

# Java - P1

## Java 特性

语法与 C 和 C++ 相近，但是去弃了一些特性 at final instance : shall not override

如没有运算符重载，多继承，自动的强制类型转换

特别地，Java 不使用指针，而是引用，并提供自动废料收集

(unlike C/C++, Java does not support pointers)

面向对象 Java 提供类，接口，继承等面向对象的特性

只支持类之间的单继承，但支持接口之间的多继承

通过关键字 implements 支持类与接口之间的实现机制

Java 是纯的面向对象程序设计语言

全面支持动态绑定，相比 C++ 只对虚函数使用动态绑定

(unlike C++, Java supports dynamic binding via methods, not via pointers)

分布式 Java 支持 Internet 应用开发

基本 Java 应用编程接口中有网络应用编程接口 (java.net)

提供用于网络应用编程的类库，如 URL, URLConnection, Socket, ServerSocket

提供用于开发分布式应用的远程方法激活 (remote method invocation, RMI)

## 健壮

Java 具有强类型机制，异常处理，自动废料收集

异常处理机制

安全 Java 提供安全机制用于在网络环境防止恶意代码攻击

类 ClassLoader：对通过网络下载的类的安全防范机制

分配不同的名字空间以防止替换本地的同名类，字节代码检查

类 SecurityManager：允许 Java 应用设置安全哨兵的安全管理机制

## 体系结构独立

.java 文件被编译为体系结构独立的字节码格式，即 .class 文件

可以在 Java 平台的任何系统运行，适用于异构的网络环境和软件的分发

## 可移植

Java 严格规定了数据类型的长度，其编译器用 Java 实现，运行环境用 ANSI C 实现

## 解释型

Java 解释器对 .class 中的字节码进行解释执行，需要的类在联接阶段载入运行环境

## 高性能

相比解释型的高级脚本语言，运行速度随着 JIT (Just-In-Time) 编译器发展而接近于 C++

## 动态

Java 适应于动态变化的环境，类能向多动态地载入运行环境，也可以通过网络载入

类具有运行时刻的表示，支持运行时刻的检查

# Java - P2

多线程 Java 中线程是一种特殊对象，必须由 Thread 类或其子类创建

可以使用型构为 Thread(Runnable) 的构造子类

将一个实现了 Runnable 接口的对象包装成一个线程

或者从 Thread 类派生出子类并重写 run 方法

则使用该子类创建的对象即为线程

注意，Thread 类已经实现了 Runnable 接口

于是任何线程均有其 run 方法，而 run 方法中包含线程所运行的代码

Java 支持多个线程的同时执行

关键字 synchronized：用于多线程间的同步机制

Java 程序可以认为是一系列对象的集合通过调用彼此的方法来协同工作

对象 (object)：类的一个实例，有状态 (attribute / property) 和方法 (method)

类 (class)：描述一类对象的 ~~所有~~ 状态与方法的模板

方法 (method)：类及其实例所能执行或被执行的行为

实例变量 (instance variable)：对象的状态由其独有的实例变量值决定

在 Java 的继承中，类可以由其他类派生

即新创建的类可以继承已存在的类，并拥有该类具有的属性和方法

被继承的类称为超类 (superclass)，

由超类派生的类称为子类 (subclass)

~~interface~~ interface

接口 (interface)，接口可理解为 Java 中对象间相互通信的协议

接口可以只定义派生类需要使用的方法，而方法的具体实现由派生类中

Java 源程序与编译型源程序的区别

在 Java 中，对于已经编写好的代码，如 .java 文件

首先通过 javac 编译命令 编译 .java 中的代码

生成包含字节码 (byte code) 的 .class 文件

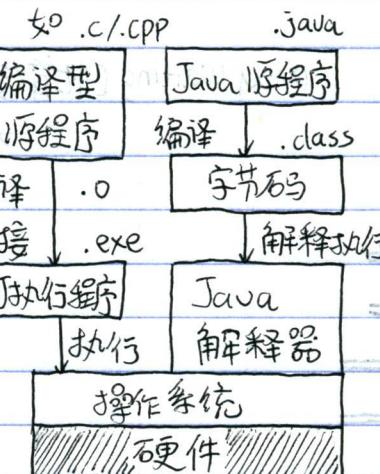
或是包含一系列 class 的 .jar 文件

再通过解释器命令 java 解释执行字节码文件

即不同于如 C++ 的编译型源程序

首先通过编译器与连接器 将代码生成 .exe 可执行文件

再由操作系统直接执行 .exe 文件



# Java - P3

Java - standard

大小写敏感 在Java中，对于大小写是敏感的，所以对于命名有通常的规范

类名：类名的首字母应大写。

如果类名由多个单词组成，则每个单词的首字母分别大写

方法名：方法名的首字母应小写。

如果方法名由多个单词组成，则除第一个单词外的单词首字母大写

源文件名：源文件名(.java)必须与其中定义的类名(class)相同

即当保存文件时，应使用类名作为文件名且后缀为.java

特别地必须保持大小写一致

如果类名与文件名不一致则导致编译错误

主方法入口 在Java中，所有Java程序均由 public static void main(String[] args) 开始执行

标识符 (identifier) 在Java中，所有组成部分(如类，变量，方法)都需要命名

标识符应当以字母(A-Z, a-z), 美元符(\$), 下划线(\_) 开始

标识符可由字母(A-Z, a-z), 美元符(\$), 下划线(\_), 数字(0-9)组合

特别地有，关键字(keyword)不可用作标识符

修饰符 (modifier) 在Java中，可以用修饰符修饰类中的方法和属性

访问控制修饰符：default, public, protected, private

非访问控制修饰符：final, abstract, static, synchronized

关键字 (keyword)

访问控制 private : 私有 protected : 受保护 public : 公共 (接口)

类,方法,变量 abstract : 声明抽象 class : 类 extends : 扩充, 继承 final : 终值, 不可修改 implements : 实现

interface : 接口 new : 新创建 static : 静态 strictfp : 严格, 精准 synchronized : 线程同步

transient : 短暂 volatile : 易失 native : 本地, 原生方法(非Java实现)

程序控制语句 break : 退出循环 case : 定义 switch 选择 continue : 继续下循环 default : 默认 do : 运行

if/else : 如果/否则 for : 循环 instanceof : 实例 return : 返回 switch : 根据选择执行

错误处理 assert : 断言真假 catch : 捕获异常 finally : 无论有无异常 throw : 抛出一个异常对象

try : 捕获异常 throws : 声明可能抛出一个异常 (方法) : 方法签名 (参数)

包 import : 引入 package : 包 (成员) : 成员 (方法) : 方法 (参数)

基本类型 boolean : 布尔值 byte : 字节型 char : 字符型 double : 双精度浮点 float : 单精度浮点

int : 整型 long : 长整型 short : 短整型 (成员) : 成员 (方法) : 方法 (参数)

变量引用 super : 父类超类 this : 本类 void : 无返回值 (成员) : 成员 (方法) : 方法 (参数)

保留 goto : 不可使用 const : 不可使用 null : 空 (成员) : 成员 (方法) : 方法 (参数)

# Java - P4

## 类中的变量

局部变量：在方法、构造方法或者语句块中定义的变量

变量的声明、初始化和使用者都在方法中

方法结束后变量即自动销毁

成员变量：定义在类中，方法体之外的变量，在创建对象的时候实例化  
可以被类中的方法、构造方法和特定类的语句块访问

类变量：与成员变量类似，声明在类中，方法体之外  
区别是必须声明为 static 类型

## 构造方法

等同于 C++ 中的类的构造函数 (constructor)，也可以有多个传入参数不同的构造方法

如果没有显式地定义构造方法，Java 编译器将提供一个默认构造方法

创建类的实例对象时，至少调用一个构造方法

## 创建对象

在 Java 中，使用关键字 new 创建一个新的对象

基本语法结构为 T t = new T(args)

声明：等号左边声明了一个对象，包括对象类型和对象名称

实例化：使用关键字 new 创建一个对象

初始化：new 之后提供了对应对象类型的构造方法

通过传入参数调用构造方法初始化对象

## 源文件声明

在 Java 中，在一个源文件中 定义多个类，以及使用 import 语句和 package 语句时应注意规则

1. 一个源文件中只能有一个 public 类
2. 一个源文件中可以有多个非 public 类
3. 源文件名应与 public 类的类名保持一致
4. 使用 package 语句定义一个类在某个包中
5. 在同一源文件中，不能给不同的类以不同的包声明
6. import 语句和 package 语句对源文件中 定义的所有类均有效
7. 源文件的顺序应为： package 语句 / import 语句 / 类定义

## Java 包

在 Java 中，包主要用于对类和接口进行分类

import 语句 在 Java 中，import 语句提供一个合理的路径，即一个包括包名、类名的完整限定名，则 Java 编译器可以定位到源文件或者类

如 import java.io.\*; 载入 .. /java/io/ 路径下的所有类

# Java - P5

数据类型 内存管理系统根据Java中变量类型为变量分配存储空间

而分配的空间只能用来储存该类型的数据

Java中的数据类型分为内置数据类型和引用数据类型

内置数据类型 在Java中，提供提供6种数字类型(4种整型, 2种浮点型) 1种字符型, 1种布尔型

byte (字节型)：8-bit有符号二进制补码表示的整数

最小值：-128，最大值：127，默认值：0

由于相比其他整数占用空间更小

在大型数组中用于节约空间

short (短整型)：16-bit有符号二进制补码的整数

最小值：-32,768，最大值：32,767，默认值：0

int (整型值)：32-bit有符号二进制补码的整数

最小值：-2,147,483,648，最大值：2,147,483,647，默认值：0

一般整型变量默认认为int类型

long (长整型)：64-bit有符号二进制补码的整数

最小值： $-2^{63}$ ，最大值： $2^{63}-1$ ，默认值：0L

主要用于需要使用大整数的系统

注意：虽然Java是大小写敏感的

但是0L中的L是大小写均可的，但小写更容易和“1”混淆

float (单精度)：单精度32-bit符合IEEE 754标准的浮点数

默认值：0.0f

用于在储存大型浮点数组时节省内存空间

double (双精度)：双精度64-bit符合IEEE 754标准的浮点数

默认值：0.0d

一般浮点型变量默认认为double类型

特别注意：浮点数不能用于表示精确的值，如货币

boolean (布尔值)：1-bit用于记录true/false情况的标志

取值仅有true/false，默认值：false

char (字符型)：16-bit Unicode字符

最小值：'\u0000'，最大值：'\uffff'

通常而言char可以储存任何字符

在Java中，还提供了基本类型void(空)，对应包装类java.lang.Void

但是通常无法直接对void类型进行操作

# Java - P6

第8章 - 引用类型

## 引用类型

在Java中，引用类型的变量与C/C++的指针相似，

指向一个对象，指向对象的变量是引用变量，如数组

变量在声明时被指定为一个特定类型，且声明类型不可改变

引用类型的默认值都是null(空值)

虽然类型不可改变，但是引用变量可以引用任何与之兼容的类型

## 常量

在Java中，常量在程序运行时是不可修改的

关键字final用于定义常量，声明方式与C/C++中const关键字类似，

为便于识别，常量通常用大写字母表示

## 字面量

即Java中的立即数，可以赋给任何内置类型的变量

整型值(byte, short, int, long)可用十进制、八进制或十六进制表示

前缀0表示八进制，如0144，前缀0x表示十六进制，如0xFF

字符型(char)与字符串(String)可以包含任何Unicode字符

前缀\u表示Unicode字符，形如'\uxxxx'，其中xxxx为4-bit十六进制数

## 转义字符

\n	换行符(0x0A)	\r	回车(0x0D)	\f	换页符(0x0C)	\b	退格(0x08)	\0	空字符串(0x20)	\s	字符串
\t		"		'			"	\ddd			
制表符		双引号		单引号			反斜杠		八进制字符		

## 自动类型转换

在Java中，整数型、实数型、字符型数据可以混合运算

先将不同类型数据统一为同一类型，再进行运算

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的类的对象

3. 转换前的数据类型位数应低于转换后的数据类型

byte, short, char ≤ int ≤ long ≤ float ≤ double

4. 在容量大的类型转换为容量小的类型时应用强制类型转换

格式为 (type) value

转换的数据类型必须是兼容的

5. 强制转换过程中可能导致溢出或精度损失

6. 浮点数转换为整型值时，舍弃小数部分

# Java - P7

变量 - 基本概念

## 变量

在 Java 中，变量声明的基本语法结构为：

<type> <identifier> [= <value>] [, <identifier> [= <value>]]\* ;

在 Java 中，支持的变量类型有三种

局部变量：类的方法中的变量

实例变量：独立于方法之外的变量，但没有 static 修饰

类变量：独立于方法之外的变量，使用 static 修饰

## 局部变量

在方法、构造方法或者语句块中声明，在栈上分配

在方法、构造方法或语句块被执行时创建，在执行完成后销毁

访问控制修饰符不可用于局部变量

只在声明的方法、构造方法或者语句块中可见

没有默认值，所以在声明后必须初始化才可以使用

如果使用未初始化的局部变量

抛出错误：variable might not have been initialized

## 实例变量

声明在类中，但在方法、构造方法 和 语句块之外，可以在使用前或使用后

当对象实例化后，实例变量的值随之确定

在对象创建时创建，在对象销毁时随之销毁

外部应该至少能够通过一个方法、构造方法或语句块引用以获取实例变量信息

可以使用访问控制修饰符修饰，通过访问控制修饰符使其对子类可见

对于类的方法、构造方法或语句块可见，通常应设为私有的。

具有默认值，引用变量默认为 null，值可以在声明时指定，或在构造方法中指定

可以通过变量名访问，在静态方法和其他类中应使用完全限定名 Object Reference. Variable

## 类变量

也称为静态变量，在类中以 static 关键字声明，但必须在方法之外

不论一个类创建了多少实例对象，类仅拥有类变量的一份拷贝

除了声明量外很少使用，在初始化后不可改变。

声明为 public/private final static，存储在静态存储区

在第一次被访问时创建，在程序结束时销毁

默认值与实例变量类似，除了声明和构造方法，还可以在静态语句块中初始化

通常声明为 public 以对类的使用者可见，通过 Class Name. Variable 访问

声明为 public static final 类型时，通常使用大写字母

# Java - Pg

修饰符(modifier) 在Java中主要分为访问修饰符和非访问修饰符  
修饰符用于定义类，方法或变量，通常在语句的最前端

访问控制修饰符，用于保护对类，变量，方法和构造方法的访问

default(空)：对于类，接口，变量，方法，在同一包内可见

private：对于变量，方法，在同一类内可见，不能修饰类(外部类)

protected：对于变量，方法，对同一包内的类和子类可见

public：对于类，接口，变量，方法，对所有类可见

访问控制 修饰符	当前类	同一包内	子孙类(同一包)	子孙类(不同包)	其他包
public	T	T	T	T	T
protected	T	T	T	T/F	F
default	T	T	T	F	F
private	T	F	F	F	F

default(默认访问修饰符)，即不使用任何关键字，对同一个包内的类可见

接口里的变量都隐式声明为 public static final

接口里的方法默认情况下访问权限为 public

public(公有访问修饰符)，声明为public的类，方法，构造方法和接口可被任何其他类访问

需要导入public类所在的包，使不同包的public类可相互访问

由于类的继承性，类所有的公有方法和变量都能被其子类继承

Java的main()方法必须设置为公有，否则Java解释器无法运行.class

private(私有访问修饰符)，最严格的访问级别，类和接口不能声明为 private

声明为 private 的方法，变量和构造方法只能被所属类访问

声明为 private 的变量只能通过类中 public 的 getter 方法被外部类访问

主要用于隐藏类的实现细节和保护的数据

protected(受保护的访问修饰符)，不能修饰外部类，接口及接口的成员变量和成员方法

子类与基类在同一包：被声明protected的变量，方法，构造方法可被同一包的任何类访问

子类与基类在不同包：子类的实例可以访问其从基类继承而来的protected方法

不能访问基类实例的protected方法

保护不相关的类使用protected的方法和变量

# Java - Pg

017-10-15

访问控制修饰符的继承规则：

父类中的 public 方法：子类中必须为 public

父类中的 protected 方法：子类中可声明为 protected / public，但不能为 private

父类中的 private 方法：不能继承

protected 可见性：基类的 protected 成员是包内可见的，且对子类可见

如果子类与基类在不同包中，则对于子类实例

可以访问从基类继承而来的 protected 方法

而不可以访问基类实例的 protected 方法

首先确定 protected 成员来自何方及其可见范围

再判断当前访问用法是否可行

```
package P1;
public class Father1 {
    protected void f() { } } ] 父类 Father1 中的 protected 方法 f()

package P1;
public class Son1 extends Father1 { } } package P1
public class Son11 extends Father1 { } } 中间类

package P1;
public class Test1 {
    public static void main(String[] args) {
        Son1 son1 = new Son1();
        Son11 son11 = new Son11();
        son1.f(); } ] f() 从类 Father1 继承，可见性为包 P1 及其子类 Son1
        son11.f(); } ] Test1() 也在包 P1 中，于是编译通过
        son1.clone(); } ] clone() 可见性为 java.lang 包及其子类
        son11.clone(); } } } ] 于是 Son1, Son11 可见而 Test1 不可见，于是编译失败
```

package P3

class MyObject3 extends Test3 {} } MyObject3 为类 Test3 的子类

package P3; } P3 为类 MyObject3 的包

public class Test3 { }

public static void main(String[] args) { } ] clone() 方法来自于类 Test3.

MyObject3 obj = new MyObject3(); } ] 可见性为包 P3 及其子类 MyObject3

obj.clone(); } } ] 在 P3 的类 Test3 中调用，编译通过

# Java - P10

PF - 2023

protected package P2;  
class MyObject2 {  
 protected Object clone() throws CloneNotSupportedException {  
 return super.clone(); } }  
package P2;  
public class Test2 extends MyObject2 {  
 public static void main(String[] args) {  
 MyObject2 obj = new MyObject2(); ] clone() 来自于类 MyObject2, 包 P2 及其子类  
 obj.clone(); } } 编译失败, Test2 不能访问基类实例的 clone()  
Test2 tobj = new Test2(); ] Test2 的 clone() 由 MyObject2 继承  
tobj.clone(); } } 编译通过, 可以访问其自身的 clone() 方法,

package P4;  
class MyObject4 extends Test4 {  
 protected Object clone() throws CloneNotSupportedException {  
 return super.clone(); } }  
package P4;  
public class Test4 {  
 public static void main(String[] args) {  
 MyObject4 obj = new MyObject4(); ] clone() 来自于类 MyObject4, 且没有子类  
 obj.clone(); } } 编译失败, 可见性只有包 P4, 而 Test4 在包 P4 中

package P5;  
class MyObject5 {  
 protected Object clone() throws CloneNotSupportedException {  
 return super.clone(); } }  
public class Test5 {  
 public static void main(String[] args) {  
 MyObject5 obj = new MyObject5(); ] clone() 来自于类 MyObject5, 且没有子类  
 obj.clone(); } } 编译通过, 可见性为包 P5, Test5 在包 P5 中

# Java - P11

```
protected package P6;
class MyObject6 extends Test6 { }
public class Test6 {
    public static void main(String[] args) {
        MyObject6 obj = new MyObject6();
        obj.clone();
    }
}
```

Test6 可见性为子类  
以及包 P6  
MyObject6 来自于类 MyObject6  
Test6 在包 P6 中，编译通过

```
package P7;
class MyObject7 extends Test7 {
    public static void main(String[] args) {
        Test7 tobj = new Test7();
        tobj.clone();
    }
}
public class Test7 { }
```

Test7 来自于类 Object，可见性为  
和子类 Test7, MyObject7 不在范围内  
编译失败

非访问修饰符

- static : 用于类方法与类变量
- final : 用于类(不能被继承), 方法(不能被继承类重定义), 变量(不可修改)
- abstract : 用于创建抽象类和抽象方法
- synchronized / volatile : 用于线程编程

static 修饰符

- 静态变量 : 声明独立于对象的静态变量，也称为类变量
- 无论类实例化了多少对象，静态变量仅有了一份拷贝
- 注意：局部变量不可声明为 static
- 静态方法 : 声明独立于对象的静态方法
- 不能使用类的非静态变量，从参数列表得到数据并计算
- 可使用 classname.variableName 和 classname.methodName() 的方式访问

final 修饰符

- final 变量 : 实例变量必须显式地指定初值，一旦赋值则不能重新赋值
- 通常和 static 修饰符一起使用创建类常量
- final 方法 : 父类的 final 方法可以被子类继承，但不能被子类修改
- 通常的主要目的是防止方法的内容被修改
- final 类 : 不能被其他类继承
- 即使有类能从 final 类继承任何特性

# Java - P12

abstract 修饰符，在 Java 中用于声明抽象类和抽象方法

抽象类：不能用于实例化对象，声明抽象类的目的为对抽象类进行扩充

一个类不能同时被 abstract 和 final 修饰

抽象类可以包含抽象方法 和非抽象方法

如果类中包含抽象方法，则必须声明为抽象类

否则将抛出编译错误

抽象方法：抽象方法是没有实现的方法，具体实现由子类提供

不能被声明为 final 或 static

任何继承抽象类的子类必须提供父类中所有抽象方法的实现

除非将继承抽象类的子类也声明为抽象类

抽象方法的声明以分号结尾（形式上与 C++ 的函数声明一样）

如，public abstract void method();

transient 修饰符，在 Java 中，对于实现为 serializable 的类中

用 transient 修饰符修饰的成员变量不参与序列化过程 (serialization)

transient 修饰的变量不再是对象持久化的一部分，序列化后变量内容无法访问

只能用于修饰变量，不可用于方法和类

注意： transient 不可用于修饰本地变量

对于用户自定义变量，需要类实现 serializable 接口

静态变量 (static) 不论是否被 transient 修饰，均不能被序列化

特别注意：Java 中对象的序列化通过 Serializable 和 Externalizable 接口实现

Serializable：序列化将会自动进行

Externalizable：没有自动序列化，需要在 writeExternal 方法中指定序列化变量

于是与变量是否被 transient 修饰无关

synchronized 修饰符，在 Java 中用于线程的编程，用于修饰方法

synchronized 修饰的方法同一时间只能被一个线程访问

可以与四种不同的访问修饰符组合使用

volatile 修饰符，在 Java 中用于线程的编程，用于修饰线程访问的共享成员变量

volatile 修饰的成员变量每次被线程访问时，强制从共享内存中重新读取变量值

当发生修改时，强制线程将成员变量的值写回共享内存

在任何时候，不同线程总是获得成员变量的同一个值

# Java - P13

算术运算符 在Java中，提供了与C++类似的算术运算符

加法(+)，减法(-)，乘法(\*)，除法(/)，取余(%)

前缀自加/自减(++a, --a)：先进行自加/自减，再进行表达式运算

后缀自加/自减(a++, a--)：先进行表达式运算，再进行自加/自减

关系运算符 在Java中，提供了与C++类似的关系运算符

LT(<), LE(<=), EQ(==), NE(!=), GE(>=), GT(>)

位运算符 在Java中，提供了与C++类似的位运算符

应用于整型值(int), 短整型(short), 长整型(long), 字符型(char), 字节型(byte)

按位与(&), 按位或(|), 按位非(~), 按位异或(^)

左移(<<), 右移(>>), 右移并补零(>>>)

如  $0x3C \ll 2 \rightarrow 0xFF$ ,  $0x3C \ggg 2 \rightarrow 0x0F$

逻辑运算符 在Java中，提供了与C++类似的逻辑运算符

逻辑与(&&), 逻辑或(||), 逻辑非(!)

特别注意：由于Java不会自动将其他类型转化为boolean类型

所以逻辑运算符的运算数必须是boolean类型

短路逻辑：如果(&&)的左运算数为false，则忽略右运算数，并返回false

如果(||)的左运算数为true，则忽略右运算数，并返回true

赋值运算符 在Java中，提供与C++类似的赋值运算符(=)，并同样可与其他运算符组合使用

与算术运算符：+=, -=, \*=, /=, %=

与位运算符：&=, |=, ^=, <<=, >>=

条件运算符 在Java中，提供与C++类似的条件运算符

基本语法为：<boolean expression> ? <value if true> : <value if false>

特别注意：<value if true>与<value if false>必须类型相同

instanceof运算符 在Java中，用于检查对象实例是否是给定类型(类类型/接口类型)

基本语法为：<object reference variable> instanceof <class/interface type>

如果检查对象是给定类型的一个对象，则返回true

检查对象兼容于给定类型，则仍然返回true

否则返回false

# Java - P14

运算符优先级	运算符	操作符	结合性
高	后缀	( ), [ ], .	左结合
	一元	+, -, +!, ~	右结合
	乘除	*	左结合
	加减	/, %	左结合
	移位	<<, >>	左结合
	关系	<, <=, !=, >, >=	左结合
	相等	==, !=	左结合
	按位与	&	左结合
	按位异或	^K	左结合
	按位或		左结合
	逻辑与	&&	左结合
	逻辑或		左结合
低	条件 赋值	=, +=, *=, /=, %=, <<=, >>=, &=, ^=,  =	右结合
	逗号	,	左结合

循环结构 在Java中，提供3种主要循环结构：while, do...while, for

while 循环基本语法结构为  
`while (boolean expression) { statements }` 只要判断 boolean expression 为 true  
 则执行循环体 statements

`}`, 在执行完一次循环体 statements 后  
 do...while 循环基本语法结构为

`do { statements } while (boolean expression);` 在执行完一次循环体 statements 后  
 判断 boolean expression 是否为 true  
`}` if (boolean expression) { 如果为 true, 则执行下一次循环体 }

for 循环基本语法结构为

`for (initiate; boolean expression; update) { statements }` 判断 boolean expression 是否为 true  
 如果为 true 则执行循环体  
`}` 执行更新步骤 update

初始化 initiate 可以初始化多个循环控制变量  
 三部分均可使用空语句，如果 boolean expression 为空，则视为 true

# Java - P15

循环结构 增强 for 循环的基本语法结构为

```
for (variable : collection) { }  
    statements  
}
```

对于由 collection 生成的迭代器 Iterator  
判断其 hasNext() 方法返回值是否为 true  
并将 next() 的返回值绑定到 variable

variable : 声明的局部变量，其类型必须与 collection 中元素类型一致

collection : 数组 或实现了 Iterable 接口的类对象

break 关键字 : 用于在循环结构或 switch 语句中跳出整个语句块

注意 break 只跳出其所在 的循环结构或 switch 结构

continue 关键字 : 用于在循环结构中立刻跳转到循环的下一次迭代

在 for 循环中, continue 跳转至更新语句的部分

增加 for 循环中为跳转至下一次 hasNext() / next() 的位置

在 while / do ... while 循环中, continue 跳转至下一次布尔表达式判断

条件结构 if...if...else 条件语句的基本语法结构为

```
if (boolean expression) {  
    statements  
}
```

```
    else if (boolean expression) { } * 可以有 0 个或多个 else if 结构  
        statements  
}
```

else { } 至少有一个 else 结构，且必须在所有 else if 结构之后

```
    statements  
}
```

当任意一个 if / else if 语句的 boolean expression 值为 true

则执行对应的 statements

并且跳过其后的所有 else if / else 语句

注意所有 if / else if 语句的 boolean expression 按照顺序执行

即仅在之前的 boolean expression 为 false 时才执行下一个判断

在所有的 boolean expression 判断为 false 后，执行 else 语句

注意如果字符串的末尾是 : 请将 return 和 final 语句放在同一行

# Java - P16

switch 结构，Java 中 switch .. case 结构的基本语法为

```
switch (expression) {  
    [ case value : ] * 可以有任意数量的 case 语句结构  
        statements  
    [ break; ]  
    [ default: ] 可以有一个 default 分支结构  
        statements  
}
```

case 判断中的变量类型可以是 byte / short / int / long / char

Java SE 7 开始支持使用字符串 String

case 标签必须为字面量 / 字符串常量

每个 case 后面须有一个用于比较的值和冒号

switch 语句中的 expression 值必须与 case 标签类型相同

switch 结构顺序地比较 expression 的值与 case 标签的值

当某个 case 标签的值匹配成功时，开始执行其后跟随的 statements

只要某个 case 标签匹配成功，则不再判断其余标签

持续执行直到 switch 结构结束，或者执行到 break 语句

当 break 语句执行时，switch 结构结束，跳出 switch

default 分支结构通常是 switch 结构的最后一个分支

只要 switch 结构到达 default 分支，则视为匹配成功

原则上 default 分支不需要 break 语句

Numbers 类，在 Java 中用于解决使用对象而非内置数据类型的情况

Numbers 为抽象类，有对应于内置数据类型的子类

包装类：Integer, Long, Float, Double, Short, Byte

当内置数据类型被当作对象使用的时候

编译器会将内置类型装箱成为包装类

相对地也可以拆箱为内置类型

Numbers 类属于 java.lang 包

Math 类，在 Java 中提供用于执行基本数学运算的属性和方法

Math 类中的方法定义为 static 形式

即可以 直接通过 Math.funcName() 的方式直接调用