==========Java==========

/**基本数据类型
//当创建变量的时候，需要在内存中申请空间
//内存管理系统根据变量的类型为变量分配存储空间，分配的空间只能用来储存该类型数据

内置数据类型
六种数字类型（四个整数型，两个浮点型），一种字符类型，还有一种布尔型

byte：
byte 数据类型是8位、有符号的，以二进制补码表示的整数；
最小值是 -128（-2^7）
最大值是 127（2^7-1）
默认值是 0
byte 类型用在大型数组中节约空间，主要代替整数
byte 变量占用的空间只有 int 类型的四分之一

short：
short 数据类型是 16 位、有符号的以二进制补码表示的整数
最小值是 -32768（-2^15）
最大值是 32767（2^15 - 1）
默认值是 0
Short 数据类型也可以像 byte 那样节省空间
一个short变量是int型变量所占空间的二分之一

int：
int 数据类型是32位、有符号的以二进制补码表示的整数
最小值是 -2,147,483,648（-2^31）
最大值是 2,147,483,647（2^31 - 1）
默认值是 0
一般地整型变量默认为 int 类型

long：
long 数据类型是 64 位、有符号的以二进制补码表示的整数
最小值是 -9,223,372,036,854,775,808（-2^63）
最大值是 9,223,372,036,854,775,807（2^63 -1）
默认值是 0L
这种类型主要使用在需要比较大整数的系统上
例子： long a = 100000L，Long b = -200000L
"L"理论上不分大小写，但是若写成"l"容易与数字"1"混淆，不容易分辩，所以最好大写

float：
float 数据类型是单精度、32位、符合IEEE 754标准的浮点数
float 在储存大型浮点数组的时候可节省内存空间
默认值是 0.0f
浮点数不能用来表示精确的值，如货币
例子：float f1 = 234.5f

double：
double 数据类型是双精度、64 位、符合IEEE 754标准的浮点数
浮点数的默认类型为double类型
double 类型同样不能表示精确的值，如货币
默认值是 0.0d
例子：double d1 = 123.4

boolean：
boolean数据类型表示一位的信息
只有两个取值： true 和 false
这种类型只作为一种标志来记录 true/false 情况
默认值是 false

例子：boolean one = true

char ：
char 类型是一个单一的 16 位 Unicode 字符
最小值是 \u0000（即为0）
最大值是 \uffff（即为 65 ,535）
char 数据类型可以储存任何字符；
例子：char letter = 'A'

引用数据类型
在Java中，引用类型的变量非常类似于C/C++的指针
引用类型指向一个对象，指向对象的变量是引用变量
这些变量在声明时被指定为一个特定的类型，比如 Employee、Puppy 等
变量一旦声明后，类型就不能被改变了

对象、数组都是引用数据类型
所有引用类型的默认值都是null
一个引用变量可以用来引用任何与之兼容的类型
例子：Site site = new Site("Runoob")

Java常量
final double PI = 3.1415927
为了便于识别，通常使用大写字母表示常量

转义字符

\n	换行 (0x0a)
\r	回车 (0x0d)
\f	换页符(0x0c)
\b	退格 (0x08)
\0	空字符 (0x0)
\s	空格 (0x20)
\t	制表符
\"	双引号
\'	单引号
\\	反斜杠
\ddd	八进制字符 (ddd)
\uxxxx	16进制Unicode字符 (xxxx)

自动类型转换
整型、实型（常量）、字符型数据可以混合运算
运算中，不同类型的数据先转化为同一类型，然后进行运算
转换从低级到高级
低 ----------> 高
byte,short,char—> int —> long—> float —> double

数据类型转换必须满足如下规则：
1. 不能对boolean类型进行类型转换
2. 不能把对象类型转换成不相关类的对象
3. 在把容量大的类型转换为容量小的类型时必须使用强制类型转换
4. 转换过程中可能导致溢出或损失精度，例如：
int i =128;
byte b = (byte)i;
因为 byte 类型是 8 位，最大值为127，所以当 int 强制转换为 byte 类型时值 128 时候就会导致溢出
5. 浮点数到整数的转换是通过舍弃小数得到，而不是四舍五入，例如：
(int)23.7 == 23;
(int)-45.89f == -45

必须满足转换前的数据类型的位数要低于转换后的数据类型
例如: short数据类型的位数为16位，就可以自动转换位数为32的int类型
同样float数据类型的位数为32，可以自动转换为64位的double类型

强制类型转换
1. 条件是转换的数据类型必须是兼容的
2. 格式：(type)value type是要强制类型转换后的数据类型

隐含强制类型转换
1. 整数的默认类型是 int
2. 浮点型不存在这种情况，因为在定义 float 类型时必须在数字后面跟上 F 或者 f

*/

```
// 对象、类、方法、实例变量（相当于python中的属性）

    public class HelloWorld {
      /* 第一个Java程序
       * 它将打印字符串 Hello World
       */
      public static void main(String[] args) {
        System.out.println("Hello World"); // 打印 Hello World
      }
    }

// 大小写敏感：Java 是大小写敏感的，这就意味着标识符 Hello 与 hello 是不同的
// 类名：对于所有的类来说，类名的首字母应该大写
          //如果类名由若干单词组成，使用驼峰式命名，例如 MyFirstJavaClass
// 方法名：所有的方法名都应该以小写字母开头
          //如果方法名含有若干单词，则后面使用驼峰式命名
// 源文件名：源文件名必须和类名相同
    //当保存文件的时候，你应该使用类名作为文件名保存（切记 Java 是大小写敏感的）
    //文件名的后缀为 .java
    //如果文件名和类名不相同则会导致编译错误
// 主方法入口：所有的 Java 程序由 public static void main(String[] args) 方法开始执行


//类名、变量名以及方法名都被称为标识符

/**
关于 Java 标识符，有以下几点需要注意：
  1.所有的标识符都应该以字母（A-Z 或者 a-z）,美元符（$）、或者下划线（_）开始
  2.首字符之后可以是字母（A-Z 或者 a-z）,美元符（$）、下划线（_）或数字的任何字符组合
  3.关键字不能用作标识符
  4.标识符是大小写敏感的*/

/**
Java修饰符
  像其他语言一样，Java可以使用修饰符来修饰类中方法和属性
    主要有两类修饰符：
      访问控制修饰符 : default, public , protected, private
        default (即默认，什么也不写）
          在同一包内可见，不使用任何修饰符
          使用对象：类、接口、变量、方
        private
          在同一类内可见
          使用对象：变量、方法
          注意：不能修饰类（内部类除外）
```

声明为私有访问类型的变量只能通过类中公共的 getter 方法被外部类访问
private 访问修饰符的使用主要用来隐藏类的实现细节和保护类的数据

```
public class Logger {
  private String format;
  public String getFormat() {
    return this.format;
  }
  public void setFormat(String format) {
    this.format = format;
  }
}
```

Logger 类中的 format 变量为私有变量，所以其他类不能直接得到和设置该变量的值
为了使其他类能够操作该变量，定义了两个 public 方法
getFormat() （返回 format的值）和 setFormat(String)（设置 format 的值）

public
对所有类可见
使用对象：类、接口、变量、方法
Java 程序的 main() 方法必须设置成公有的，否则，Java 解释器将不能运行该类

protected
对同一包内的类和所有子类可见
使用对象：变量、方法
注意：不能修饰类（内部类除外）
子类与基类在同一包中：
被声明为 protected 的变量、方法和构造器能被同一个包中的任何其他类访问
子类与基类不在同一包中：
那么在子类中，子类实例可以访问其从基类继承而来的 protected 方法
而不能访问基类实例的protected方法
接口及接口的成员变量和成员方法不能声明为 protected

方法继承的规则：
父类中声明为 public 的方法在子类中也必须为 public
父类中声明为 protected 的方法在子类中要么声明为 protected
要么声明为 public，不能声明为 private
父类中声明为 private 的方法，不能够被继承

非访问控制修饰符 : final, abstract, static, synchronized
static：
静态变量：
无论一个类实例化多少对象，它的静态变量只有一份拷贝
静态变量也被称为类变量 局部变量不能被声明为 static 变量
静态方法：
静态方法不能使用类的非静态变量
静态方法从参数列表得到数据，然后计算这些数据

final ：
final 变量
变量一旦赋值后，不能被重新赋值 被 final 修饰的实例变量必须显式指定初始值
final 修饰符通常和 static 修饰符一起使用来创建类常量
final 方法
父类中的 final 方法可以被子类继承，但是不能被子类重写
声明 final 方法的主要目的是防止该方法的内容被修改
final 类
inal 类不能被继承，没有类能够继承 final 类的任何特性

abstract 修饰符
抽象类：
抽象类不能用来实例化对象，声明抽象类的唯一目的是为了将来对该类进行扩充

一个类不能同时被 abstract 和 final 修饰
如果一个类包含抽象方法，那么该类一定要声明为抽象类，否则将出现编译错误
抽象类可以包含抽象方法和非抽象方法

抽象方法
抽象方法是一种没有任何实现的方法，该方法的的具体实现由子类提供
抽象方法不能被声明成 final 和 static
任何继承抽象类的子类必须实现父类的所有抽象方法，除非该子类也是抽象类
如果一个类包含若干个抽象方法，那么该类必须声明为抽象类
抽象类可以不包含抽象方法
抽象方法的声明以分号结尾，例如：public abstract sample();
*/
/**
Number & Math 类方法
下面的表中列出的是 Number & Math 类常用的一些方法：

xxxValue()	将 Number 对象转换为xxx数据类型的值并返回
compareTo()	将number对象与参数比较
equals()	判断number对象是否与参数相等
valueOf()	返回一个 Number 对象指定的内置数据类型
toString()	以字符串形式返回值
parseInt()	将字符串解析为int类型
abs()	返回参数的绝对值
ceil()	返回大于等于(>=)给定参数的的最小整数，类型为双精度浮点型
floor()	返回小于等于（<=）给定参数的最大整数
rint()	返回与参数最接近的整数 返回类型为double
round()	表示四舍五入，算法为 Math.floor(x+0.5)，将原来的数字加上 0.5 后再向下取整 Math.round(11.5) 的结果为12，Math.round(-11.5) 的结果为-11
min()	返回两个参数中的最小值
max()	返回两个参数中的最大值
exp()	返回自然数底数e的参数次方
log()	返回参数的自然数底数的对数值
pow()	返回第一个参数的第二个参数次方
sqrt()	求参数的算术平方根
sin()	求指定double类型参数的正弦值
cos()	求指定double类型参数的余弦值
tan()	求指定double类型参数的正切值
asin()	求指定double类型参数的反正弦值
acos()	求指定double类型参数的反余弦值
atan()	求指定double类型参数的反正切值
atan2()	将笛卡尔坐标转换为极坐标，并返回极坐标的角度值
toDegrees()	将参数转化为角度
toRadians()	将角度转换为弧度
random()	返回一个随机数

*/
/**Character 方法：

isLetter()	是否是一个字母
isDigit()	是否是一个数字字符
isWhitespace()	是否是一个空白字符
isUpperCase()	是否是大写字母
isLowerCase()	是否是小写字母
toUpperCase()	指定字母的大写形式
toLowerCase()	指定字母的小写形式
toString()	返回字符的字符串形式，字符串的长度仅为1

*/
/**
创建格式化字符串
String 类使用静态方法 format() 返回一个String 对象而不是 PrintStream 对象
String 类的静态方法 format() 能用来创建可复用的格式化字符串，而不仅仅是用于一次打印输出

```
   System.out.printf("浮点型变量的值为 " +
                     "%f, 整型变量的值为 " +
                     " %d, 字符串变量的值为 " +
                     "is %s", floatVar, intVar, stringVar);

   也可以这样：
   String fs;
   fs = String.format("浮点型变量的值为 " +
                      "%f, 整型变量的值为 " +
                      " %d, 字符串变量的值为 " +
                      " %s", floatVar, intVar, stringVar);
*/
/**String 方法：
charAt(int index)          返回指定索引处的 char 值
compareTo(Object o)        把这个字符串和另一个对象比较
                                返回值是整型，它是先比较对应字符的大小(ASCII码顺序)
                                如果第一个字符和参数的第一个字符不等，结束比较
                                返回他们之间的差值，返回值为int
compareToIgnoreCase(String str)        按字典顺序比较两个字符串，不考虑大小写

String concat(String str)          将指定字符串连接到此字符串的结尾
contentEquals(StringBuffer sb)  当字符串与指定的StringBuffer有相同顺序的字符时返回真，返回布尔值
copyValueOf(char[] data)                    返回指定数组中表示该字符序列的 String
copyValueOf(char[] data, int offset, int count)     返回指定数组中表示该字符序列的 String
endsWith(String suffix)            测试此字符串是否以指定的后缀结束，返回布尔值
equals(Object anObject)            将此字符串与指定的对象比较，返回布尔值
equalsIgnoreCase(String anotherString)    将此 String 与另一个 String 比较，不考虑大小写，返回布尔值
getBytes(String charsetName)       使用指定的字符集将此 String 编码为 byte 序列，不提供默认平台编码
                                并将结果存储到一个新的 byte 数组中
getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)        将字符从此字符串复制到目标字符数组
hashCode()                                  返回此字符串的哈希码

indexOf() 方法没有结果时，返回-1
indexOf(int ch)                  返回指定字符在此字符串中第一次出现处的索引
indexOf(int ch, int fromIndex)   返回在此字符串中第一次出现指定字符处的索引，从指定的索引开始搜索
indexOf(String str)              返回指定子字符串在此字符串中第一次出现处的索引
indexOf(String str, int fromIndex)返回指定子字符串在此字符串中第一次出现的索引，指定的索引开始

intern()                   返回字符串对象的规范化表示形式

lastIndexOf(int ch)                  返回指定字符在此字符串中最后一次出现处的索引
lastIndexOf(int ch, int fromIndex) 返回指定字符在此字符串中最后一次出现处的索引
                                   从指定的索引处开始进行反向搜索
lastIndexOf(String str)              返回指定子字符串在此字符串中最右边出现处的索引
lastIndexOf(String str, int fromIndex)   返回指定子字符串在此字符串中最后一次出现处的索引
                                   从指定的索引开始反向搜索

int length()返回此字符串的长度
matches(String regex)                告知此字符串是否匹配给定的正则表达式
regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len) 两个字符串区域是否相等
regionMatches(int toffset, String other, int ooffset, int len)  测试两个字符串区域是否相等
replace(char oldChar, char newChar)    返回一个新的字符串
                                   它是通过用 newChar 替换此字符串中出现的所有 oldChar 得到的
replaceAll(String regex, String replacement)     使用给定的 replacement 替换
                                                 替换此字符串所有匹配给定的正则表达式的子字符串
replaceFirst(String regex, String replacement)
```

split(String regex)　　根据给定正则表达式的匹配拆分此字符串
split(String regex, int limit)　　根据匹配给定的正则表达式来拆分此字符串，限制分割个数
startsWith(String prefix)　　测试此字符串是否以指定的前缀开始
startsWith(String prefix, int toffset)　　测试此字符串从指定索引开始的子字符串是否以指定前缀开始
subSequence(int beginIndex, int endIndex)　　返回一个新的字符序列，它是此序列的一个子序列
substring(int beginIndex)　　返回一个新字符串，它是此字符串的一个子字符串
substring(int beginIndex, int endIndex)　　返回一个新字符串，它是此字符串的一个子字符串

toCharArray()　　将此字符串转换为一个新的字符数组
toLowerCase()　　用默认语言环境的规则将此 String 中的所有字符都转换为小写
toLowerCase(Locale locale)　　使用给定 Locale 的规则将此 String 中的所有字符都转换为小写
toString()　　返回此对象本身（它已经是一个字符串！）
toUpperCase()　　使用默认语言环境的规则将此 String 中的所有字符都转换为大写
toUpperCase(Locale locale)　　使用给定 Locale 的规则将此 String 中的所有字符都转换为大写
trim()　　返回字符串的副本，忽略前导空白和尾部空白
valueOf(primitive data type x)　　返回给定data type类型x参数的字符串表示形式
contains(CharSequence chars)　　判断是否包含指定的字符系列
isEmpty()　　判断字符串是否为空
*/
/**
StringBuffer 和 StringBuilder 类
当对字符串进行修改的时候，需要使用 StringBuffer 和 StringBuilder 类
和 String 类不同的是，StringBuffer 和 StringBuilder 类的对象能够被多次的修改
并且不产生新的未使用对象

StringBuilder 类在 Java 5 中被提出，它和 StringBuffer 之间的最大不同在于
StringBuilder 的方法不是线程安全的（不能同步访问）

由于 StringBuilder 相较于 StringBuffer 有速度优势，所以多数情况下建议使用 StringBuilder 类
然而在应用程序要求线程安全的情况下，则必须使用 StringBuffer 类

StringBuffer 类支持的主要方法：
public StringBuffer append(String s)　　将指定的字符串追加到此字符序列
public StringBuffer reverse()　　将此字符序列用其反转形式取代
public delete(int start, int end)　　移除此序列的子字符串中的字符
public insert(int offset, int i)　　将 int 参数的字符串表示形式插入此序列中
replace(int start, int end, String str)　　使用给定 String 中的字符替换此序列的子字符串中的字符

capacity()　　返回当前容量
charAt(int index)　　返回此序列中指定索引处的 char 值
ensureCapacity(int minimumCapacity)　　确保容量至少等于指定的最小值
getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)　　将字符从此序列复制到目标字符数组 dst
indexOf(String str)　　返回第一次出现的指定子字符串在该字符串中的索引
indexOf(String str, int fromIndex)　　从指定的索引处开始.....

lastIndexOf(String str)　　返回最右边出现的指定子字符串在此字符串中的索引
lastIndexOf(String str, int fromIndex)　　返回 String 对象中子字符串最后出现的位置
length()　　返回长度（字符数）
setCharAt(int index, char ch)　　将给定索引处的字符设置为 ch
setLength(int newLength)　　设置字符序列的长度
CharSequence subSequence(int start, int end)　　返回一个新的字符序列，该字符序列是此序列的子序列
substring(int start)　　返回一个新的 String，它包含此字符序列当前所包含的字符子序列
substring(int start, int end)　　返回一个新的 String，它包含此序列当前所包含的字符子序列
toString()　　返回此序列中数据的字符串表示形式

*/