添加学生、添加老师等方法

代码中如何获得对象？

第一步：抽象一种事物，来定义出需要的类

例如，定义一个类，用来描述学生对象应该具有的属性和方法
其实就是　根据学生的特点和行为 --抽象--》 Student类

```
package com.briup.day03;

/**
 * 学生类，描述学生应该具有的属性和方法
 */
public class Student {
    //学生编码
    public long id;
    //学生姓名
    public String name;
    //学生年龄
    public int age;
    //学生要说的话
    public String msg;

    //构造方法，用来创建一个具体的学生对象
    public Student(long id, String name, int age) {
        this.id = id;
        this.name = name;
        this.age = age;
    }

    /**
     * 来吧，展示
     * 先介绍自己的名字、学号和年龄
     * 再说出自己想说的话
     */
    public void show(){
        System.out.println("I am "+name);
        System.out.println("My studentNum is "+id);
        System.out.println("I am "+age);
        System.out.println(msg);
    }

}

```

第二步：使用类中的构造方法，创建出这个类的对象

```java
package com.briup.day03;

/**
 * 学生类，描述学生应该具有的属性和方法
 */
public class Student {

    //... 这里的代码和上面一样，这里省去不写了

    //比上面代码多了一个main方法
    public static void main(String[] args) {
        //什么一个Student类型的变量
        Student stu;
        //new+构造器的方式，创建一个Student类具体对象
        //并且把这个新创建出来的对象，赋值给了stu变量
        stu = new Student(1L,"tom",20);
    }

}

```

> 上面代码也可以这样写：Student stu = new Student();

> Student类中的构造，和new关键字一起使用，就可以创建出一个Student类的对象

第三步：使用对象访问它的属性并赋值，使用对象调用类中定义的方法

```java
package com.briup.day03;

/**
 * 学生类，描述学生应该具有的属性和方法
 */
public class Student {

    //... 这里的代码和上面一样，这里省去不写了

    //main方法
    public static void main(String[] args) {
        //声明一个Student类型的变量
        Student stu;
        //new+构造器的方式，创建一个Student类具体对象
        //并且把这个新创建出来的对象，赋值给了stu变量
        stu = new Student(1L,"tom",20);

        stu.msg = "可爱的朋友们，你们好么~";
        stu.show();

    }

}

```

> 这时候就完成了：
>
> 分析事物--》定义类--》声明属性/方法/构造器--》编写main方法--》创建对象--》使用对象访问属性、调用方法 --》完成功能（来吧，展示）

8 引用类型变量

使用八种基本类型中的任意一种类型，声明出的变量，就是基本类型变量，例如 int a;

使用三种引用类型中的任意一种类型，声明出的变量，就是引用类型变量，例如 Student stu;

基本类型变量和引用类型变量的区别：

- 基本类型变量**只能**接收基本类型数据，它是一种比较简单的数据，没有属性，也没有方法
- 引用类型变量**只能**接收引用类型数据（也就是对象），但对象是一种比较复杂的数据，它里面可以有很多属性，也可以有很多方法

> 引用类型变量，简称为引用，它可以用来接收对象，也叫做引用指向对象。

例如：

```
//a中没有任何属性和方法，它在这只是表示了一个很简单的数据1
int a = 1;

//stu接收了一个对象，它里面有属性和方法
Student stu = new Student(1L,"tom",20);
//使用stu可以访问属性并赋值
stu.msg = "hello world";
//使用stu可以调用方法
stu.show();

```

> 可以看出：
> 基本类型变量只能接收基本类型数据，只能表示最简单数字（8/16/32/64位）
> 引用类型变量只能接收引用类型数据（对象），可以访问属性、调用方法，完成比较复杂的功能
>
> 基本类型变量和引用类型的变量核心区别：是否可以指向对象