1. 程序设计
2. 数据

长整数L结尾 浮点数f结尾 十六进制0x开头 二进制0b开头 八进制0开头

1\_000\_000\_000 方便易读，编译器会忽略\_

特殊值：正无穷，负无穷，NaN(不是一个数字,如0/0,负数的平方根)

If(Double.NaN(x)) sout(“X不是一个数字”) //当x不是一个数字的时候为true

当心注释中的\u，\u后面要跟四个十六进制数否则报错，并且这样会表示Unicode符号，如\u00A0表示换行

尽量不使用char，除非需要处理UTF-16代码单元，最好用String

当/的时候时，只有两个操作数都是整数才会返回整数，否则返回浮点数

整数被0除会抛出异常，浮点被0除会得到无穷大或NaN

strictfp标记，规定方法必须严格进行浮点计算，即始终保持64位计算（有可能会出现浮点溢出）

负数进行取模运算，会得到负数，也就是余数不满足≥0。Java提供了floorMod(被模数，除数)方法返回余数，但除数为负数仍会返回负数结果。

位运算：>>>会用0填充高位，>>会用符号位填充。不存在<<<。当操作数为int，右操作数需要进行模32的运算（long需要进行64模运算），如1<<3＝1<<38

字符串的substring(a,b)返回原字符串a到b索引的子串（不含b），长度为b-a。join(“分隔符”,“元素1”,“元素2”,…)返回以分隔符分隔元素的字符串。toLowerCase(),toUpperCase()返回新串，大改小或小改大

1. 控制流程

在嵌套的代码块内部，不能声明同名变量

1. 对象与类
2. 成员方法与成员数据
   1. get返回一个可变对象的引用时，首先应该克隆再return。

如：return (Date) birthday.clone();

如果不克隆，返回赋值的变量名将指向封装好的数据，破坏了封装

* 1. final修饰的可变类，如StringBuilder，引用不可变，但内容可以修改

1. 方法参数

Java中，参数都是值传递。基本数据类型值传递很容易理解，引用数据类型的值传递，类似进行浅拷贝。

1. 包
2. java.util.\* 只能导入util下的一个包，而不是所有包

不同包下含有同名类，如Date类，有两种解决歧义方法：

1. import java,util.\*;

import java.sql.\*;

import java,sql,Date; //指明使用sql下的Date

1. var date = new java.sql.Date(); //加上完整包名
2. 静态导入

import static java.lang.System.\*;

这样导入就可以直接 out.println(“我❤Java”);

还可以实现这样：sqrt(2,4); //而不用Math.spqrt(2,4);

1. 类的设计建议
2. 一定保证数据私有
3. 一定保证初始化数据，不要系统的默认值
4. 不要在类中使用过多的基本数据类型，将它们封装成一个类
5. 不是所有的成员数据都需要get和set
6. 分解过多职责的类，类不要太臃肿
7. 类名和方法名见名知意。类名应该是名词或是-ing动名词
8. 继承
   1. 覆盖

super和this并不是类似的概念。super是调用超类方法的关键字，而不是对象的引用。例如，不能将super赋值给另一个对象变量。但它们有类似的用法，如构造器调用，

* 1. 多态
     1. “is a”原则判断是否设计为继承关系
     2. 声明类型为超类的对象，不能调用子类的方法

不能将超类的引用赋给子类变量