1. 基本类型：

程序的基本功能是处理数据
程序用变量来表示数据；
程序中必须能使用；
定义变量是指设定变量的数据类型和变量的名字，定义变量的基本语法为：

数据类型 变量名；

Java语言把数据类型分为基本类型和引用类型。

基本类型　　数值类型　　浮点数类型
位数
boolean 布尔类型　1位
byte(8位)<short=char(16位)<int=float(32位)<double=long(64位)
窄范围转换成宽范围,自动转换
宽范围转换成窄范围,需要强制类型转换

二进制数(仅4位的2进制数)=　十进制数　=　16进制数

0000	=	0	=	0
0001	=	1	=	1
0010	=	2	=	2
0011	=	3	=	3
0100	=	4	=	4
0101	=	5	=	5
0110	=	6	=	6
0111	=	7	=	7
1000	=	8	=	8
1001	=	9	=	9
1010	=	10	=	A
1011	=	11	=	B
1100	=	12	=	C
1101	=	13	=	D
1110	=	14	=	E
1111	=	15	=	F

1*2^0+1*2^1+1*2^2+1*2^3 =15
十六进制表示方法：0xff 15*16^1+15*16^0=255(max) 0x00 = 0(min)
11111101101001011001101 1（这是一个2进制数)
转换成十六进制,十六进制再转换成十进制
先把它所包含的数字分成4个4个在一块，如下所示：
1111 1101 1010 0101 1001 1011
根据上述常用表可以得到
1111=F
1101=D
1010=A

0101=5
1001=9
1011=B
那么它所对应的16进制数就是“0xFDA59B”

2. boolean类型

位置	boolean类型变量取值
Java源程序	只能是true或false
class文件	用int或byte表示boolean 0 非0
虚拟机中	用整数0来表示false, 有任意一个非零整数表示true

强调，在Java源程序中不允许把整数或null赋给boolean类型的变量，这是有别于其它语言(如c语言)的地方. 例：

```
boolean isMarried = 0;        //编译出错，提示类型不匹配
boolean isMarried = null;     //编译出错，提示类型不匹配
```

3. 文本数据类型——char和String

1) 字符编码

Java语言对文本字符采用Unicode字符编码。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。

所谓字符编码，是指用一串二进制数据来表示特定的字符。常见的字符编码包括：

a. ASCII字符编码

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。

b. ISO-8859-1字符编码

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。

c. GB2312字符编码

它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。

d. GBK字符编码 windows系统

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。

e. Unicode字符编码：

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。

UCS(Universal Character Set)是指采用Unicode字符编码的通用字符集。

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;

f. UTF字符编码 unix操作系统使用

有些操作系统不完全支持16位或32位的Unicode字符编码，UTF(UCS Transformation Format)字符编码能够把

Unicode字符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-7和UTF-16.

2) char的几种可能取值

Java语言采用UCS-2字符编码，字符占2个字节(1个字节8位)

字符a的二进制数据形式为 0000 0000 0110 0001

十六进制数据形式为 0x0061

十进制数据形式为 97

以下4种赋值方式是等价的：

```
char c = 'a';
char c = '\u0061';     //设定"a"的十六进制数据的Unicode字符编码
char c = 0x0061;       //设定"a"的十六进制数据的Unicode字符编码
char c = 97;           //设定"a"的十进制数据的Unicode字符编码
```

3) 转义字符

Java编程人员在给字符变量赋值时，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。

对于有些特殊字符，比如单引号，如不知道它的Unicode字符编码，直接从键盘输入编译错误：

```
char c = ''';          //编码出错
```

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 = '\\';
```

转义字符以反斜杠开头，常用转义字符：

\n　　换行符，将光标定位到下一行的开头；
\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

4）类型转换
char—>int 直接转换
e.g
char a = 'a' int i = a 此时i的值就是a字母对应的是十进制的值
但是 如果String—>int 不能直接转换 需要使用Integer.parseInt(String)方法进行转换
即使使用方法转换成功,如果String本来对应的值就不是数字类型 ,转换过去也不是数字类型
e.g
String s = "123" int ii = Integer.parse(s) ii时整数
String s = "name" int ii = Integer.parse(s) ii此时不是数字类型返回false

4. 整数类型

源码 第一位是符号位 其他位是真值位 可以求出具体十进制的二进制表达式
反码 源码 除去符号位保留不变 其他位按位取反获得的二进制表达式
补吗 反码+1得到的二进制表达式
内存中按照补吗进行保存
正数 源码 反码 补吗相同
负数 源码—>取反(反码)+1—>补吗

二进制(原码)	1000 1010
二进制(反码)	1111 0101
二进制(补码)	1111 0110
10进制	–10

不管啥 正数变负数 负数变整数 取反+1就完事了

---
byte b = (byte)129;
按照数字推算出的二进制表达形式是源码
129　00000000000000(24位) 1000 0001 (源码－－－》补吗)
1000 0001 (补吗－－->源码)
取反+1 1111 1111(源码 -127)
---
byte b = (byte)-129
源码：1000000000000000(24位) 1000 0001
补码: 1111111111111111(24位) 0111 1111
0111 1111(正数补码　源码)

无符号:0~~~255
0000 0000~~~~~1111 1111
. 无符号整数把二进制数的所有位转换为正整数。对于一个字节的二进制数, 它对应的十进制数的取值范围是0 - 255。

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。举例：

一个字节的二进制数	八进制数	十六进制数	有符号十进制数	无符号十进制数
0000 0000	0000	0x00	0	0
1111 1111	0377	0xFF	-1	255
0111 1111	0177	0x7F	127	127
1000 0000	0200	0x80	-128	128

如果一个整数值在某种整数类型的取值范围内，就可以把它直接赋给这种类型的变量，否则必须进行强制类型的转换。

二进制转十进制
Integer.valueOf("0101",2).toString()
另外还有
十进制转成十六进制：
Integer.toHexString(int i)
十进制转成八进制
Integer.toOctalString(int i)
十进制转成二进制
Integer.toBinaryString(int i)
十六进制转成十进制
Integer.valueOf("FFFF",16).toString()

八进制转成十进制
Integer.valueOf("876",8).toString()

5. 浮点类型

默认情况下,如果直接构建带有小数的数字,表示的是double类型
float接受需要进行强制类型转换.
e.g float f = (float)10.0;
double d1 = 1000.1;

如果把double类型的数据直接赋给float类型变量，有可能会造成精度的丢失，因此必须进行强制类型的转换，否
则会导致编译错误，例如：

```
float f1 = 1.0                  //编译错误，必须进行强制类型转换；
float f2 = 1;                   //合法，把整数1赋值给f2，f2的取值1.0;
float f3 = (float)1.0;          //合法，f3的取值为1.0;
float f4 = (float)1.5E+55;      //合法, 1.5E+55超出了float类型的取值范围,
                                f4的取值为正无穷大
System.out.println(f3);         //打印1.0;
System.out.println(f4);         //打印Infinity

      Float.NaN                 非数字
Float.POSITIVE_INFINITY         无穷大
Float.NEGATIVE_INFINITY         负无穷大
float f1 = (float)(0.0/0.0);    //f1的取值为Float.NaN
float f2 = (float)(1.0/0.0);    //f2的取值为Float.POSITIVE_INFINITY
float f3 = (float)(-1.0/0.0);   //f3的取值为Float.NEGATIVE_INFINITY
System.out.println(f1);         //打印NaN;
System.out.println(f2);         //打印Infinity
System.out.println(f3);         //打印-Infinity
```

Java语言之所以提供以上特殊数字， 是为了提高Java程序的健壮性，并且简化编程。当数字运算出错时，可以用浮
点数取值范围内的特殊数字来表示所产生的结果。否则，如果Java程序在进行数学运算遇到错误时就抛出异常，会影
响程序的健壮性，而且程序中必须提供捕获数学运算异常的代码块，增加了编程工作量。

---

6. 变量的申明和赋值

final public static int MAX_ROW = 100;

static修饰变量一开始就被加载到内存,不依赖当前类,称之为静态变量

7. 推荐命名规则

1) 类名以大写字母开头；
2) 接口名以大写字母开头；
3) 方法名以小写字母开头；
4) 变量名以小写字母开头；
5) 常量名全部大写，多个单词以"_"连接；

8. 理解对象

面向对象的开发方法把软件系统看成各种对象的集合，对象就是最小的子系统，一组相关的对象能够组合成更复杂的
子系统。面向对象的开发方法将软件系统看成各种对象的集合，接近人的自然思维方式。

对象是对问题领域中事件的抽象。对象具有以下特性：

1) 万物皆为对象。问题领域中的实体和概念都可以抽象为对象。例如学生，成绩单、教师、课和教室。
2) 每个对象都是惟一的。正如世界上不存在一模一样的叶子。
3) 对象具有属性和行为。
例如小张，性别女，年龄22，身高1.6m, 体重40kg, 能够学习，唱歌。小张的属性包括姓名、性别、年龄、身高和
体重，行为包括学习、唱歌。
例如一部手机，牌子是诺基亚、价格是2000元，银白色，能够拍照、打电话和收发短信等。这部手机的属性包括品
牌类型type、价格price和颜色color，行为包括拍照takePhoto()，打电话call()，收发短信receiveMessage()和发短
信sendMessage().
4) 对象具有状态。状态是指某个瞬间对象的各个属性的取值。对象的某些行为会改变对象自身的状态，即属性的取值。
例如小张本来体重为40kg，经为减肥后，体重减到35kg.

肥胖状态: 40kg
|
| 减肥行为

|

肥胖状态: 35kg

5) 每个对象都是某个类的实例。小张和小王都属于学生类、中国和美国都属于国家类、中文和英文都属于语言类。

类是具有相同属性和行为的对象的集合。

同一个类的所有实例都有相同属性，但属性取值不一,事实上相同，但是它们的状态不一定相同。例如小张和小王都属

于学生类，都有姓名、性别、年龄、身高和体重这些属性，但是他们的属性取值不同。

同一个类的所有实例都有相同行为，意味着它们具有一些相同的功能。

9. 创建类

类是一组具有相同属性和行为对象的模板。面向对象编程的主要任务就是定义对象模型中的各个类。

```
package sample;

public class Teacher {

   /**attributes of a teacher*/
   private String name;
   private int age;
   private double salary;

    /** Creates a new instance of Teacher */
   public Teacher(String name, int age, double salary) {
      this.salary = salary;
      this.age = age;
      this.name = name;
   }
   /**operations on properties */
   /** get the name of this teacher */
   public String getName() { return name; }
   /**get the salary of this teacher */
   public double getSalary() { return salary; }
   /**get the age of teacher teacher */
   public int getAge() { return age; }
   ……
}
```

代码解析：

```
public String toString() {
        return “name:”+name+” age:”+age+” gender:“+gender;
}
```

toString方法:返回用户指定输出的字符串格式
可以使用class引用.toString()调用
如果直接写class引用实际也是调用toString()方法.
即使没有构建toString(),class引用默认也是调用
toString()方法.默认的toString()方法输出
当前对象位于的内存地址
如果自己构建toString()方法,执行自己的toString()方法
如果自己没有构建toString()方法,执行系统自带的默认toString()方法.
关键字this:表示当前类本身,输出toString()方法

e.g

Student student = new Student();

该表达式在内存中分为两步走
Student student; 在栈区的构建
new Student();在堆区的构建
=:栈区的引用指向堆区的地址
创建对象类型 对象的引用(任意指定) = new(创建) 对象的构造方法:真正创建对象

无参数的构造方法，类自动构建
对象的引用.属性 可以调用属性(赋值 取值)
对象的引用.方法 可以调用方法(处理数据)
对象构建:通过使用new 可以构建对象.new 对象的构造方法

10. 创建实例

类创建好之后，通过new关键字创建具体对象。它有以下作用：
. 为对象分配内存空间，将对象的实例变量自动初始化为其变量类型的默认值；
. 如实例变量显示初始化，将初始化值赋给实例变量；
. 调用构造方法；
. 返回对象的引用；
Student student(对象引用) = new Student()(创建对象);

11. 程序用变量来表示数据；程序中必须先定义变量才能使用；

成员变量位于堆空间,有默认值

. 局部变量：局部必须初始化.jvm不会默认初始化 。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
中变量的可以被代码块使用,但是代码块的变量不能被方法以及其他方法使用.
局部变量可以作用域包括方法内部的代码块中,而代码块中变量
的作用域只能是当前代码块中,即使是包含这个代码块的方法
也不能使用.
局部变量位于栈空间

. 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,类似于局部变量

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；

```
public void method1() {
   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；
}
```

栈中的数据一但不需要被使用,就会被内存删除.使用有，不使用没有,效率高
堆中的数据一旦被构建,就不能随便删除.效率比较低，它的删除依赖垃圾回收机制
jvm判断删除,用户没办法控制.

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) {
                    main是整个程序的路口
        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；
            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；
        }
    }
```

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；

2). 实例变量

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 = n1 + n2;
        return result;
    }
}
```

2) 实例变量有默认值，使用之前可无须初始化；

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 = n1 + 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Test类，Java虚拟机运行结束。

jvm是否关闭也决定不了堆中的数据的消失，只有等到gc(垃圾回收器)处理完毕才结束生命周期

这里我们主要关心栈，堆和常量池

对于基础类型的变量和常量，变量和引用存储在栈中，常量存储在常量池中

栈中的数据大小和生命周期是可以确定的，当没有引用指向数据时，这个数据就会消失,具有很大的灵活性

堆中的对象的由垃圾回收器负责回收，因此大小和生命周期确定不了,灵活性不高。

对于字符串：其对象的引用都是存储在栈中的，如果是编译期已经创建好(直接用双引号定义的)的就存储在常量池中，如果是运行期（new出来的）才能确定的就存储在堆中。

如以下代码：

Java代码 收藏代码

```
String s1 = "china";
String s2 = "china";
String s3 = "china";
String ss1 = new String("china");
String ss2 = new String("china");
String ss3 = new String("china");
ss1==ss2: false
ss1.equals(ss2):true
```

常量池位于堆区间:主要存放基本数据类型的变量值,String类型的变量值

final和static修饰变量值,所有对象在常量池构建时

先不构建,先查看当前池是否有对应值.如果有直接指向

如果没有先创建后指向

对于基础类型的变量和常量：变量和引用存储在栈中，常量存储在常量池中，

创建的对象位于堆区间