面向对象程序设计（OOP）：使用对象进行程序设计

类（class）：同一类型的对象

导入Java.lang包

Java的每个类都源于lava.lang.Object类，如果在定义一个类时没有指定继承性则它的父类默认为Object。

对象（object）：某一类的实例，调用对象的方法就是让这个对象完成某个任务。

创建对象：new 类 (引用值)

访问对象的数据和方法：.（对象成员访问运算符）。对象名.数据域/方法

实例（instance）：常常和对象互换，意思差不多

方法：对象的行为，用来创建可重用的代码。带返回值的方法必须有return语句，返回值为零时也称过程，在C语言中方法被称为函数

Main方法和其他方法的唯一区别是它由JVM调用

系统将方法的参数，局部变量存储在内存区域stack中

常量声明：final（关键字）数据类型 常量名（一般为大写字母）=值

静态常量声明：final static 数据类型 常量名

静态变量，方法：被一个类中所有实例共享的变量和方法。static 数据类型 变量名/方法。

将一个值赋给范围更小的类型的变量时应进行类型转换：转换后的变量a=（转换后的类型）被转换的变量b，注意b的值不变。

public static 表示公共的静态方法；  
public 表示公共的方法；  
静态方法不需要实例化，直接通过 类名.方法() 调用；  
公共方法需要实例化，通过new 类名.方法() 调用；

字符数据类型：char letter=’A’（=’\u0041’，ASCII码）

字符串数据类型：String newString =new String(“字符串序列”)

String newString =”字符串序列”也是合法的

字符数组char[] charArray={‘A’,’B’,’C’};

String newString =new String(charArray)也是合法的

string str=”java”+233（结果为java233）

在java中，字符串是一个对象。String类中有11个构造方法以及40多个处理字符串的方法。

File类中只包含获取文件属性的方法以及重命名和删除文件，不包含读写文件的方法。

I/O类（java.io）包含文件读写操作

使用符号时在前面加\

要使一个类成为不可变类，必须满足：

1. 所有数据域都是私有的
2. 没有修改器方法
3. 没有一个访问器方法，它会返回一个指向可变数据域的引用

一个类的实例变量和静态变量被称为类变量或数据域，在方法内部被定义的变量称为局部变量

类的抽象：将类的实现与使用分离。类的创建者提供类的描述，从类外可以访问全部的方法和数据域，以及期望这些成员如何行动的描述。

类的封装：类的使用者不需要知道这些类是如何实现的，实现细节经过封装，对用户隐藏起来。

继承：从已有的类派生出新类。

如果类c1扩展（extend）自另一个类c2，则：

c1称为次类（subclass），子类（child class），扩展类（extended class），派生类（derived class）

c2称为超类（superclass），父类（parent class），基类（base class）。

子类从它的父类中继承可访问的数据域和方法，也可以添加新数据域和方法。

Super关键字：用于调用父类的构造方法以及调用父类的方法

接口（interface）

定义：修饰符 interface 接口名{

常量声明（接口中的数据域默认是public static final类型的，可以直接定义：数据类型 变量名 = 值）

方法签名.

}

Implement：让对象的类实现通过接口完成

Cloneable接口：这个接口是空的，不包括常量也不包括方法，一个带空体的接口称为标记接口。它用来表示一个类拥有的某些特定的属性，它的对象可以使用在Object类中定义的clone()方法克隆。

JVM(又称JAVA解释器)

编译时环境：程序源文件.java->java编译器->程序的.class文件

运行时环境：程序的.class文件和JAVA API的class文件->类装载器

JVM的主要任务是装载class文件并执行其中的字节码，装载通过类装载器（class loader）从程序和API中装载，字节码由执行引擎执行

类装载器：

1. 启动(bootstrap)类装载器：JVM实现的一部分，从本地磁盘中装载类
2. 用户定义的类装载器：不是JVM实现的部分，只是JAVA APP可执行代码的一部分。在JAVA程序运行时，使用自定义方式装载类，使JAVA程序可扩展

安全性：JAVA体系结构允许一个JAVA APP中建立多个命名空间，每个类装载器都有它自己的命名空间，它们不能互相访问，除非APP显示地允许。

类装载子系统的动作顺序

1. 装载：查找并把java类型的二进制数据读入JVM中
   1. 通过该类型的完全限定名，产生一个代表该类型的二进制数据流
   2. 解析这个二进制数据流为方法区的内部数据结构
   3. 创建一个表示该类型的java.lang.Class类的实例
2. 连接：将已读入的二进制数据合并到JVM的运行时状态中，分为三个阶段：
   1. 验证：确保被导入类型数据格式的正确性
   2. 准备：为类变量分配内存，并将其初始化为默认值
   3. 解析：把类型中的符号引用转换为直接引用（比如指针），这个步骤可以推迟至初始化之后执行。
3. 初始化：JVM实现必须在每个类或接口首次主动使用时，初始化为正确初始值，有以下情形。
   1. 创建某个类的新实例
   2. 调用某个类的静态方法
   3. 使用某个类或接口的静态字段，或者对该字段赋值时（final修饰的静态字段除外）
   4. 调用Java API中的某些反射方法
   5. 初始化某个类的子类（其超类已被初始化）
   6. 启动含有main()方法的类（启动类）时

超类总是在子类之前被初始化，但是接口并不需要初始化其父接口。

对象的生命周期

类的初始化方法：类和接口的Java class文件中的<clinit>，编译器按声明顺序将类变量初始化语句和静态初始化语句的代码都放到其中，在JVM第一次加载class文件时调用

实例的初始化方法：<init>,在实例创建出来的时候调用

1. 类实例化：四种显式途径，还有一些隐含方法…
   1. 使用new操作符
   2. 调用Class或java.lnag.reflect.Constructor对象的newInstance()方法
   3. 调用现有对象的clone()方法
   4. 通过java.io.ObiectInputStream类的getObject()方法反序列化

Java.class文件：对Java程序二进制文件格式的精确定义。一个class文件中只能包含一个类或者接口。

1. 平台无关性。二进制字节码：JAVA虚拟机的“机器语言”
2. 网络移动性。设计紧凑，需要的时候才下载

可变长度的ClassFIle表中的项

1. magic（魔数）：分辨是否是Java class文件，若以0XCAFEBABE则是。
2. minor\_version和major\_version：主次版本号
3. constant\_pool\_count和constant\_pool：常量池，包含了与文件中类和接口相关的常量
4. access\_flags：展示了文件中定义的类或者接口的信息
5. this\_class：常量池索引
6. super\_class：常量池索引，指向该类超类全限定名的CONSTANT\_Class\_info入口。对于Object类，它的值为0；对于接口，它的值为Java.lang.Object
7. interface\_count和interfaces：在文件中由该类实现或者接口所扩展的父接口的数量，interfaces为数组名，包含了对每个由该类实现或者接口所扩展的父接口的常量池索引
8. fields\_count和fields：对在该类或者接口中所声明的字段的描述。fields\_count是类变量和实例变量的字段的数量总和，fields是不同长度的fields\_info表的序列。
9. methods\_count和menthods：对在该类或者接口中所声明的方法的描述
10. attributes\_count和attribute：该文件中类或者接口所定义的属性的基本信息。

常量池：一个可变长度cp\_info表的有序序列。

常量池中的特殊字符串

1. 全限定名：当常量池入口指向类或接口时，它们给出该类或者接口的全限定名
2. 简单名称：在常量池入口，字段名和方法名都以简单名称出现
3. 描述符：字段的描述符给出了字段类型；方法描述符给出了方法的返回值，参数的数量，类型以及顺序

字段：类的成员（private,public,protected），在class文件中，由一个可变长度的field\_info表描述

方法：在class文件中，由一个可变长度的method\_info表描述

属性

方法区：一段内存，存储从class文件中提取的被装载类型的信息以及类（静态）变量

基本类型信息：

1. 类型的全限定名（在JVM中，类所属包的名称+“ . ”+类名，class文件中“ . ”被“/”替换）
2. 直接超类的全限定名（除非是没超类的java.lang.Object）
3. 这个类型是类类型还是接口类型
4. 访问修饰符（public,abstract或final的某个子集）
5. 任何直接超接口的全限定名的有序列表

其他信息

1. 该类型的常量池：该类型所用的常量的有序集合，包括直接常量和其他的符号引用。像数组一样通过索引访问，在动态连接中起着核心作用。
2. 字段信息：
   1. 字段名
   2. 字段类型
   3. 字段修饰符
3. 方法信息
   1. 方法名
   2. 方法的返回类型（或void）
   3. 方法参数的数量和类型（按声明顺序）
   4. 方法的修饰符

……

1. 除了常量以外的所有类（静态）变量：类变量是所有类实例共享的，即使没有任何类实例它也可以被访问。
2. 一个到类ClassLoader的引用
3. 一个到Class类的引用
4. 类型信息的其他数据结构

堆

所有的对象都存在这里，通过引用传递到栈，每个Java程序都有独立的堆空间，同一个Java程序的多个线程共享其堆空间。

在内存中不必连续，具有动态可扩展性

程序不能主动的释放内存空间，辣鸡收集器负责回收不再运行的对象内存

PC：线程私有。

在内存中不必连续。每个运行中的Java程序在线程启动时都会创建一个自己的PC。内容为下一条被执行指令的“地址”，可以是本地指针也可以是偏移量

Java栈：线程私有

在内存中不必连续。每当启动一个新线程，JVM就会为其分配一个Java栈。

功能：以帧为单位保存线程的运行状态

操作：以帧为单位压栈或出栈。

方法完成，释放出栈：

1. return正常返回
2. 抛出异常

栈帧：

1. 局部变量区：包含对应方法的参数和局部变量，编译器按声明顺序将它们放入局部变量数组，通过索引访问
2. 操作数栈：数据存储方式同局部变量区，但它是通过压栈，出栈操作来访问的。JVM的“寄存器”，从这里取操作数，执行运算，存储结果。
3. 帧数据区：支持常量池解析，正常方法返回及异常派发等

Java API：运行库的集合，提供访问主机系统资源的标准方法。Java API的class文件负责调用本地方法，平台相关。API内部设计是平台无关的。

本地方法栈

调用本地方法时，JVM会保持Java栈不变，不再在线程的Java栈中压入新的帧。本地方法栈与具体实现的本地方法接口有关。

执行引擎：字节码->本地机器代码

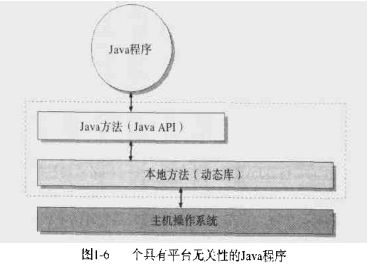
1. 一次性解释字节码
2. 即时编译器(JIT compiler)，快，但消耗内存多（要缓存本地机器代码）
3. 自适应优化器（只把使用最频繁的代码编译为本地代码）
4. 硬件芯片

Java中的两种方法：

1. java方法。由java语言编写，编译成字节码后，存储在.class文件中。平台无关
2. 本地方法。由其他语言编写，编译成与处理器相关的机器代码，存储在动态链接库中。平台相关

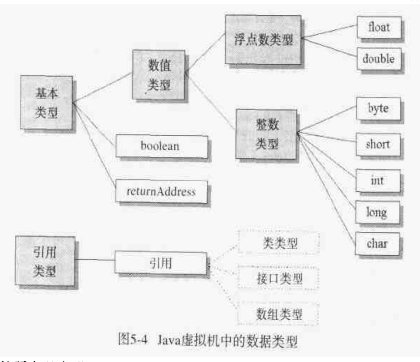
Java程序->本地方法（装载本地方法的动态库）->访问底层操作系统

如果保证程序的平台无关性，只能通过JAVA API来访问底层操作系统



JNI(Java Native Interface)：JAVA本地接口，使本地方法可以在特定主机系统的任何一个JAVA平台实现运行，但是不一定必须支持JNI。

JVM中的数据类型：



卸载：

使用启动类装载器装载的类型永远是可触及的，不会被卸载，只有用户定义的类

装载器的类型才会变成不可触及的，被辣鸡收集。

判断动态装载的类型的Class实例在正常的辣鸡收集过程中是否可以触及：

1. 程序保持对Class实例的明确引用
2. 堆中存在一个可触及的对象，在方法区中它的类型数据指向一个Class实例。

Java中的同步机制：监视器

1. 互斥：对象锁
2. 协作：通过Object类的wait方法和notify方法来实现