Java应用程序开发入门-配套教材

# Java简介

Java环境变量: JAVA\_HOME、CLASSPATH、PATH

Windows:

**JAVA\_HOME=C:\JDK\_1.5**

**PATH=%JAVA\_HOME%\bin;%PATH%**

**CLASSPATH=.;%JAVA\_HOME%\lib\tools.jar**

Java命令行: javac、java

Class & Object类和对象

Variables & methods变量和方法

<http://java.net/downloads/jfjug/SCJP%20Sun%20Certified%20Programmer%20for%20Java%206-0071591060.pdf>

**Table: 完整的Java关键字列表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| abstract | boolean | break | byte | case | catch |
| char | class | **const** | continue | default | do |
| double | else | extends | final | finally | float |
| for | **goto** | if | implements | import |  |
| int | interface | long | native | new | package |
| private | protected | public | return | short | static |
| strictfp | super | switch |  | this | throw |
| throws | transient | try | void | volatile | while |
| assert | enum | synchronized | | instanceof | |

### 访问修饰符号

private, protected, public

### 类,方法和变量修饰:

abstract: 抽象类,声明不能实例化的类,子类必须实现.

class, 定义类

extends, 继承父类,子类扩展父类

final, 最终修饰,无法重写的方法和变量

implements,实现接口

interface:定义接口

native:指出方法是用与平台相关的代码,通常是C代码

new: 实例化一个对象

static:静态方法,或者静态变量,方法或变量属于一个类,而不是实例

strictfp:用在方法或类的前, 浮点数计算严格遵循FP标准

synchronized:同步关键字,代码线程安全

transient: 防止永久串行化,防止序列化.

volatile:挥发变量,无法保证完全同步, 不可见的方式改变值(中断,多线程,寄存器)

### 流程控制

break:从其所在的代码块中出

case:根据switch的值,来执行某个代码块

continue:停止当前循环代码块,继续下一次循环

default:如果任何一个switch-case句不匹则执个代同块:

do: 与while配合,循环语句

if:判断语句

else:如果if测试为false,则执行else的代码块

for:为条件循环

instanceof:确定同一种类、子类或接口的一个实例

return:返回(可以选择返回一个变量);不执行该语句后的任何代码

switch:指出与case语句比较的变量

while:当某个条件为true时，重复地执行代码块。

### 错误处理

catch:声明用于处理异常的代码块

finally:这个代码块通常跟在的-catch

throw:把异常向上传递到调用该方法的方法

throws:指出该方法将把异常传给调用它的方法

try:所要执行但可能引发异常的代码块

assert:计算条件表达式，以验证程序员的假设

### 包控制

import:该语句把包或类导入到代码

package:指出源文件中的所有类属于哪个包

### 原始类型

boolean:表示true或false的值

byte: 8位整数(有符号);

char:单个Unicode字符06位无符号);

double: 64位浮点数(有符号);

float: 32位浮点数(有符号);

int: 32位整数(有符号);

long: 64位整数(有符号);

short: 16位整数(有符号)。

### 变量关键字

super:引用直接父类的引用变量

this:引用对象当前实例的引用变量

### 无返回类型

void

### 未使用的保留关键字

goto

const

# 声明与访问控制

### Identifiers and Keywords

命名:

第一个字符必须是美元符号($),下划线(\_)或者字母

字符后面可以跟数字或者字符

无法使用关键字作为变量

变量是大小写敏感的.

### Java命名规范

类名,第一个字符需要大写,后面的使用驼峰规则

方法,第一个字符需要小写

**变量,驼峰方式,buttonWidth**

**常量 static final MIN\_HEIGHT**

### JavaBeans 规范get & set

属性不是boolean,前缀必须是get

如果是boolean,前缀可以是get或者是is

set方法必须声明成public void set(Object id){}

get方法必须声明成public Type getProperty(){return new Type();}

### Declare Classes

每个源代码文件里面只能有’一个’public的类

public的类需要和文件名一致

import需要在package和class声明之间,package必须在第一行

不是public的类,与文件名可以不一致

一个文件里面可以有多个非public的类

**类的访问修饰符只有public和default**

**Default Access:** *package*-level access,同一个包可见.

final abstract class Wrong{}

//final类,无法被继承

public final class Right {

public void myMethod() { }

}

//abstract类,可以有抽象方法

public abstract class Right2 {

public void myMethod() { }

public abstract void goHome();

}

### Declare Interfaces

接口里面的所有方法隐式的默认都是 public and abstract.

所有接口里面的变量都是常量 public, static, and final—in other words, interfaces can declare only constants, not instance variables.

接口里面的方法无法声明成静态 static.

接口无法声明成final, strictfp, or native.

接口可以继承一个或者多个接口An interface can *extend* one or more other interfaces.

接口无法实现另外一个接口或者类An interface cannot implement another interface or class.

接口可以用于多态Interface types can be used polymorphically

### Declare Class Members

* public 共有成员,都可见
* protected
* default
* private

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Visibility** | **Public** | **Protected** | ***Default*** | **Private** |
| 在同一个类里面 | Yes | Yes | Yes | Yes |
| 同一个package里面的class类里面 | Yes | Yes | Yes | No |
| 同一个package的子类class | Yes | Yes | Yes | No |
| 不在同一个package的子类class | Yes | Yes, *through* *inheritance* | No | No |
| 不在同一个package的其它class类 | Yes | No | No | No |

可用修饰符号

|  |  |  |
| --- | --- | --- |
| **方法内的变量** | **成员变量 (非方法内的变量)** | **成员方法** |
| final | final public protected private static transient volatile | final public protected private static  abstract synchronized strictfp native |

Final Methods最终方法

Final Arguments最终参数

Abstract Methods抽象方法

Synchronized Methods同步方法

Native Methods本地方法

Strictfp Methods严格浮点方法

Constructor Declarations构造器声明

Declaring Primitives 声明原始类型

Array Declarations数组声明

int[] key;// (推荐方式)

int key [];// (可读性比较差,不推荐)

Final Variables最终变量

Transient Variables瞬间变量

Volatile Variables挥发变量

Static Variables and Methods静态变量和方法

**Table 1-3: Ranges of Numeric Primitives**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type** | **Bits** | **Bytes** | **Minimum Range** | **Maximum Range** |
| byte | 8 | 1 | -27 | 27-1 |
| short | 16 | 2 | -215 | 215-1 |
| int | 32 | 4 | -231 | 231-1 |
| long | 64 | 8 | -263 | 263-1 |
| float | 32 | 4 | n/a | n/a |
| double | 64 | 8 | n/a | n/a |

# Java-面向对象

### 继承&重写方法

--当继承一个类时，就有机会重写该方法(final方法无法被重写,final类也无法被继承)

--子类如果不是抽象类,必须重写父类的抽象方法

--重写的主要优点是能够定义某个子类型特有的行为

--调用被重写方法的父类版本,使用关键字super

\*重写方法的修饰符只能扩大,无法缩小(父类为public的方法,子类不能修饰成protected)

\*重写方法的异常声明只能缩小,无法扩大(父类为RuntimeException,子类无法声明Exception的异常) ,不能抛出新的检查异常

\*参数列表必须完全与被重写的相同

\*返回类型必须完全与被重写的运回类型相同

\*不能重写被标识为final的方法:

### 重载方法

\*重载方法必须改变参数列表

\*重载的方法可以改变返回类型

\*被重载的方法可以改变访问修饰符

\*被重载的方法可以声明新的或更广的检查异常

\*方法能够在同一个类中或者在一个子类中被重载

### 构造函数与函数链

\*每个类(包括抽象类)都必须有一个构造函数

\*如果没有写构造函数,编译器会自动添加一个默认的[无参构造器]

\*构造函数没有任何返回类型(如果存在return,就变成方法了)

\*构造函数名称必须完全与类名相同

\*构造函数能够使用任何访问修饰符，包括private

\*每个构造函数第一条语句:必须调用重载构造函数(this ( ) )或调用父类构造函数(super() )

\*如果构造器第一条语句没有输入super调用,编译器则将插入一个默认的super ()无参数调用

\*接口没有构造函数。接口不是对象继承树的一部分:

\*调用构造函数的唯一方法是从另一个构造函数之内调用。

### 关于默认构造器

\*默认构造函数与类具有相同的访问修饰符

\*默认构造函数没有任何参数

\*默认构造函数包含对父构造函数super() 的无参数调用。

# 赋值 & 操作符

### 赋值与操作符

**Table : 原始类型与引用类型,默认值**

|  |  |
| --- | --- |
| **Variable Type** | **Default Value** |
| Object reference | null (not referencing any object) |
| byte, short, int, long | 0 |
| float, double | 0.0 |
| boolean | false |
| char | '\u0000' |

*赋值操作符 (limited to:* =, +=, -=*),*

*算术运算符 (limited to:* +, -, \*, /, %, ++, --*)*

*关系运算符 (limited to:* <, <=, >, >=, ==, !=*)*

*类型判断instanceof operator:*

*逻辑运算 (limited to: &, |, ^, !, &&, | |)*

*三目运算符 (? :)*

*移位运算符(<<:左移,>>:右移,>>>:0填充右移)*

&&:短路与:||: 短路或。短路运算特点是:不在无意义的计算上面浪费时间

x++,++x的区别

方法:Passing Object Reference Variables

方法Passing Primitive Variables

### 封装器,自动开箱

valueOf()返回封装对象

parseXxx()返回原始类型

**Table 3-2: 封装器类型与原始值类型**

|  |  |  |
| --- | --- | --- |
| **Primitive** | **Wrapper Class** | **Constructor Arguments** |
| boolean | Boolean | boolean or String |
| byte | Byte | byte or String |
| char | Character | char |
| double | Double | double or String |
| float | Float | float, double, or String |
| int | Integer | int or String |
| long | Long | long or String |
| short | Short | short or String |

# 控制语句 & 异常断言

### if 和 switch 语句

if-else Branching,注意:条件里面不要缺失括号

//// Break and Fall-Through in switch Blocks

int x = 3;

switch (x) {

case 2:

System.out.println("x is equal to 2");

break;

case 3:

System.out.println("x is equal to 3");

break;

default:

System.out.println("Still no idea what x is");

}

int x = 2;

switch (x) {

case 2: System.out.println("2");

default: System.out.println("default");

case 3: System.out.println("3");

case 4: System.out.println("4");

}

### 循环和迭代

Using while Loops

int x = 2;

while(x == 2) {

System.out.println(x);

++x;

}

Using do Loops

do {

System.out.println("Inside loop");

} while(false);

basic for Loop

\*for大量循环,要小心创建变量

for (int i = 0; i<10; i++) {

System.out.println("i is " + i);

}

Termination the for statment

break,跳出当前循环

return,返回方法的结果

System.exit(),程序停止执行,JVM虚拟机关闭

continue,当前循环停止,继续下一次循环

### 异常类

try {

//

}catch(CantGetFileFromNetwork) {

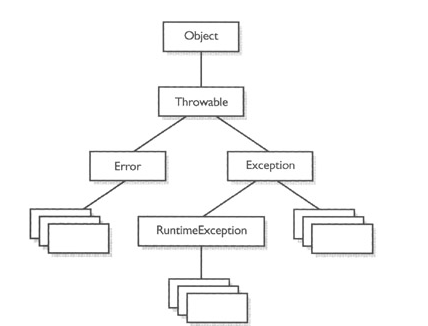
//

}finally{

//

}

如何抛出异常???



JVM Thrown Exceptions: NullPointerException, StackOverflowError

Programmatically Thrown Exceptions

**Table 5-2: 一般常见的异常类.**

|  |  |  |
| --- | --- | --- |
| **Exception (Chapter Location)** | **Description** | **Typically Thrown** |
| **ArrayIndexOutOfBoundsException**  ([Chapter 3](itss://chm/10909/BBL0035.html#397), "Assignments") | Thrown when attempting to access an array with an invalid index value (either negative or beyond the length of the array). | By the JVM |
| **ClassCastException**  ([Chapter 2](itss://chm/10909/BBL0022.html#234), "Object Orientation") | Thrown when attempting to cast a reference variable to a type that fails the IS-A test. | By the JVM |
| **IllegalArgumentException**  ([Chapter 3](itss://chm/10909/BBL0035.html#397), "Assignments") | Thrown when a method receives an argument formatted differently than the method expects. | Programmatically |
| **IllegalStateException**  ([Chapter 6](itss://chm/10909/BBL0058.html#878), "Formatting") | Thrown when the state of the environment doesn't match the operation being attempted, e.g., using a Scanner that's been closed. | Programmatically |
| **NullPointerException**  ([Chapter 3](itss://chm/10909/BBL0035.html#397), "Assignments") | Thrown when attempting to access an object with a reference variable whose current value is null. | By the JVM |
| **NumberFormatException**  ([Chapter 6](itss://chm/10909/BBL0058.html#878), "Formatting") | Thrown when a method that converts a String to a number receives a String that it cannot convert. | Programmatically |
| **AssertionError**  (This chapter) | Thrown when a statement's boolean test returns false. | Programmatically |
| **ExceptionInInitializerError**  ([Chapter 3](itss://chm/10909/BBL0035.html#397), "Assignments") | Thrown when attempting to initialize a static variable or an initialization block. | By the JVM |
| **StackOverflowError**  (This chapter) | Typically thrown when a method recurses too deeply. (Each invocation is added to the stack.) | By the JVM |
| **NoClassDefFoundError**  ([Chapter 10](itss://chm/10909/BBL0088.html#1448), "Development") | Thrown when the JVM can't find a class it needs, because of a command-line error, a classpath issue, or a missing .class file. | By the JVM |

异常在继承的方法中只能缩小,无法扩大

# 字符 & Math

字符对象String不变性,String常量池

String myString = new String(“HelloWorld”);

myString.concat(“more stuff”) ;//not assignment

myString.toLowerCase();//not assignment

System.out.println(“myString = ”+ myString);

**比较字符串,使用equal**

**Object中的方法**

**比较对象使用equal**

**toString()方法 ,可以描述这个对象的信息, 调试的时候使用print(obj),obj自动调用toString方法,并打印这个方法return的字符串**

**什么时候重写hashCode()和equals方法?**

**equal与==操作符的区别,**

**如果不重写equal方法,有可能只是比较了[对象的引用].**

**如果不重写eqaul方法,不能将对象作为Hash散列的key,不能将对象用于检索元素List的search.**

**1,确保测试对象类型是否正确,使用instanceof进行测试.**

**2,比较关键属性的比较.**

**HashCode:**

1,在 Java应用程序执行期间,如果没有修改对象的 equals ()比较内使用的任何信息, 则多次调用 hashCode()方法时,必须一致返回同一个整数。而在应用程序的一次执行和同一个应用程序的另一次执行期间,  这个整数不需保持一致:

2,如果equal方法返回true,则hashCode方法必须返回一致的整数

3,如果equal方法返回false,则hashCode方法可能返回一样的整数(散列算法效率)

**注意:**

**for循环里面不要使用字符串加操作(**每个不被引用的String,都被废弃

**,造成垃圾回收),使用StringBuffer代替,**

public synchronized StringBuffer append(String s)

StringBuilder非线程安全

**思考题:**

**为何StringBuffer效率高?**

Math 类

randomO**返回大于等于0.0 并且小于1.0 的随机double 值。**

**round(),把参数加上0.5 ，并截断为最接近整数。四舍五入**

sqrtO返回平方根

abs ()绝对值

ceil ()进一法

floor ()去尾法

sin()返回角度的正弦弧度, cosO, tanO

Math. toRadians ( **)把角度转换为弧度**

Math.toDegrees(Math.PI • 2.0) // returns 360.0把弧度转换为角度

# 泛型 & 集合

### 泛型 & 集合

Collection接口,表示集合-数据结构,可以被存储,被反序列化.包含List和Set

Collections为实用类,包含一些实用的静态方法,用于处理集合数据.

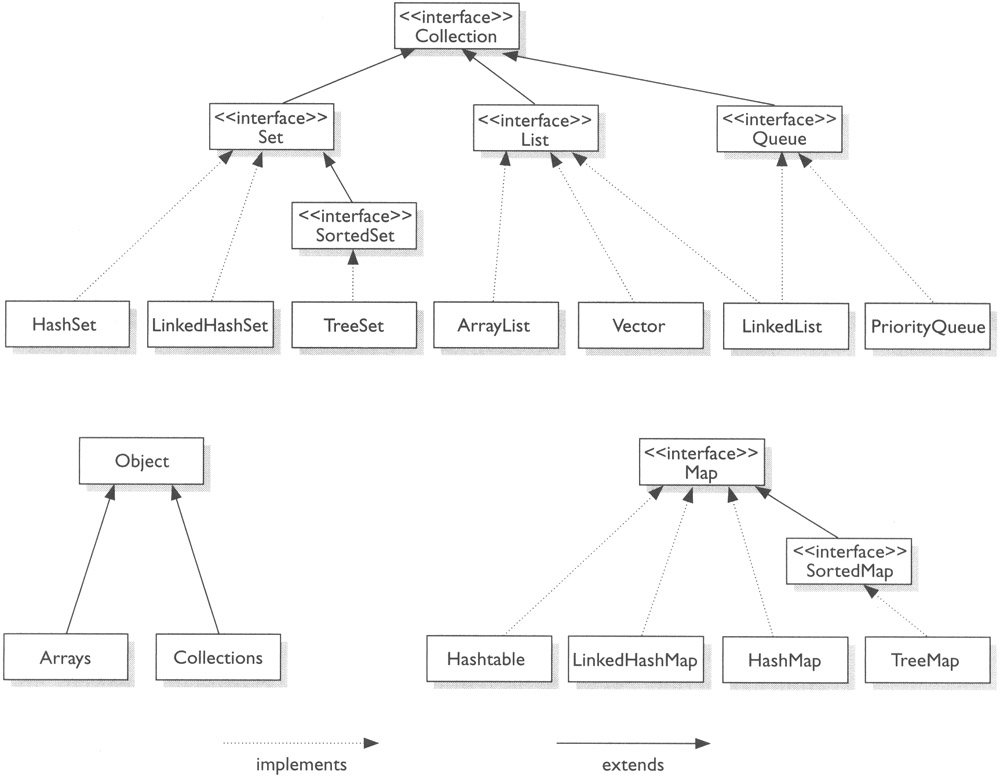
List有序性,

Set唯一性,

Map唯一标志,key:value

Ordered & Sorted,关于有序和自然顺序

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Maps** | **Sets** | **Lists** | **Queues** | **Utilities** |
| HashMap | HashSet | ArrayList | PriorityQueue | Collections |
| Hashtable | LinkedHashSet | Vector |  | Arrays |
| TreeMap | TreeSet | LinkedList |  |  |
| LinkedHashMap |  |  |  |  |



### 垃圾回收:

finalize()方法在垃圾回收之前被执行一次.

记住:不要去重写这个finalize()方法,并把实质的代码放进去.因为垃圾回收的执行时间,并不是确定的.

System.gc()

# 内类(嵌套类)

普通内部类:内部类可以访问外部类的成员变量;实例化内部类;访问外部类的this指针

方法内部类: 方法内部类只能在定义该法内实例化。其它地方都无法实例化

匿名内部类:1,重载类方法的匿名内部类;2实现接口的匿名内部类;3参数定义的内部类

静态内部类:没有外部类实例,也可以访问静态内部类(相当于静态成员)

# 线程

线程start(),yield(),sleep(),run(),wait()

public class TestThreads {

public static void main (String [] args){

Thread t = new Thread(new Runnable(){

public void run(){

System.out.println(“Thread running”);

}

});

t.start();

}

}

1,线程运行顺序没有保证

2,线程状态:新Thread->可运行(调用start)->运行->结束

----等待/堵塞/睡眠

3,线程同步

只能同步方法,而不能同步变量:

每个对象只有一个锁:

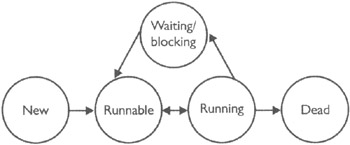
不必同步一个类中的所有方法,类可以有同步和非同步方法:

如果同步一个类中的两个同步方法,则只有<一个线程>能够访问这两个同步方法。

可以同步代码块

线程进入睡眠,将带锁睡眠

静态方法也可以同步



# 开发 & 调试

开发 & 调试