访问控制

1) private 私有的

private 关键字是访问控制修饰符，可以应用于类、方法或字段（在类中声明的变量）。 只能在声明 private（内部）类、方法或字段的类中引用这些类、方法或字段。在类的外部或者对于子类而言，它们是不可见的。 所有类成员的默认访问范围都是 package 访问，也就是说，除非存在特定的访问控制修饰符，否则，可以从同一个包中的任何类访问类成员。

2) protected 受保护的

protected 关键字是可以应用于类、方法或字段（在类中声明的变量）的访问控制修饰符。可以在声明 protected 类、方法或字段的类、同一个包中的其他任何类以及任何子类（无论子类是在哪个包中声明的）中引用这些类、方法或字段。所有类成员的默认访问范围都是 package 访问，也就是说，除非存在特定的访问控制修饰符，否则，可以从同一个包中的任何类访问类成员。

3) public 公共的

public 关键字是可以应用于类、方法或字段（在类中声明的变量）的访问控制修饰符。 可能只会在其他任何类或包中引用 public 类、方法或字段。所有类成员的默认访问范围都是 package 访问，也就是说，除非存在特定的访问控制修饰符，否则，可以从同一个包中的任何类访问类成员。

类、方法和变量修饰符

1) abstract 声明抽象

abstract关键字可以修改类或方法。abstract类可以扩展（增加子类），但不能直接实例化。abstract方法不在声明它的类中实现，但必须在某个子类中重写。采用 abstract方法的类本来就是抽象类，并且必须声明为abstract。

2) class 类

class 关键字用来声明新的 Java 类，该类是相关变量和/或方法的集合。类是面向对象的程序设计方法的基本构造单位。类通常代表某种实际实体，如几何形状或人。类是对象的模板。每个对象都是类的一个实例。要使用类，通常使用 new 操作符将类的对象实例化，然后调用类的方法来访问类的功能。

3) extends 继承、扩展

extends 关键字用在 class 或 interface 声明中，用于指示所声明的类或接口是其名称后跟有 extends 关键字的类或接口的子类。子类继承父类的所有 public 和 protected 变量和方法。 子类可以重写父类的任何非 final 方法。一个类只能扩展一个其他类。

4) final 最终、不可改变

final 关键字可以应用于类，以指示不能扩展该类（不能有子类）。final 关键字可以应用于方法，以指示在子类中不能重写此方法。一个类不能同时是 abstract 又是 final。abstract 意味着必须扩展类，final 意味着不能扩展类。一个方法不能同时是 abstract 又是 final。abstract 意味着必须重写方法，final 意味着不能重写方法。

5) implements 实现

implements 关键字在 class 声明中使用，以指示所声明的类提供了在 implements 关键字后面的名称所指定的接口中所声明的所有方法的实现。类必须提供在接口中所声明的所有方法的实现。一个类可以实现多个接口。

6) interface 接口

interface 关键字用来声明新的 Java 接口，接口是方法的集合。

接口是 Java 语言的一项强大功能。任何类都可声明它实现一个或多个接口，这意味着它实现了在这些接口中所定义的所有方法。

实现了接口的任何类都必须提供在该接口中的所有方法的实现。一个类可以实现多个接口。

7) native 本地

native 关键字可以应用于方法，以指示该方法是用 Java 以外的语言实现的。

8) new 新,创建

new 关键字用于创建类的新实例。

new 关键字后面的参数必须是类名，并且类名的后面必须是一组构造方法参数（必须带括号）。

参数集合必须与类的构造方法的签名匹配。

= 左侧的变量的类型必须与要实例化的类或接口具有赋值兼容关系。

9) static 静态

static 关键字可以应用于内部类（在另一个类中定义的类）、方法或字段（类的成员变量）。

通常，static 关键字意味着应用它的实体在声明该实体的类的任何特定实例外部可用。

static（内部）类可以被其他类实例化和引用（即使它是顶级类）。在上面的示例中，另一个类中的代码可以实例化 MyStaticClass 类，方法是用包含它的类名来限定其名称，如 MyClass.MyStaticClass。

static 字段（类的成员变量）在类的所有实例中只存在一次。

可以从类的外部调用 static 方法，而不用首先实例化该类。这样的引用始终包括类名作为方法调用的限定符。

模式：public final static <type> varName = <value>; 通常用于声明可以在类的外部使用的类常量。在引用这样的类常量时需要用类名加以限定。在上面的示例中，另一个类可以用 MyClass.MAX\_OBJECTS 形式来引用 MAX\_OBJECTS 常量。

10)strictfp 严格,精准

strictfp的意思是FP-strict，也就是说精确浮点的意思。在Java虚拟机进行浮点运算时，如果没有指定strictfp关键字时，Java的编译器以及运行环境在对浮点运算的表达式是采取一种近似于我行我素的行为来完成这些操作，以致于得到的结果往往无法令人满意。而一旦使用了strictfp来声明一个类、接口或者方法时，那么所声明的范围内Java的编译器以及运行环境会完全依照浮点规范IEEE-754来执行。因此如果想让浮点运算更加精确，而且不会因为不同的硬件平台所执行的结果不一致的话，那就请用关键字strictfp。

可以将一个类、接口以及方法声明为strictfp，但是不允许对接口中的方法以及构造函数声明strictfp关键字。

11)synchronized 线程、同步

synchronized 关键字可以应用于方法或语句块，并为一次只应由一个线程执行的关键代码段提供保护。

synchronized 关键字可防止代码的关键代码段一次被多个线程执行。

如果应用于静态方法，那么，当该方法一次由一个线程执行时，整个类将被锁定。

如果应用于实例方法，那么，当该方法一次由一个线程访问时，该实例将被锁定。

如果应用于对象或数组，当关联的代码块一次由一个线程执行时，对象或数组将被锁定。

12)transient 短暂

transient 关键字可以应用于类的成员变量，以便指出该成员变量不应在包含它的类实例已序列化时被序列化。

当一个对象被串行化的时候，transient型变量的值不包括在串行化的表示中，然而非transient型的变量是被包括进去的。

Java的serialization提供了一种持久化对象实例的机制。当持久化对象时，可能有一个特殊的对象数据成员，我们不想用serialization机制来保存它。为了在一个特定对象的一个域上关闭serialization，可以在这个域前加上关键字transient。transient是Java语言的关键字，用来表示一个域不是该对象串行化的一部分。当一个对象被串行化的时候，transient型变量的值不包括在串行化的表示中，然而非transient型的变量是被包括进去的。

13)volatile 易失

volatile 关键字用于表示可以被多个线程异步修改的成员变量。

注意：volatile 关键字在许多 Java 虚拟机中都没有实现。 volatile 的目标用途是为了确保所有线程所看到的指定变量的值都是相同的。

Java 语言中的 volatile 变量可以被看作是一种 “程度较轻的 synchronized”；与 synchronized 块相比，volatile 变量所需的编码较少，并且运行时开销也较少，但是它所能实现的功能也仅是 synchronized 的一部分。

程序控制语句

1) break 跳出，中断

break 关键字用于提前退出 for、while 或 do 循环，或者在 switch 语句中用来结束 case 块。

break 总是退出最深层的 while、for、do 或 switch 语句。

2) continue 继续

continue 关键字用来跳转到 for、while 或 do 循环的下一个迭代。

continue 总是跳到最深层 while、for 或 do 语句的下一个迭代。

3) return 返回

return 关键字会导致方法返回到调用它的方法，从而传递与返回方法的返回类型匹配的值。

如果方法具有非 void 的返回类型，return 语句必须具有相同或兼容类型的参数。

返回值两侧的括号是可选的。

4) do 运行

do 关键字用于指定一个在每次迭代结束时检查其条件的循环。

do 循环体至少执行一次。

条件表达式后面必须有分号。

5) while 循环

while 关键字用于指定一个只要条件为真就会重复的循环。

6) if 如果

if 关键字指示有条件地执行代码块。条件的计算结果必须是布尔值。

if 语句可以有可选的 else 子句，该子句包含条件为 false 时将执行的代码。

包含 boolean 操作数的表达式只能包含 boolean 操作数。

7)else 否则

else 关键字总是在 if-else 语句中与 if 关键字结合使用。else 子句是可选的，如果 if 条件为 false，则执行该子句。

8) for 循环

for 关键字用于指定一个在每次迭代结束前检查其条件的循环。

for 语句的形式为 for(initialize; condition; increment)

控件流进入 for 语句时，将执行一次 initialize 语句。

每次执行循环体之前将计算 condition 的结果。如果 condition 为 true，则执行循环体。

每次执行循环体之后，在计算下一个迭代的 condition 之前，将执行 increment 语句。

9) instanceof 实例

instanceof 关键字用来确定对象所属的类。

10)switch 观察

switch 语句用于基于某个表达式选择执行多个代码块中的某一个。

switch 条件的计算结果必须等于 byte、char、short 或 int。

case 块没有隐式结束点。break 语句通常在每个 case 块末尾使用，用于退出 switch 语句。

如果没有 break 语句，执行流将进入所有后面的 case 和/或 default 块。

11)case 返回观察里的结果

case 用来标记 switch 语句中的每个分支。

case 块没有隐式结束点。break 语句通常在每个 case 块末尾使用，用于退出 switch 语句。如果没有 break 语句，执行流将进入所有后面的 case 和/或 default 块。

12)default 默认

default 关键字用来标记 switch 语句中的默认分支。

default 块没有隐式结束点。break 语句通常在每个 case 或 default 块的末尾使用，以便在完成块时退出 switch 语句。

如果没有 default 语句，其参数与任何 case 块都不匹配的 switch 语句将不执行任何操作。

错误处理

1) try 捕获异常

try 关键字用于包含可能引发异常的语句块。

每个 try 块都必须至少有一个 catch 或 finally 子句。

如果某个特定异常类未被任何 catch 子句处理，该异常将沿着调用栈递归地传播到下一个封闭 try 块。如果任何封闭 try 块都未捕获到异常，Java 解释器将退出，并显示错误消息和堆栈跟踪信息。

2) catch 处理异常

catch 关键字用来在 try-catch 或 try-catch-finally 语句中定义异常处理块。

开始和结束标记 { 和 } 是 catch 子句语法的一部分，即使该子句只包含一个语句，也不能省略这两个标记。

每个 try 块都必须至少有一个 catch 或 finally 子句。

如果某个特定异常类未被任何 catch 子句处理，该异常将沿着调用栈递归地传播到下一个封闭 try 块。如果任何封闭 try 块都未捕获到异常，Java 解释器将退出，并显示错误消息和堆栈跟踪信息。

3) throw 抛出一个异常对象

throw 关键字用于引发异常。

throw 语句将 java.lang.Throwable 作为参数。Throwable 在调用栈中向上传播，直到被适当的 catch 块捕获。

引发非 RuntimeException 异常的任何方法还必须在方法声明中使用 throws 修饰符来声明它引发的异常。

4) throws 声明一个异常可能被抛出

throws 关键字可以应用于方法，以便指出方法引发了特定类型的异常。

throws 关键字将逗号分隔的 java.lang.Throwables 列表作为参数。

引发非 RuntimeException 异常的任何方法还必须在方法声明中使用 throws 修饰符来声明它引发的异常。

要在 try-catch 块中包含带 throws 子句的方法的调用，必须提供该方法的调用者。

包相关

1) import 引入

import 关键字使一个包中的一个或所有类在当前 Java 源文件中可见。可以不使用完全限定的类名来引用导入的类。

当多个包包含同名的类时，许多 Java 程序员只使用特定的 import 语句（没有“\*”）来避免不确定性。

2) package 包

package 关键字指定在 Java 源文件中声明的类所驻留的 Java 包。

package 语句（如果出现）必须是 Java 源文件中的第一个非注释性文本。

例:java.lang.Object。

如果 Java 源文件不包含 package 语句，在该文件中定义的类将位于“默认包”中。请注意，不能从非默认包中的类引用默认包中的类。

基本类型

1) boolean 布尔型

boolean 是 Java 原始类型。boolean 变量的值可以是 true 或 false。

boolean 变量只能以 true 或 false 作为值。boolean 不能与数字类型相互转换。

包含 boolean 操作数的表达式只能包含 boolean 操作数。

Boolean 类是 boolean 原始类型的包装对象类。

2) byte 字节型

byte 是 Java 原始类型。byte 可存储在 [-128, 127] 范围以内的整数值。

Byte 类是 byte 原始类型的包装对象类。它定义代表此类型的值的范围的 MIN\_VALUE 和 MAX\_VALUE 常量。

Java 中的所有整数值都是 32 位的 int 值，除非值后面有 l 或 L（如 235L），这表示该值应解释为 long。

3) char 字符型

char 是 Java 原始类型。char 变量可以存储一个 Unicode 字符。

可以使用下列 char 常量：\b - 空格, \f - 换页, \n - 换行, \r - 回车, \t - 水平制表符, \' - 单引号, \" - 双引号, \\ - 反斜杠, \xxx - 采用 xxx 编码的 Latin-1 字符。\x 和 \xx 均为合法形式，但可能引起混淆。 \uxxxx - 采用十六进制编码 xxxx 的 Unicode 字符。Character 类包含一些可用来处理 char 变量的 static 方法，这些方法包括 isDigit()、isLetter()、isWhitespace() 和 toUpperCase()。char 值没有符号。

4) double 双精度

double 是 Java 原始类型。double 变量可以存储双精度浮点值。

由于浮点数据类型是实际数值的近似值，因此，一般不要对浮点数值进行是否相等的比较。

Java 浮点数值可代表无穷大和 NaN（非数值）。Double 包装对象类用来定义常量 MIN\_VALUE、MAX\_VALUE、NEGATIVE\_INFINITY、POSITIVE\_INFINITY 和 NaN。

5) float 浮点

float 是 Java 原始类型。float 变量可以存储单精度浮点值。

使用此关键字时应遵循下列规则：

Java 中的浮点文字始终默认为双精度。要指定单精度文字值，应在数值后加上 f 或 F，如 0.01f。

由于浮点数据类型是实际数值的近似值，因此，一般不要对浮点数值进行是否相等的比较。

Java 浮点数值可代表无穷大和 NaN（非数值）。Float 包装对象类用来定义常量 MIN\_VALUE、MAX\_VALUE、NEGATIVE\_INFINITY、POSITIVE\_INFINITY 和 NaN。

6) int 整型

int 是 Java 原始类型。int 变量可以存储 32 位的整数值。

Integer 类是 int 原始类型的包装对象类。它定义代表此类型的值的范围的 MIN\_VALUE 和 MAX\_VALUE 常量。

Java 中的所有整数值都是 32 位的 int 值，除非值后面有 l 或 L（如 235L），这表示该值应解释为 long。

7) long 长整型

long 是 Java 原始类型。long 变量可以存储 64 位的带符号整数。

Long 类是 long 原始类型的包装对象类。它定义代表此类型的值的范围的 MIN\_VALUE 和 MAX\_VALUE 常量。

Java 中的所有整数值都是 32 位的 int 值，除非值后面有 l 或 L（如 235L），这表示该值应解释为 long。

8) short 短整型

short 是 Java 原始类型。short 变量可以存储 16 位带符号的整数。

Short 类是 short 原始类型的包装对象类。它定义代表此类型的值的范围的 MIN\_VALUE 和 MAX\_VALUE 常量。

Java 中的所有整数值都是 32 位的 int 值，除非值后面有 l 或 L（如 235L），这表示该值应解释为 long。

9) null 空

null 是 Java 的保留字，表示无值。

将 null 赋给非原始变量相当于释放该变量先前所引用的对象。

不能将 null 赋给原始类型（byte、short、int、long、char、float、double、boolean）变量。

10)true 真

true 关键字表示 boolean 变量的两个合法值中的一个。

11)false 假

false 关键字代表 boolean 变量的两个合法值之一。

变量引用

1) super 父类,超类

super 关键字用于引用使用该关键字的类的超类。

作为独立语句出现的 super 表示调用超类的构造方法。

super.<methodName>() 表示调用超类的方法。只有在如下情况中才需要采用这种用法：要调用在该类中被重写的方法，以便指定应当调用在超类中的该方法。

2) this 本类

this 关键字用于引用当前实例。

当引用可能不明确时，可以使用 this 关键字来引用当前的实例。

3) void 无返回值

void 关键字表示 null 类型。

void 可以用作方法的返回类型，以指示该方法不返回值。

保留字

正确识别java语言的关键字（keyword）和保留字（reserved word）是十分重要的。Java的关键字对java的编译器有特殊的意义，他们用来表示一种数据类型，或者表示程序的结构等。保留字是为java预留的关键字，他们虽然现在没有作为关键字，但在以后的升级版本中有可能作为关键字。

识别java语言的关键字，不要和其他语言如c/c++的关键字混淆。   
const和goto是java的保留字。 所有的关键字都是小写

1) goto 跳转

goto 保留关键字，但无任何作用。结构化程序设计完全不需要 goto 语句即可完成各种流程，而 goto 语句的使用往往会使程序的可读性降低，所以 Java 不允许 goto 跳转。

2) const 静态

const 保留字，是一个类型修饰符，使用const声明的对象不能更新。与final某些类似。

3)native 本地

　Java不是完美的，Java的不足除了体现在运行速度上要比传统的C++慢许多之外，Java无法直接访问到操作系统底层（如系统硬件等)，为此Java使用native方法来扩展Java程序的功能。

　　可以将native方法比作Java程序同Ｃ程序的接口，其实现步骤：

　　１、在Java中声明native()方法，然后编译；

　　２、用javah产生一个.h文件；

　　３、写一个.cpp文件实现native导出方法，其中需要包含第二步产生的.h文件（注意其中又包含了JDK带的jni.h文件）；

　　４、将第三步的.cpp文件编译成动态链接库文件；

　　５、在Java中用System.loadLibrary()方法加载第四步产生的动态链接库文件，这个native()方法就可以在Java中被访问了。