Java 学习笔记及其与 C++的比较

目 录

0. 问题

- Java 中有模板吗？Java 中如何实现类似模板的功能？
- Java 没有多重继承，如何实现多重继承的功能？
- Java 中各种主要数据结构的实现与 C++有什么区别？
- Java 的设计思想与 C++有什么重要区别？
- Java 相对 C++有什么优势与弱势？
- 在 Java 中抽象类和接口类有区别吗?
- 在各个重要特性上 Java 与 C++有什么重要区别?
- 什么时候该选择 Java 作为开发语言,什么时候该选择 C++?
- Java 中 static 定义与 C++中有区别吗?
- 什么是 Java 类的 Persistence?
- 为什么 Java 程序是可移植的?
- Java 中如何控制名字空间,与 C++有什么不同?
- Java 中定义 literal string，什么时候用＝＝比较时返回 true？什么时候返回 false?
- Java 舍弃了 C++中的什么?增加了什么?
- Java 中的引用与 C++的引用有什么区别吗?
- Java 的 finalize 方法起什么作用?
- AWT 与 Swing 有什么不同? Swing 与 JavaBeans 有什么联系？
- 在 Java 中 ASCII 码与 Unicode 的关系如何?
- Applet 与普通 Windows 应用程序有什么不同？
- 什么是 Javabeans? 为什么需要 Javabeans?
- 怎样实现 business logic 与 UI logic 的分离？如何实现两者之间的信息传递？
- Jar 的用法如何？
- 如何在 Java 中实现 C/C++之 const 常量？Java 中的 final 与 C/C++中的 const 有什么不同？
- Class 与 interface 有什么区别？
- 接口（interface）与抽象类(abstract class)有什么不同？什么时候使用 interface?什么时候使用 abstract class?
- Java class 中默认（即不加 public, private 或 protected 控制关键字）的控制属性是什么？
- Linux 中的 gcj 和 gij 命令如何使用？与普通的 javac 与 java 命令有什么区别？
- Java 中有无符号整型吗？
- Java 中的 byte 和 char 类型有什么区别？
- Math 类中有哪些方法可用？
- 什么是 numeric promotion? 有什么用？
- 哪些类型转换是不要显示申明的？哪些需要？哪些转换可能是危险的？哪些没有危

险？

- java.io.Serializable 和 Cloneable 类有什么用？
- 什么是 FP-strict 表达式？什么是 FP-strict 类？
- Object 类在 Java 中有哪些重要作用？
- Interface 类在什么时候需要使用？如何使用
- 已经有了 package1 Class1，可以定义 package package1, class Class2 来使 Class2 对 Class1 有包访问控制吗？
- Java 的类与 C++的类有什么不同？使用方法有什么不同？
- Java 中的 protected 关键字什么时候非常有用？Java 的 protected 与 C++的 protected 有区别吗？
- Java 中如何实现多态？用基态的引用如何知道派生类的类型？如何使用派生类的方法和变量
- 什么是设计模式(Design Patterns)? 设计模式有什么用？
- 如何实现对象之间的信息传递？如何完成对象之间的通信？
- 如何设置类中各部分的权限？哪些应设为 private?哪些应设为 public?哪些应设为 static?哪些应设为 final?
- Java 中有内联函数吗？如何保证存取函数的效率？
- Java 中有运算符重载吗?如何实现运算符的重载？
- 如何设计 package?
- Abstract interface 和 non-abstract interface 有什么区别？
- 哪些方法应定义为 abstract?哪些不应定义为 abstract?
- Java 方法内参数传递是按值还是按引用传递？f(X x)相当于 C++的 f(const X& x), f(X& x) 还是 f(X x)？如何实现按引用传递？如何实现按值传递？
- Java 中允许按返回值重载吗？
- Java Exception 机制的特点？与 C++的区别？Java 中 Exception 的分类？
- 怎样区别使用 throws 和 throw 关键字？
- Java 如何运行一个程序？如何将 Java 的.class 文件变成与操作系统相关的可执行文件？可以不用装 JRE 就可以使用吗？
- 为什么说 Java 是一种多线程的语言？多线程处理是如何内置于 Java 语言中的？
- 关键字 instanceof 的作用是什么？
- Java 中如何得到一个引用具体指向的类类型？
- Java 是纯粹面向对象的语言吗？为什么？
- Java 中有拷贝构造函数吗？如何实现拷贝构造函数的功能？为什么在 Java 中类似 C++构建拷贝构造函数会产生问题？
- 如何防止对象被拷贝

- 对于 Java 编程有什么常见建议？
- Java 程序的一般命名规则如何？
- Java 中有哪些集合类？各适用于哪些应用？Java 中有哪些包容器类？
- 什么是 hash code，在 Java 中有何作用？如何编写高校的 hash 函数？
- System.out 和 System.err 有什么区别？
- 在 Java 中如何播放声音文件？javax.sound.midi 包如何使用？
- Java 中如何创建目录？如何创建文件？如何写文件到指定目录？
- 怎样持久化一个对象以保持一个对象的目前状态？
- 什么是对象的 serialization？serialization 对对象做什么工作？如何实现对象的串行化？
- 什么是 regular expressions？如何使用？它可以完全代替以前的 StringTokenizer 类吗？
- 什么是 thread? thread 与 process 有什么区别与联系？不同 thread 之间如何进行通信？什么时候需要多线程编程？
- threadName.join()实现什么功能？
- 已经继承了某类的类不能在继承 Thread 类，如何实现多线程功能?
- JPanel，JFrame，JApplet 有什么不同？
- 什么是 Ant?
- Java 的 Wrapper classes 有什么用？
- 什么是 Reflection? 有什么用？
- JVM 怎样装载类？
- Java 的 String 类与 C++ STL 中的 string 类使用上有什么区别？

1．Java 特性

1．1 语法和词汇

Java 的语言是在 C++基础上发展起来的一种面向对象的高级语言，它的语法基本是继承了 C++的语言，但有以下几点值得注意：

- C 语言是函数的集合。C++兼容 C, 既有函数，也有类。C++可以实现面向对象编程，但是仍然离不开一个非对象的 main 主函数。这也是初学 C++者常常会感到费解之处。他们虽然理解了 C++的语法，但是不知道怎样实现面向对象编程。事实上，如果将 C++当成更好的 C 使用，其实本来就是面向过程的语言。只有利用面向对象的思想，使用面向对象的特性进行编程时，它才是一种面向对象的语言。Java 是比 C++更纯的面向对象的语言[1]。Java 程序是类的集合，程序的执行进入点是某一个类的 public void static main()方法。正因为 Java 程序是类的集合，Java 类定义后并不需要“;”（C++中类后面忘记加“;”有时是个会害死人的 Bug!）。
- Java 没有 goto 语句，虽然 Java 可以用带标号的 break, continue 语句实现跳转，但是跳转的范围是有限制，它不像 C/C++那样可以随意跳转。
- Java 语言中 boolean 变量是一种单独的特殊的变量，它只有两个值：true, false。并且 boolean 变量不能与整型变量相互转换。逻辑操作符 AND(&&), OR(||)和 NOT(!)在 Java 中只能作用在 boolean 变量上。C 的布尔变量是用整型代替的，ANSI C++虽然发展出 bool 变量，但是可以用整型代替，也可以与整型相互转换。因次 if(a==b)错写成 if(a=b)的错误将被编译器发觉。
- Java 中相对 C++增加了无符号右移运算符(>>>, >>>=)。
- Java 语言使用的字符集是 Unicode 字符集，Java 用\uxxxx 来标识一个 Unicode。标识符名没有长度限制，而且可以使用绝大部分 Unicode 字符。标识符第一个字符是 Character.isJavaIdentifierStart 方法对之返回 true 的字符。Java 语言提供了一种用 Unicode 编写的程序向 ASCII 转换的方式，以便能为基于 ASCII 的工具处理（\uxxxx 变成\uuxxxx, 而非 ASCII 字符转换为\uxxxx）。C++使用 ASCII 字符集，标准 C++标识符可识别的长度为 32。特别的字符包括\b(BS),\t(HT), \n(LF), \f(FF), \r(CR) \”(双引号)，\’(单引号)，\\(反斜杠)。
- String 类型是 Java 语言默认包含包 java.lang 中定义的类型，可以直接使用。Literal strings 何时指向同一对象值得注意：

  （1）literal strings 在同一包里，无论属于同一类，还是不同类，都指向同一 String 对象；

  （2）在不同包不同类的 literal strings 也指向同一对象；

  （3）在编译期间通过常量计算表达式出的 String 仿佛是 literal string(“hel”+”lo”==”hello”)

[1] 但是 Java 也不是纯粹的面向对象的语言，后面会对此做出解释。

(4) 运行时计算出的 strings 是新建的，所以是不同的（String lo=”lo”, “hel”+lo!=”hello”）。

（5）显式用 intern()方法变换的 String 与预先定义的相同。

（6）使用 new String（）方法定义的对象与 literal string 不同（String a=”abcd”, String b=new String(“abcd”), a 与 b 是不同的）。

- Java 的 String 是 java.lang 中定义的，可以说是内置的，而 C++的 string 是 STL 中引入的，注意 C++是用 string 而不是 String。Java 的 String 不可改变，要改变必须使用 StringBuffer。C++中 string 相当于 Java 的 StringBuffer。
- 位操作运算符(bitwise AND(&), bitwise OR(|), bitwise XOR(^))都可以作用在 boolean 量上，但是位操作 NOT 运算符(bitwise NOT(~))不能作用在 boolean 量。对于 boolean 量，位操作运算符的产生的结果是与逻辑运算符一样的，但是它们不 short circuit 表达式（逻辑运算符只要得到结果就不往后计算）。

表 1.1　Java 与 C++的比较 1 （语法和词汇）

项 目	Java	C++
字符集	Unicode	ASCII,其它语言支持是附加的。
标识符	不定长。第一个字符可使用任何 Unicode 字符	规定字符（第一个字符必须是字母或下划线，**变量名中只能使用字母、数字和下划线）**，标准 C++未规定标识符可识别长度（可识别长度随系统而定，但一般不要超过 32 个字符）。
关键字	特殊：boolean, abstract, extends, final, finally, implements, import, Interface, package, native, strictfp, synchronized, throws, transient (true, false, null)[2] （const, goto）[3]	extern, register, sizeof, struct, union, bool, delete, friend, inline, virtual, operator, template, typedef
操作符	同 C++,另加>>>, >>>=	=, >, < , !, ~, ?, : , ==, <=, >= !=, &&, ||, ++, --, +, -, * ,/, &, |, ^, %, <<, >>, +=, -=, *=, /=, &=, |=, ^= , %=, <<=, >>=
逗号表达式	只能在 for 循环中使用	任意地方使用
布尔变量	关键字 boolean, 与整型没有任何联系	开始是用整型代替的，0 为 false, 非 0 即为 true
字符串变量	String （不可变）	string （可变）

[2] true, false, null 与关键字很类似，因为它们不能作为标识符，但是它们实际是常数。

[3] const, goto 虽然作为 Java 语言的保留关键字，但是没有使用。它们作为关键字可以有助于编译器更好地报错。

1.2 类型、值、变量、表达式

- Java 语言是一种强类型的语言，这意味着所有的变量和表达式在编译期间都有一个类型。
- Java 的类型有两种：基本类型（primitive types）和引用类型(reference types)。基本类型的存在说明 Java 语言不是一种纯粹的面向对象的语言（纯粹面向对象的语言一切都是对象，而基本类型不是对象）。以下是 Java 类型的分类图：

  基本类型 <布尔类型（boolean>

  <void>

  <数值类型(numeric)> <整型（integral）> <byte>

  <short>

  <int>

  <long>

  <char>

  引用类型 <类（class）>

  <接口（interface）>

  <数组 (array)>

  <空类型(null)>
- 当任何类型在类变量定义时，如果没有初始化，都会赋予默认值，数值类型默认值为 0，**布尔类型默认值为 false.** 引用类型默认值为 null。Java 中所有数值类型的长度在任何机器、任何平台上都是固定的，因此也没有 C/C++的那种 sizeof 运算符。本条也是 Java 语言只所以具有平台无关性的重要原因。另有一点非常值得注意的是：**对于局域基本类型，Java 不允许其不做初始化就使用。而且不允许重定义一局域变量，以下代码在 Java 中是错的：**

```
{
    int x=12;
    {
        int q=96;
        int x=10;
    }
}
```

- 在 Java 中，boolean 类型除了可以转换为 String(true, false)外，不允许转换为整型或浮点类型。
- 数值类型的运算不会产生上溢或下溢异常，这一点值得注意。double d=100./0.;d 将为

Infinity。Java 中没有 C++那样的 signed 和 unsigned 整型。

- Java 中浮点型向整型转换的过程中默认采用的是去尾法。
- 注意 Java 语言中的 final 量与 C++中的 const 量是有区别的：一个 final 变量是只能赋值一次的量，这个 final 量是可以通过类中的方法来改变的，但是 C/C++的 const 量却不能再改变。类申明中加 final 保证此类不能再被继承。Java 中 void f(final int i)类似于 C++中 void f(const int i)，而 void g(final Bob b)却完全不同于 C++中的 void g(const Bob& b)。
- 在 Java 中方法也可以被定义为 final 的，final 方法不能在继承时重定义，可能获得更好效率（编译器可以将它变成内联函数）。
- 在 Java 中所有的引用类型都在堆中创建，不像 C++中有在 heap 和 stack 中创建两种选择， 在 Java 中 new 关键子的使用率远比在 C++中使用高。在 C++中，RealPoint rp; 已经建立对象 rp 并进行了初始化（假定 RealPoint 有默认构造函数）。而在 Java 中，RealPoint rp;只是申明了一个 RealPoint 的引用，未进行任何初始化，它的值为 null。Java 中也不允许用 Integer ia(3);方式来定义一个 Integer,而只能通过 Integer ia=new Integer(3)的方式。
- Java 语言中，数组类型是一种普通类型。数组的长度通过访问其公有的 length 变量得到。注意，语言 Point[] ia=new Point[5];只是定义了长度为 5 的 Point 数组引用，任何一个都未进行初始化。
- 所有的 Java 基本类型（primitive types）都有其对应的包装类(Wrapper Classes)。包装类的作用一是为某一类型相关的方法和变量提供一个家，二是将原始类型包装成类以便为某些只能处理对象引用的类使用，比如 HashMap 等 Collection 对象。
- 包装类包括 Boolean, Character, Void, Number(Byte,Short, Integer, Long, Float, Double)。
- 应注意 Float 和 Double 对象的比较与 float 和 double 值的比较是不同的。每一包装类定义了一个顺序：-0 小于+0,NaN 大于所有值(包括+∞)，所有的 NaN 相等。

表 1.2　Java 与 C++的比较 2

项 目	Java	C++
类型	长度固定，无 signed, unsigned 整型，无 sizeof 运算符。整型默认都是 signed 的。	长度随系统、机器不同，有 signed,unsigned 整型，有 sizeof 运算符
类型初始化	类定义中进行默认初始化	类中用构造函数初始化
对象创建方式	堆中创建	堆中或栈中创建
常量表示	final，但是与 C++之常量有很大区别。	const
局域变量重定义	不允许	允许
局域变量未初始化使用	不允许	允许

1.3 Java 类与对象

- Java 中的变量名与方法名可以重名（但这通常不是好的做法），C++中不能。
- Java 是单继承的语言，所有的类都是从 Object 类继承的，所以可以使用 Object 类中定义的 toString(),clone(), hashCode(), equals（）等方法。
- Java 中非 static 变量可以在定义时进行初始化（这代替了 C++中初始化表达式的功能），而 C++中只能通过初始化表达式或在构造函数中初始化，C++中非 static 变量初始化会产生编译错误。
- Java 类中默认的访问（即既不加 private, public 或 protected 关键字）控制是包访问，它与 private, public, protected 都不同。C++中类的默认访问控制是 private。另外 Java 中的访问控制是分开定义的（即每个字段或方法分开定义），而 C++中是集中定义的（即通过 public: private:的方式）。
- Java 的函数申明中不能使用默认参数。这似乎会造成在有多个构造函数时需要做更多的工作，但是 Java 中允许在一个构造函数中调用另一个构造函数，实际上也很方便。但是应注意，如果调用其它构造函数，必须在构造函数体的第一个语句使用。
- Java 中 this 代表对象本身，所以用 this.x, this.y 方式可以访问对象本身的 x,y 域。C++中 this 是一个指向自身的指针，必须用 this->x, this->y 方式访问 x,y 域。C++用 this.x,this.y 会产生编译错误。
- Java 语言中没有运算符重载的功能。唯一一个重载的运算符是 String 中定义的"+"运算符。利用这一个运算符，以及 Object 的 toString()方法，可以实现类似 C++中重载<<运算符的功能。
- Java 中无 const 支持（final 不能实现类似 C++中 const 的功能），所以要防止改变对象，只有靠用户自己控制（可以使用 clone()方法对对象进行拷贝，然后使用）。
- **Java 方法内的参数传递方式，**依照文献【3】的说法，**"Java 语言不是按引用传递对象，而是按值传递对象引用。"**很多文献会说是**"Java 方法内引用类型按引用传递，基本类型按值传递"。**尽管一般也可以这样说，但严格地说，这种说法是有问题的！按引用传递意思是，当一个参数传递给一个函数，函数得到一个原始对象的引用，而不是一个值的拷贝。引用是只能一次赋值的，赋值后不能再改变，而 Java 中并不是如此。比如：

```
class Body {
    public long idNum;
    public String name="<unnamed>";
    public Body orbits=null;
    private static long nextID=0;

    {
        idNum=nextID++;
    }
    public Body(String bodyName,Body orbitsAround){
        name=bodyName;
        orbits=orbitsAround;
```

```
    }
    public String toString(){
        String desc=idNum+" ("+name+")";
        if(orbits!=null)
        desc+=" orbits "+orbits.toString();
        return desc;
    }
}
class BodyRef {
   public static void main(String[] args){
       Body sirius=new Body("Sirus",null);
       System.out.println("before "+sirius);
       commonName(sirius);
       System.out.println("after "+sirius);
   }
   public static void commonName(Body bodyRef) {
      bodyRef.name="Dog Star";
      bodyRef=new Body("Other",null);
   }
}
```

输出结果为:

```
before 0 (Sirus)
after 0 (Dog Star)
```

尽管 bodyRef 已经改为”Other”，但是这并没有改变 sirius 对象。

如果在 C++中使用以下代码:

```
class Body {
    long nextID;
public:
     long idNum;
     string name;
     Body(string bodyName){
         name=bodyName;
     }
     string toString(){
         string desc=" ("+name+")";
         return desc;
     }
};

void commonName(Body& bodyRef) {
    bodyRef.name="Dog Star";
    Body newBody("Other");
    bodyRef=newBody;
}

int main(void) {
    Body sirius("Sirius");
    cout<<"before "<<sirius.toString()<<endl;
    commonName(sirius);
    cout<<"after "<<sirius.toString()<<endl;
}
```

则运行结果为:

```
before  (Sirius)
after  (Other)
```

而 C++中才是真正的按引用传递。

- **Java** 中没有拷贝构造函数，按 C++的方式定义拷贝构造函数会产生问题。
- Native 方法是用平台相关的代码实现的方法。
- 在 Java 中，若基类中有两个 public 同名方法，派生类 overide 了其中一个，派生类将继

续继承另外一个；这与 C++中一个同名方法将 overide 所有基类方法的方式是不同的。

- 应能分别 Java 中的 shadowing(阴影化)、obscuring(屏蔽)和 hiding(隐藏)和 overriding(：

(1) Shadowing 指一些声明将被另一同名的声明屏蔽掉，以致用单独一个名字难以访问，比如构造函数中 A(int i){this.i=i;}中类定义中的 i 就被局域的 i 屏蔽（shadow）掉了。本文件中定义的类也可能屏蔽掉 import 进的类。

(2) Obscuring 是指一个名字可以被解释为变量名、类型名或者包名。在这种情况下，编译器将优先将其解释为变量名，然后是类型名，最后才是包名。因此，有可能无法访问一个可见的类型名或包名，这时称声明被 obscured. 遵守 Java 的命名规则[4]可以防止 obscuring。

(3) Hiding.如果一个类定义一个 static 方法，这个定义将 hide 基类或基接口的所有同名声明。派生类中定义的变量将 hide 基类中的同名变量。

(4) Overriding 指派生类对基类的方法进行重定义，使派生类拥有新的表现。

表 1.3 Java 与 C++的比较 3

项 目	Java	C++
继承方式	单重继承	多重继承
变量名与方法名是否可重名	可以	不能
缺省参数	不能使用	可以使用
this	对象本身引用	本对象指针
运算符重载	不可	可
拷贝构造函数	不能用	经常使用
参数传递方式	按值传递（但是引用类型传递的是对象引用的拷贝，所以非常类似传递了一个引用，而基本类型则与 C++类似）	默认按值传递
默认访问权限	包访问权限	private
访问权限定义方式	单独定义	集中定义

[4] 临时变量：b（byte）,c(char), d(double), e(Exception), f(float), i,j,k(Integer), l(long),o(Object),s(String), v(某一类型的任意值)。类名大写开头，方法名小写开头，常量（大写，下横杠分隔）。

1.4 继承和多态

- 对象、代码重用的重要机制是继承。Java 的继承方式是单重继承，这与 C++的多重继承方式有着根本区别。也正是由于这一区别，使得 Java 在使用时与 C++有着很大不同。
- 所有的 Java 对象都从 Object 基类继承的。C++没有这种类似的基类，所以它的多重继承性也是逻辑上必然的。
- Java 有 Interface 对象，它类似一种纯虚类，通过 Interface 可以实现多重继承。C++没有 Interface 对象。
- 派生类的构造函数可以调用基类的构造函数，如果不是调用默认构造函数，需要显式调用，并且必须是在第一句调用。
- Java 没有析构函数，其资源的回收通过垃圾收集器来实现；C++通过析构函数机制来安全消除对象。
- **在 Java 中，如果基类中一方法名被重载许多次，在派生类中重定义这一方法名不会隐藏任一基类中的同名方法。这与 C++有着不同。**
- Java 中类继承用 extends 关键字，而 C++中用“ ：”来实现。
- Java 中继承与组合的选择原则与 C++一样，is-a 关系时用继承，has-a 关系时用组合。
- Java 中类可以被定义为 final 的，final 类不能被继承。
- Alan Kay 关于纯粹面向对象语言的 5 条规则：
  （1） Everything is an object.(任何东西都是对象)；
  （2） A program is a branch of objects telling each other what to do by sending messages.(一个程序是一组相互传递信息的对象的集合)；
  （3） Each Object has its own memory made up of other objects. (任一对象有它自己的由其它对象组成的内存)。
  （4） Every object has a type.(任一对象都有类型)。
  （5） All objects of a particular type can receive the same messages.(所有同一类型的对象能接收相同信息)。
- 面向对象语言的三个基本特征是：数据抽象、继承和多态。
- **Java 中所有方法默认都是使用晚绑定的，除非方法是 static 或者 final 或者 private(private 方法默认是 final 的)。而 C++中函数有晚绑定与早绑定的区别。晚绑定通过虚函数来实现。**
- Java 中定义有抽象(abstract)方法的类是一个抽象类，这与 C++中至少有一个纯虚函数的类是抽象类类似。但是**在 Java 中抽象类必须用 abstract 关键字显式说明，而 C++中并没有抽象类关键字。即 C++中抽象类是由是否定义纯虚函数决定的，而 Java 中是由 abstract 关键字决定的，Java 中可以定义一个类为 abstract 而不定义任何 abstract 方法。**

- Java 中建构对象的顺序如下：
  （1） 调用基类的构造函数；
  （2） 成员按定义顺序初始化；
  （3） 派生类的构造函数被调用。
- Java 构造函数中不要轻易调用方法，因为方法在 Java 中默认是晚绑定的。这可能会造成基类构造函数调用派生类的方法。使用构造函数的原则是在保证对象进入正确状态的前提下做尽量少的事，不要调用任何方法。

表 1.4 Java 与 C++的比较 4 （继承）

项 目	Java	C++
继承方式	单重继承	多重继承
Interface 类	有	无
析构函数	无	有
方法默认绑定方式	晚绑定	早绑定
重定义方法	不隐藏基类同名方法（重载）	隐藏基类所有同名方法
继承关键字	extends（类）， implements(接口)	无，用“:”实现
抽象类	用 abstract 关键字实现	通过定义纯虚函数实现

1.5 命名控制

- C++的命名控制主要通过引进命名空间(namespace)来实现的，而 Java 的命名控制则通过包（package）定义。
- Java 中 field 和 method 和成员类（或接口）可以重名，编译器对两者的区分是通过不同的使用环境。
- Java 的一般命名规则：
  （1） 类名：大写开头，接连词首字母大写；类名应该是描述性名词或名词短语，不要太长。
  （2） 字段名：一般小写开头，级联词首字母大写；final static 字段一般全部大写，各单词间用下横杠隔开。
  （3） 方法名：动词或动词短语，首字母小写，接连词首字母大写；getV 和 setV 是获取和设置变量 V 的值；获取某量长度的方法名应被命名为 length，象 String 方法中一样[5]；boolean 字段 V 的值通过 isV 方法来获取；将对象转化为某一特定格式 F 时应用 toF 方法来命名。

[5] 应区别 String 变量与数组获取长度的不同方式，String 通过 length()方法获取长度，而数组通过 length 字段（public）来获取长度。

（4） 临时变量的命名规则：b(byte)，c(char)，d(double)，e(Exception)，f(float)，i,j,k(integer)，l(long)，o(Object)，s(String)，v(某类型的随机值).。

- java.lang 包是每一编译单元自动 import 的，所以不用重新引入。
- 最高层 declaration 不能有 protected，private 或 static 修饰语，而只能无修饰词，或者有 public 修饰词。
- 包的命名按照逆国际域名的规则来实现。另外，如果域名包含 hyphen 或者在标识符中不允许的其它字符，将它转化为下横杠，如果是关键字，加 underscore, 如果以数字开头，前面加 underscore。

表 1.5 Java 与 C++的比较 5 （命名控制）

项 目	Java	C++
命名控制主要机制	包控制	命名空间
字段名和变量名重名	可	不可

1.6 抽象类、接口和内部类

- 接口(interface)是抽象类概念的进一步深化。接口可以看成是一个完全抽象的类，是一个没有任何实现的类，是抽象方法的集合。
- 接口除了可以包含方法外，也可以包含 field，接口内的 field 默认是 public、final 和 static 的。
- 接口中的方法默认是 public 的。
- 接口的作用不仅仅是一个抽象类更纯的抽象类。因为接口没有任何实现，所以一个类可以 implments 不止一个接口，这提供了一种实现多重继承的方式。所以接口存在的原因除了与抽象类存在的原因一致外，还可以帮助将一个对象映射到多个基类。
- 如何选择接口和抽象类：接口即提供抽象类的优点也能提供接口本身独特的优点，所以应该优先选择接口类，只有你想拥有方法定义和成员变量时才应该考虑将其转化为抽象类。
- 因为接口内的变量默认是 static 和 final 的，可以用接口定义常数集合。
- 内部类是 Java 又一个解决多重继承问题的方案。通过内部类可以继承任意多的类，而所有的内部类都可以方便地使用外部类的变量和方法。
- 一个外部类可以拥有多个内部类，每一个内部类可以以不同的方式 implements 同一接口，继承相同类。内部类可以拥有不同实例，每一个可以拥有自己不同的状态。
- 内部类自动可引用外部类的所有元素（包括其私有元素），这也是内部类的一大优势。
- 内部类可以是无名的，无名类常在定义 listener 类时使用。
- Java 中内部类的特点和优点使得 Java 中内部类的使用远比 C++要频繁得多。
- 内部类如果定义为 static,通常称为嵌套类，嵌套类与 C++中的嵌套类类似（但 Java 嵌套

类可以访问 private 元素）。嵌套类可以拥有 static 数据，而普通内部类不能拥有 static 数据。

- 局部内部类（Local Inner Classes）是定义在方法、构造函数或者初始化块中的类。局部内部类不是类的成员。
- 接口内也可以定义内部类，这是很好的实现共享的可更改的资源的方式，因为接口中的内是 public 和 static 的。比如：

```
interface SharedData{
    class Data{
        private int x=0;
        public int getX(){return x;}
        public void setX(int newX){x=newX; }
    }
    Data data=new Data();
}
```

表 1.6 Java 与 C++的比较 6 （接口和内部类）

项 目	Java	C++
接口	有且很重要	没有
内部类是否与外部类有联系	有	不大

1.7 异常处理

- Java 与 C++在异常处理方面的区别是：Java 的异常处理是与语言本身紧密结合在一起的，如同 Garbage collector 一样，你被迫使用异常。而 C++的异常的异常处理是后来新加的一个特性。
- Java 的异常机制也与其同步模型结合在一起，所以当 synchronized 语句和 synchronized 方法突然中止时锁将被释放。
- Java 异常的层次结构如下：

Object－>Throwable－>Exception－>Error

－>Error

- 异常产生的原因可能有如下三种：

（1） Java 虚拟机检测到的同步异常：

（1.1）表达式运算违反通常语言语义

（1.2）装载和连接部分程序时出错；

（1.3）资源超限

（2） throw 语句被执行；

（3） 异步异常：

(3.1)Thread 类的 stop 方法被调用

（3.2）虚拟机发生内部错误。

- 异常可以分为同步异常（synchronous exception）和异步异常（Asynchronous exception），同步异常是指 throw 语句产生的异常。异步异常可能由两种原因产生：1)JVM 内部错误；2）使用 Thread.stop(deprecated)方法或 Java Virtual Machine Debug Interface (JVMDI)的 stopThread 方法。
- 同步异常可以分为 checked exceptions 和 unchecked exceptions。unchecked exceptions 是指 RuntimeException 和它的子类，Error 以及它的子类。所有其余的异常都是 checked 的异常。Checked exceptions 必须在程序中捕获处理。
- RuntimeException 和 Error 可以被继承，但是一般不要这样做，因为这样做会造成混乱。程序员应该使用 Checked Exceptions.
- Java 异常是 precise 的，这是说 Java 异常产生后，处理完毕不会自动在异常产生语句之后继续运行。
- Java 异常产生和处理的方法与 C++都有着不同：Java 产生时在方法中用 throws 说明，在内部用 throw 语句抛出，C++在任意地方抛出；Java 的构造函数中也可以抛出异常。Java 的处理方式如下：

  try{
  }catch(Exception1 e){
  }catch(Exception2 e){
  }finally{
  }
  C++中的处理方式为
  try{
  }catch(Exception1){

  }catch(Exception2){
  }catch(…){
  }
  C++中无 finally 语句，但可以用…表示所有异常。

表 1.7 Java 与 C++的比较 7（异常处理）

项 目	Java	C++
异常与语言结合程度	紧密，离开异常机制，没有 Java	附加属性
异常产生方式	throws, throw	throw
异常处理方式	见上文字说明	见上文字说明

1.8 包容器类

- Java 中对象的集合表示包括使用数组（Array）和集合类(Container class)
- Java 中的数组是一种类型，应注意如果只是定义 A[] a; a 只是一个 null 对象，没有被赋予任何值。
- Java 中数组只能由 int(int short byte char)值指标来获取，用 long 变量来获取数组值将产生编译错误。
- Java 中一个字符数组与字符串(String)是不同的。String 对象是不可变的（它的内容不能改变），而字符数组可以逐个改变其元素。String 对象中的 toCharArray()方法可以将一个 String 转化为一个字符数组。
- Java 的包容器类包括 Collection 类和 Map 类，Collection 类又包括 List 类和 Set 类。Map 类是键－值对的集合。Java 包容器类的常见类型和适用范围如下：
- 应注意 Java 包容器类和数组元素个数获取方式的不同：包容器类通过其 size()方法获取，而数组通过其公有 length 字段获取（String 通过其 length()方法获取）。
- 有了以上包容器类，一般没有必要使用 Java 1.0/1.1 的包容器类(Vector，Enumeration, Hashtable，Stack)。Vector 可以用 List 代替，Enumeration 用 Iterator 代替，Hashtable 用 HashMap 代替，Stack 用 LinkedList 代替。
- C++中的包容器类在<set><vector><list><queue><stack><valarray>中定义，它们都是模板类。C++中 vector<T>很常见，而 Java 中一般用 List. 他们都通过 size()方法来获取集合元素的个数。
- Java 中 Collection 类的使用方法如下：

```
Collection coll;
Iterator it=coll.iterator();
while(it.hasNext()){
    String str=(String)it.next();
    …
}
```

表 1.8 Java 与 C++的比较 8（包容器类）

项 目	Java	C++
包容器类的实现	Collection, Map	<set><vector><list><queue><stack><valarray>它们都是模板函数
最常用的集合类	List(ArrayList, LinkedList)	vector<T>

，

1.9 Java I/O

- java.io.File 是一个可以用来进行目录操作的类，File 既可能是一个文件，也可能是一个目录。
- Java I/O 的使用远比 C++要复杂得多，因为即使简单的键盘输入操作也不是轻易能实现的(C++的>>操作符可以轻易实现输入操作)。下面列出常见 I/O 操作的实现方式：

（1） 从标准输入读：

```
BufferedReader stdin=new BufferedReader(new InputStreamReader(System.in);
Stdlin.readLine();
```

（2） 文件输出：

```
BufferedReader in4=new BufferedReader(new StringReader(s2));
//BufferedReader in4=new BufferedReader(new FileReader(filename));
PrintWriter out1=new PrintWriter(new BufferedWriter(new FileWriter("IODemo.out")));
```

（3） 存储和恢复数据：

```
DataOutputStream out2=new DataOutputStream(new BufferedOutputStream(new
        FileOutputStream("Data.txt")));
out2.writeDouble(3.14159);
out2.wrtieUTF("That is pi");
out2.close();
DataInputStream in5=new DataInputStream(new BufferedInputStream(new
        FileInputStream("Data.txt")));
In5.readDouble();
In5.readUTF();
```

- Readers 和 Writers 类（Java1.1）与 InputStream 和 OutputStream 类（Java1.0）的区别是：

（1） Reader 和 Writer 类是基于 Unicode 和 character 的 I/O，而 InputStream 和 OutputStream 是面向 byte 的 I/O。

（2） Reader 类的派生类必须实现 Reader 类的抽象 close()方法，而 InputStream 只需继承空实现。

（3） Reader 是用 ready()方法来表示是否有数据，InputStream 用 available()方法告诉你有多少可读的数据。

两者的转换方式是：InputStreamReader 可以将 InputStream 转换为 Reader，OutputStreamWriter 可以将 OutputStream 转化为 Writer。

- Java 1.0 在 Java 1.1 中没有改变和对应的类包括 DataOutputStream，File，RandomAccessFile 和 SequenceInputStream。
- JDK 1.4 在 java.nio 包和 java.nio.channels 包中定义了新的 I/O 库，以增进速度：

ByteBuffer，Buffer，FileChannel

- System.currentTimeMillis()方法可以获取系统时间。
- Java 中包含以下压缩类，可以实现压缩功能：CheckedInputStream，CheckedOutputStream，DeflaterOutputStream， ZipOutputStream， GZIPOutputStream， InflaterInputStream，ZipInputStream，GZIPInputStream。
- 对象的串行化的方法如下：

  ObjectOutputStream out=new ObjectOutputStream(new FileOutputStream(“worm.out”));

  out.writeObject(“Worm storage\n”);

  out.writeObject(w);

  out.close();

  ObjectInputStream in new ObjectInputStream(new FileInputStream(“worm.out”);

  String s=(String)in.readObject();

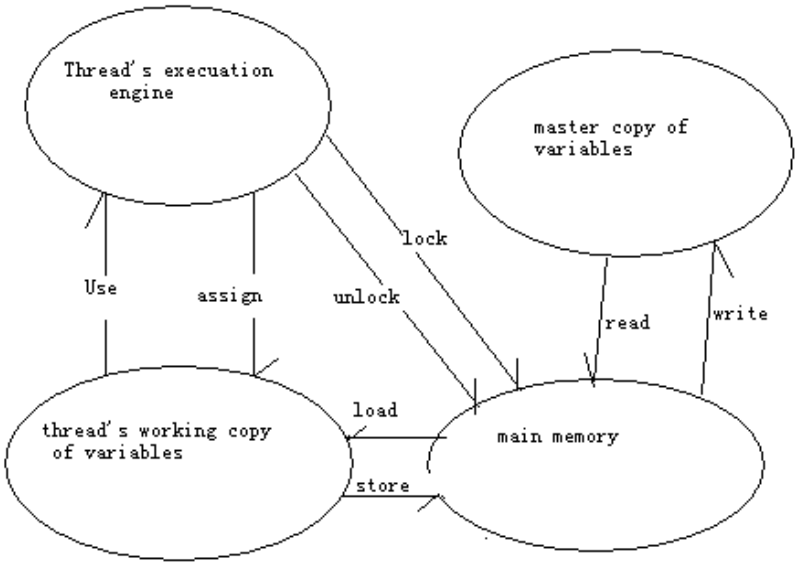
  Worm w2=(Worm)in.readObject();
- JDK 1.4 介绍了 Preferences API（java.util.prefs.*），可以方便地实现程序环境变量的自动存储（不必在退出时将环境变量存到另外的文件中），很有用。

表 1.9 Java 与 C++的比较 8（I/O）

项 目	Java	C++
标准 I/O 的实现方式	System.out, System.in 通过重载 Object 类中的 toString()方法实现输出。	<iostream> istream,ostream 一般通过重载<<, >>实现自定义输入输出。

1.10 多线程编程

- Java 是一种多线程的语言，实现多线程的机制已经内置于语言之中。Java 基类 Object 之 wait()，notify()，notifyAll()方法可以实现多线程之间的控制传递。
- 多线程的控制机制如下：



- 应理解死锁发生的四个条件（同时满足）：

（1） 互斥。至少有一个资源被多个线程共享；

（2） 至少有一个进程必须拥有一种资源，并等待另一进程占有的另一资源；

（3） 某一资源不能从一进程中被强夺；

（4） 环形等待发生，一个进程等待另一进程拥有的资源。

- 应注意 Java 语言本身不能防止或检测死锁。程序员应采用通常防止死锁的机制来防止和控制死锁。
- 一个线程可能处于以下四种状态之一：New，Runnable，Dead，Blocked。
- 实现多线程的方法有二：一是继承 Thread 类，用 start()启动 run()，在 run()方法定义线程的行为；二是 implements Runnable 接口，这种方法适用于类已经继承了其它类的场合。
- 守护进程（daemon thread）是指一个在后台提供服务的进程。当所有的非守护进程结束后，整个程序将中止：setDeamon(true)可以用来设置进程为守护进程。
- 应掌握 Thread 中 sleep()和 wait()方法的区别：sleep()不释放锁，wait()释放锁；用法：Object.wait()，Thread.sleep()。
- 掌握 Thread.jon()的作用：等待一个线程完成。

1.11 Java Windows 编程

- 应明白什么是 AWT，什么是 Swing。AWT 是 Java1.0 的产品(Abstract Window Toolkit)(java.awt.*)，Swing 是 Java 2 的 Java Foundation Classes(JFC)(javax.swing.*)
- 分别 JApplet, JFrame 和 JApplet。
- Java Windows 编程很容易，用很少的代码就可以实现 Windows，而且这些代码是可控制的。Java Windows 编程不会象某些 GUI builder 那样产生不可读代码。
- 应掌握 Java GUI 的 layout：BorderLayout，FlowLayout，GridLayout，GridBagLayout，BoxLayout。
- 掌握 Swing 的 Event 模型以及 Listener 的使用，这时候内部类经常是很有用的。
- 应明白什么是 Javabeans：Javabean 是遵守一定命名规则的特殊的 Java 类。

2．对 Java 与 C++的一般性比较

Java 是一种通用的、基于 class 的、支持并发处理的、强类型的、相对高级的面向对象的语言。Java 是一种从头开始设计的编程语言，它没有 C++那样的向前兼容的问题。Java 是比 C++更纯粹的面向对象的语言，采用 C++的语法，Smalltalk 的风格。它针对 C++的不足对 C++有一定改进，几乎可以保证不能运行坏的程序。Java 也是一种网络编程语言，几乎不会产生病毒，易于进行浏览器端，服务器端编程。Java 可以让更少的人使用更少的时间编写更大、更复杂的程序。但是同时 Java 也一直存在着速度问题。Java 不是一个操作系统，Java 也并不比 C++简单（Larry O’brien 曾说 Saying Java is simplier than C++ is like saying KZ is shorter than Everest）,也并不完美。应该说 Java 是比 C++更理想主义的一种语言，Java 更能满足理想主义者的口味。与 Java 相关技术也远远比 C++要多得多（J2SE,J2ME，J2EE，JSP, Servlet,Javabeans，JMS，Jini…）一个人可以宣称自己精通 C++编程，但是一个明智者不会随意宣称自己精通 Java。

3．对参考文献的评价

文献【1】是 Java2 规范，涉及了 Java 语言最基本的方面，值得一读。但它对于 Java Windows 编程,I/O 编程都无甚涉及。文献【2】很实用，涉及了 Java 编程的多个方面，很适合于有一定 Java 编程经验，想将自己对 Java 的理解提高到一个新的层次的读者阅读。文献【2】也可以作为 Java 编程的很好的教科书。其不足之处是对某些方面的探讨显得不够深入。文献【3】是 Java 的经典书籍，正如《C++ Programming Language》之于 C++的经典。它对 Java 语言的许多地方有较为深入的讨论或简明的概括（比如 Java 中参数传递方式的说明）。

4 附 录

4.1 Java 基本包功能描述

包 名	描　　述
java.lang	The main language classes, such as Object, String, Thread, Class, and so on
java.io	Input and output and some file system manipulation
java.util	Classes of general utility. Collection
java.security	Defines the platform security architecture
java.text	Internationlization and localization for formatting and parsing numbers and dates, sorting strings, and message lookup by key.
java.awt	The Abstract Window Toolkit
java.applet	Applets
java.beans	Components. JavaBeans is a component architecture that helps independent vendors write classes that can be treated as components of large systems assembled by users.
java.math	Mathematics. BigInteger and BigDecimal
java.net	Provides classes for working with network infrastructure, such as sockets, network addresses, and Uniform Resource Locators (URLs) Socket.URL
java.rmi	Remote Method Invocation. Gives you a way to create objects whose methods can be invoked from other virtual machines.
java.security	Security Tools
java.sql	Provides the Java Database Connectivity (JDBC) package for using relational databases.
java.util.jar	Provides classes for reading and writing the JAR (Java Archive) file fomat, which is based on the standard ZIP file format with an optional manifest file.
java.util.zip	Provides classes for reading and writing standard ZIP and GZIP files.
javax.accessibility	Accessibility for GUIs
javax.naming	Java Directory and Naming Services (JNDI)
javax.sound	javax.sound.midi provides interfaces for reading , writing, sequencing synthesizing data that conforms to the Musical Instrument Digital Interface (MIDI) format.
javax.swing	Swing GUI Components
org.omg.CORBA	CORBA APIs

4.2 Java RuntimeException 和 Error 类的继承类

类　名	父　类	说　明
ArithmeticException	RuntimeException	
ArrayStoreException		
ClassCastException		
ConcurrentModificationException		
EmptyStackException		
IllegalMonitorStateException		
IllegalStateException		
IndexOutOfBoundsException		
MissingResourceException		
NegativeArraySizeException		
NegativeArraySizeException		
NoSuchElementException		
NullPointerException		
SecurityException		
UndeclaredThrowableException		
UnsupportedOperationException		
ArrayIndexOutOfBoundsException	IndexOutOfBoundsException	
StringIndexOutOfBoundsException		
IllegalThreadStateException	IllegalArgumentException	
NumberFormatException		
LinkageError	Error	
ThreadDeath		
VirtualMachineError		
ClassCircularityError	LinkageError	
ClassFormatError		
ExceptionInIntializerError		
IncompatibleClassChangeError		
NoClassDefFoundError		
UnsatisfiedLinkError		
VerifyError		
InternalError	VirtualMachineError	
OutOfMemoryError		

StackOverflowError		
UnknownError		
AbstractMethodError	IncompatibleClassChangeError	
IllegalAccessError		
InstantiationError		
NoSuchFieldError		
NoSuchMethodError		
UnsupportedClassVersionError	ClassFormatError	

4.3 Java 常见的 Input/Output 类和使用方法

Input 类(Class)	功能(Function)	Output 类
ByteArrayInputStream	允许内存 buffer 作为一个 InputStream	ByteArrayOutputStream
StringBufferInputStream	将 String 转化为 InputStream	
FileInputStream	从文件读取信息,构造函数参数:代表文件名的 String,或者 File 对象	FileOutputStream
PipedInputStream	与 PipedOutputStream 相联系，实现 piping 概念	PipedOutputStream
SequenceInputStream	将两个或更多 InputStream 转化为一个 InputStream	
FilterInputStream	向其它 InputStream 类提供有用功能的抽象类	FilterOutputStream
DataInputStream	与 DataOutputStream 协同使用,可以从流中读取基本类型（primitives）	DataOutputStream
	产生格式化的输出，DataOutputStream 处理数据的存储，PrintStream 处理显示	PrintStream
BufferedInputStream	防止每次读时直接物理读取，意味着使用 Buffer	BuffedOutputStream
LineNumberInputStream	跟踪输入流中的行数,可以调用 getLineNumber() 和	

	setLineNumber(int)方法	
PushbackInputStream	有一个字节放回 buffer 的 InputStream	
Reader	与 InputStream 对应，adapter:InputStreamReader	Writer
FileReader	与 FileInputStream 对应	FileWriter
		StringWriter
StringReader	StringBufferInputStream	
CharArrayReader	ByteArrayInputStream	CharArrayWriter
PipedReader	PipedInputStream	PipedWriter
FilterReader		FilterWriter
BufferedReader		BufferedWriter
		PrintWriter

4.4 Java 包容器类列表

接 口			特点和适用范围
Collection	List(接口)	ArrayList	可以快速随机访问
		LinkedList	便于插入和删除
	Set（接口）	HashSet	快速查询
		TreeSet	用 tree 支持的 set，可以从 Set 中获取有序列
		LinkedHashSet（JDK1.4）	与 HashSet 的查找速度相同，但是保持插入元素的顺序
Map	HashMap		基于 HashTable 的 Map。对插入和定位有序对有常数性能
	LinkedHashMap（JDK1.4）		与 HashMap 类似，但在遍历时，以插入顺序或者最少最近使用顺序(LRU)
	TreeMap		用红-黑树实现的 Map，查看时，会以排序顺序
	WeakHashMap		弱键 Map,允许对象通过将被释放的 map 来引用
	IdentityHashMap（JDK1.4）		用==而不是用 equals()方法来比较键值的 hash map。

4.5Java 常见 Windows 组件

Java Windows 组件	说明
JButton	
JTextField	
JTextArea	
JPanel	
JButton	按钮
ButtonGroup	
JLabel	
JCheckBox	
JRadioButton	Radio Button
JScrollPanes	滚动条
JTextPane	文本 Pane
JComboBox	多选框
JList	
JTabbledPane	
JOptionPane	选择 Pane
JMenus, JMenuBar, JMenuItems	菜单
JPopupMenu	弹出菜单
JDialog	对话框
JFileChooser	文件选择框
JSlider	滑动条
JProgressBar	进度条

5 参考文献

【1】James Gosling, Bill Joy, Guy Steele and Gilad Bracha, The Java Language Specification Second Edition, ISBN 0-201-31008-2

【2】Bruce Eckel, Thinking in Java(Third Edition), 英文影印版.机械工业出版社.2004

【3】Ken Arnold, James Gosling, David Holmes. The Java Programming Language. Third Edition.(影印版). 北京：中国电力出版社. 2003

廖海仁　2006.9-2007.4

2007.12 Revised.