第2章 基本程序设计

- ✦2.1标识符和关键字
- ✦2.2 注释
- ✦2.3 变量和常量
- ✦2.4 基本数据类型
- ✦2.5 类型转换
- ✦2.6 运算符与表达式
- ✦2.7 数组

学习目标

- 掌握Java标识符的定义规则
- 掌握Java基本数据类型和数组的使用
- 掌握类型转换方法
- 掌握各种运算符的使用和它们在表达式中的优先关系

总结Java程序组成

- Java 程序总是由一个或多个类/接口组成
- 类/接口中定义：数据+方法
- 数据涉及：变量+常量+类型
- 方法体：流程控制语句&表达式
- 表达式=数据+运算符

标识符

- 程序对各种常量、变量、方法、类等元素的命名符号统称为标识符
- 标识符的命名规范：
  - (1) 标识符可由字母、数字、下划线(_)、美元符($)组成
  - (2) 首字符为字母、美元符或下划线，不能为数字
  - (3) 用户自定义的标识符不能为Java关键字
  - 标识符遵守先定义后使用的原则，且区分大小写

Java关键字(了解)

分类		关键字
类型相关	数据类型	boolean、byte、short、int、long、char、float、double、class、interface、enum、var
	引用类型扩展	extends、implements
	对象引用	this、super
	对象创建	new
	对象判断	instanceof
	返回类型	void
修饰符	访问控制	private、protected、public
	封装特性	final、abstract、static
	浮点精度控制	strictfp
	本地方法	native
	序列化	transient
	多线程	synchronized、volatile
流程控制	分支	if、else、switch、case、default
	循环	for、do、while、break、continue
	异常处理	try、catch、finally、throw、throws
	其他	assert、return
包相关		import、package
保留		const、goto

标识符举例

- 合法用户标识符：
  - age
  - $salary
  - _value
  - 元素　　不推荐
- 非法用户标识符：
  - 123abc　　首字符为数字，不合规
  - -salary　　首字符不合规，且含非法字符(-)
  - break　　关键字
  - two words　　含非法字符空格

命名规范(骆驼规则/驼峰规则)

- 编程人员还应遵守一些惯例和约定(非强制性但建议遵守)
  - 包名
    - 多单词组成时，所有字母都小写，如：com.sun
  - 类名/接口名
    - 多单词组成时，所有单词首字母大写，如：RecordInfo
  - 变量/方法名
    - 第一个单词首字母小写，其余单词首字母大写，如：recordName
  - 常量名
    - 所有字母都大写，用下划线连接，如： MAX_VALUE
  - $符号
    - 一般只用于编译器自动生成的代码，用户一般不用

注释

- 注释用来对程序中的代码做出解释
- 注释的内容在程序编译时不参与编译，因此，注释部分的有无对程序的执行不产生任何影响

Java有三种类型的注释

- // 单行注释
  - 表示从此向后，直到行尾都是注释
- /*……*/ 多行注释/块注释
  - 在“/*”和“*/”之间都是注释
  - 并可跨越多行，最后用一个“*/”结束
  - 块注释不能嵌套
- /**……*/ 文档注释
  - 所有在“/**”和“*/”之间的内容可以用来自动形成文档
  - JDK提供的javadoc.exe工具可以直接将源代码里的文档注释提取成一份系统的API文档，并输出为一个HTML文件

文档注释位置

- 文档注释在单个文档中有三类位置：类前、方法前、成员变量前。而写在其他位置，比如函数内部，是无效的文档注释

```
/**
  类的文档注释
*/
public class ExChapter1_1 {

    /** 字段的文档注释 */
     public String s="test";

    /** 方法的文档注释*/
    public void show(){
        System.out.println(s);
}}
```

注意：

- 默认情况下，javadoc只对访问修饰符为公有(public)和受保护(protected)的成员产生文档

文档注释内容

- 格式：

```
/**
 * 描述部分(description)
 * 块标签部分(block tags @标签)
 */
```

- 例如：

```
/**
 * Provides the classes necessary to create an framework object
 * <p>
 * This is an embeddable window (see the{@link java.awt.Panel} * class)
 * @since 1.0
 * @deprecated  replaced by {@link #setBounds(int,int,int,int)}
 */
```


常用@标签(了解)

标签	描述
@author	标识一个类的作者,也可包含电子邮件地址或其他任何适宜信息
@version	类的版本号，没有特殊格式要求
@param	说明一个方法的参数，格式：@param 参数名称 参数描述
@return	说明方法的返回值，格式： @return 返回值描述
@see	引用其他类，格式：@see classname
@deprecated	指明一个过期的类或成员
@throws 或 @exception	标志一个类显式抛出的检查异常(Checked Exception)
{@link}	插入一个到另一个主题的链接。格式：{@link package.class#method}

标签举例

```
import java.io.*;
/**
 * 这个类演示了文档注释
 * @author zhang
 * @version 1.0
 */
public class Chapter1_2 {
  /**
   * This method returns the square of num.
   * @param num The value to be squared.
   * @return num squared.
   */
  public double square(double num) {
      return num * num;
  }
```

```
  /**
   * This method inputs a number from the user.
   * @return get the input from the command line.
   * @exception IOException .
   * @see IOException
   */
  public double getNumber() throws IOException {
    InputStreamReader isr =
            new InputStreamReader(System.in);
    BufferedReader inData =
            new BufferedReader(isr);
    String str;
    str = inData.readLine();
    return (new Double(str)).doubleValue();
  }
}
```

用javadoc.exe为Chapter1_2.java生成帮助文档的命令格式：
javadoc -d .\api Chapter1_2.java

javadoc生成的文档(举例)

变量

- 变量是指在程序中可以被改变的值
- Java是静态类型的编程语言，因此变量应先声明(即指定类型)后使用
- 变量的声明形式：
  类型名 变量名1[=初始值1][,变量名2[=初始值2]……];
- 举例：

```
int  a;                    //声明单变量
int  b,c,d;                //一次声明多个变量
char ch1='a', ch2, ch3; //部分赋初值
```


常量

- 常量是指在程序运行过程中一旦被赋值，其值不变的量
- Java常量分为符号常量和字面常量
  - (1) 字面常量(普通常量)
    - 字面常量无需声明直接使用，如：3.14、'a'、true
  - (2) 符号常量
    - 用final关键字定义的标识符
    - 定义格式：

final 类型名 常量名1[=初始值1][,常量名2[=初始值2]…];

- 举例：

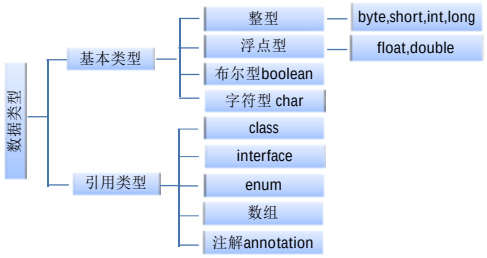
```
final int YOUTH_AGE;     // 声明一个int型常量
final float PIE;         // 声明一个float型常量
```


2.4 基本数据类型

- 2.4.1 整型
- 2.4.2 浮点型
- 2.4.3 布尔型
- 2.4.4 字符型

Java数据类型

- Java是一种强类型语言，每个变量都需要声明为一种类型
- Java数据类型包括两大类：基本数据类型和引用类型。



基本数据类型

数据类型	占内存空间	数值范围
byte	1字节	-2^7~2^7-1（-128~127）
short	2字节	-2^{15}~2^{15}-1（-32768~32767）
int	4字节	-2^{31}~2^{31}-1（-2147483648~2147483647）
long	8字节	-2^{63}~2^{63}-1（-9223372036854775808~9223372036854775807）
float	4字节	-3.4028347E+38~3.4028347E+38
double	8字节	(-1.79769313486231570E+308)~(1.79769313486231570E+308)
boolean	依赖拟机的实现1字节或4字节	true或false
char	2字节	'\u0000'~'\uFFFF'

整型

- 整型常量
  - 整型字面常量：用数值结合前缀和后缀表示值的大小，前缀表示进制、后缀表示类型，如果未标明，默认为int型的十进制数
    - 进制前缀
      - (1)0：八进制，如012
      - (2)0x或0X：十六进制，如0X12
      - (3)0b或0B：二进制整数，如0b101
      - (4)无：默认十进制，如12
    - JDK7开始支持用下划线分隔多个数位，如0b1100_1010_1011
  - 类型后缀：l或L：表示long型，如12L
  - **注意**：没有单独表示byte型和short型字面常量的方法

整型

- 整型变量
  - 整型变量赋值时需要注意如下几点：
    - (1) 大类型的变量可以接收小类型的常量，反之不可。如：
      - long val2=100 // 正确
      - int val1=100L // 错误
    - (2)没有byte和short型字面常量，允许将int型常量赋给它们，但所赋常量不能超过变量的相应范围，如：
      - byte b1=100 // 正确
      - byte b2=130 // 错误，超过了2^7-1=127
    - (3)值不要超过类型表示范围，如：
      - int val3=12345678900 // 错误，值常量超过int型上界 // 2147483647

浮点数

- 浮点表示法利用指数使小数点的位置可以根据需要而左右浮动，从而灵活地表达更大范围的实数
- 浮点数所表示的数因为有小数部分，所以用二进制表示是有误差的

浮点数常量

- 类型后缀
  - 缺省的浮点型常数是double 型(用D或d表示)，若表示float型，要加后缀F或f
  - 不能带表示进制的前缀，例如：
    - 5.1F、4f //正确，单精度
    - 5.1、.4 //正确，双精度
    - 0X12.3 //错误

浮点数常量

- 科学计数法表示，格式：[±][0x|0]尾数E|P[±]指数 [f|d]
  - 尾数可以是整数或小数
  - 前缀0X表示十六进制，即使小数也可以带0X前缀
  - 前缀0是前导零而并非八进制
  - E|P表示底数，E表示10为底，P表示2为底
    - 当尾数为十进制时底数只能用E
    - 当尾数为十六进制，只能用P(因E是十六进制数中的一个数字)
  - 指数必须是十进制整数，可带前导零
  - f|d表示类型后缀

浮点数常量

- 科学计数法表示举例
  - 2E3F　　//2 ×10³,且是float类型
  - -2.3E04　　//-2.3 ×10⁴,且是double类型
  - 2.3E4.5　　//非法，指数不能是小数，
    //只能是十进制整数
  - 0x2E3F　　//十六进制整数11839(不是浮点数)，
    //因为，如果尾数是十六进制则底数只
    //能用P，否则因E是十六进制中的一个
    //数，会混淆
  - 12.3P4　　// 非法，尾数为十进制时，底数只能用E
  - 0X1.2P-3　　// 0.140625，double类型

浮点数变量

- 浮点变量赋值
  - float f1=12.34　　//错误，将高精度的double类型值赋
    //给低精度的float类型
  - double d1=12.34 , d2=12.34f　　//正确

布尔型

- 布尔常量 ：
  - 两个值： true和false
  - 不对应整数值，因此也不能在布尔类型和整型之间进行类型转换
    - 这在程序设计中避免了潜在的逻辑错误
- 布尔变量赋值：
  - boolean a=true, b=false;

字符型

- Java字符集为16位的Unicode编码
  - 用\u开头的4位十六进制数表示
  - 范围：'\u0000'~'\uFFFF'
- 字符常量
  - 常规表示
    - 用单引号直接括住单个字符，如'A'、'北'
  - 转义序列表示
    - 转义序列由反斜杠（\）后面加上一个字符或者一些数位组成

2.4.4字符型

- 转义字符

字符	转义字符	Unicode码	说明
'	\'	\u0027	西文单引号
"	\"	\u0022	西文双引号
\	\\	\u005c	反斜杠
回车	\r	\u000d	Return键
换行	\n	\u000a	
退格	\b	\u0008	
制表符	\t	\u0009	
拉丁字符	\ddd		八进制表示
任意字符	\uxxxx		十六进制表示

字符在内存中存储为字符的Unicode编码，是一个整型数

字符型

- 字符变量
  - 字符变量可以接收字符常量，也可以接收字符的Unicode整型编码
  - 例：
    - char ch1='a' ;
    - char ch2='\u0061' ;
    - char ch3='\141' ;
    - char ch4=97 ;　　//字符'a'的Unicode码值

上述表达式都是给字符变量赋予字符'a'

字符串

- 字符串常量：两个双引号之间的字符序列
- 例：
  - "hello world"、"中国"、"welcome \n xxx"
- Java语言把字符串常量当作String类型的一个对象来处理，不是基本数据类型

字符集(了解)

- ASCII字符集
- GB2312/GBK字符集
- Unicode(Universal Code 统一码)字符集，编码实现方式
  - UTF-8(长度可变,1-3字节，兼容ASCII)
  - UTF-16
- 源文件---------- 字节码----------class文件被执行

（源码字符转成Unicode编码）　　Unicode编码的字符转换成所在系统所设置的编码

2.5 类型转换

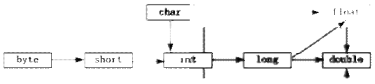
- ✦2.5.1 自动转换
- ✦2.5.2 强制转换

各类型数据间优先级及相互转换

- 基本数据类型会自动向高的数据类型转换(也为隐式转换)
- 高类型向低类型转换，需要通过强制转换：
  - 格式：(类型名)转换数值
  - 例：(int)3.14

自动类型转换

- 整型、实型、字符型数据可以混合运算
- 原则：当不同类型的数据进行混合运算时，低精度的数先转换成高精度同一类型，再进行运算



- 注意：
  - 在算术运算过程中，运算结果至少是int型，即如果参与运算的两个数级别比int型低或是int型，则结果为int型
  - int类型的常量赋给byte、short变量时不需要强制类型转换，编译器帮助完成强制转换
  - 例：byte b=123;
    short s=123;

强制类型转换

- 容量大的数据要转换成容量小的数据需用到强制类型转换，这种转换有时会丢失一些信息(失真)
- 格式：(类型名)变量/表达式
- 例：

```
double x=3.7;
int cx=(int) x ;     //cx的值为3
```

强制类型转换

- 注意：
  - (1) 强制类型转换可能造成信息的丢失（失真）
  - (2) 布尔型与其它基本类型之间不能转换
  - (3) 把int类型的变量赋给byte、short类型的变量时必须强制转换，否则会出错
  - 例：
    - int i=123; byte b=123 ;
    - byte b=i; //错误，i为int型变量，而不是常量
    - b=b+3 //错误，b+3的结果为int型临时中间变量