

第四节 线程的互斥

[单选、填空]在 Java 语言中,引入了“对象互斥锁”的概念,也称为监视器,使用它来实现不同线程对共享数据操作的同步。“对象互斥锁”阻止多个线程同时访问同一个条件变量。Java 可以为每一个对象的实例配有一个“对象互斥锁”。

[简答]在 Java 语言中,有两种方法可以实现“对象互斥锁”:

- (1)用关键字 volatile 来声明一个共享数据(变量)。
- (2)用关键字 synchronized 来声明操作共享数据的一个方法或一段代码。

第五节 线程的同步

[单选、填空]为了解决线程运行速度问题,Java 提供了一种建立在对象实例之上的交互方法。Java 中的每个对象实例都有两个线程队列和它相连。第一个用来排列等待锁定标志的线程。第二个则用来实现 wait() 和 notify() 的交互机制。

[填空]类 java.lang.Object 中定义了 3 个方法,即 wait()、notify() 和 notifyAll()。wait() 方法导致当前的线程等待,它的作用是让当前线程释放其所持有的“对象互斥锁”,进入 wait 队列(等待队列);而 notify() / notifyAll() 方法的作用是唤醒一个或所有正在等待队列中等待的线程,并将它们移入等待同一个“对象互斥锁”的队列。

Java 语言程序设计(一)

目 录

第一章 Java 概述/1

第二章 数据和表达式/2

第三章 流程控制语句/5

第四章 面向对象程序设计/8

第五章 数组和字符串/12

第六章 继承与多态/15

第七章 输入和输出流/18

第八章 图形界面设计/22

第九章 Swing 组件/26

第十章 多线程/28

进程之中,由进程触发执行。在系统资源的使用上,属于同一进程的所有线程共享该进程的系统资源,但是线程之间切换的速度比进程切换要快得多。

[简答]在 Java 中,线程由以下 3 部分组成:

(1)虚拟 CPU,封装在 `java.lang.Thread` 类中,它控制着整个线程的运行。

(2)执行的代码,传递给 `Thread` 类,由 `Thread` 类控制按序执行。

(3)处理的数据,传递给 `Thread` 类,是在代码执行过程中所要处理的数据。

[单选、填空]线程一共有 4 种状态,分别是新建(New)、可运行状态(Runnable)、死亡(Dead)及阻塞(Blocked)。

第二节 创建线程

[填空]创建线程有两种方法,一种是定义一个继承 Thread 类的子类,另一种是实现 Runnable 接口。

[简答]创建线程两种方法的适用条件:

(1)适用于采用实现 `Runnable` 接口方法的情况。因为 Java 只允许单重继承,如果一个类已经继承了 `Thread`,就不能再继承其他类,在一些情况下,这就被迫采用实现 `Runnable` 的方法。另外,由于原来的线程采用的是实现 `Runnable` 接口的方法,可能会出于保持程序风格的一贯性而继续使用这种方法。

(2)适用于采用继承 `Thread` 方法的情况。当一个 `run()` 方法置于 `Thread` 类的子类中时, `this` 实际上引用的是控制当前运行系统的 `Thread` 实例。

第三节 线程的基本控制

[填空]虽然一个线程已经被创建,但它实际上并没有立刻运行。要使线程真正地在 Java 环境中运行,必须通过方法`start()`来启动, `start()` 方法也在 `Thread` 类中。

[填空]在 Java 中,线程调度通常是抢占式,而不是时间片式。抢占式调度是指可能有多个线程准备运行,但只有一个在真正运行。一个线程获得执行权,这个线程将持续运行下去,直到它运行结束或因为某种原因而阻塞,再或者有另一个高优先级线程就绪,最后一种情况称为低优先级线程被高优先级线程所抢占。

[简答]每个线程都有一个优先级,Java 的线程调度采用如下的优先级策略:

(1)优先级高的先执行,优先级低的后执行。

(2)每个线程创建时都会被自动分配一个优先级,默认时,继承其父类的优先级。

(3)任务紧急的线程,其优先级较高。

(4)同优先级的线程按“先进先出”的调度原则。

[单选、填空]当一个线程从 `run()` 方法的结尾处返回时,它自动消亡并且不能再被运行,可以将其理解为自然死亡。另一种情况是遇到异常使得线程结束,可以将其理解为强迫死亡。还可以使用 `interrupt()` 方法中断线程的执行。

[填空]有几种方法可以用来暂停一个线程的运行,暂停一个线程也称为挂起。在挂起之后,必须重新唤醒线程进入运行状态,只是线程执行命令的速度非常慢。

第四节 对话框

[填空]对话框是一个临时的可移动窗口,且要依赖于其他窗口,当它所依赖的窗口消失或最小化时,对话框也将消失。当窗口还原时,对话框会自动恢复。一般地,要先创建一个窗口类,再创建一个对话框类,且让对话框依附于窗口。

[填空]对话框分为强制型和非强制型两种。强制型对话框被关闭之前,其他窗口无法接收任何形式的输入,也就是该对话过程不能中断,这样的窗口也称为模式窗口。非强制型对话框可以中断对话过程,去响应对话框之外的事件。

[单选、填空]JDialog 类通常用于创建自定义的对话框,除此之外,在 Swing 中还提供了用于显示标准对话框的JOptionPane 类。在 JOptionPane 类中定义了多个 showXxxDialog 形式的静态方法,可以分为以下 4 种类型:

- (1) showConfirmDialog, 确认对话框, 显示问题, 要求用户进行确认(yes/no/cancel)。
- (2) showInputDialog, 输入对话框, 提示用户进行输入。
- (3) showMessageDialog, 信息对话框, 显示信息, 告知用户发生了什么情况。
- (4) showOptionDialog, 选项对话框, 显示选项, 要求用户进行选择。

[程序设计]文件对话框是专门用于对文件(或目录)进行浏览和选择的对话框,常用的构造方法有 3 种形式:

- (1) JFileChooser(), 构造一个指向用户默认目录的文件对话框。
- (2) JFileChooser(File currentDirectory), 使用给定的 File 作为路径来构造一个文件对话框。
- (3) JFileChooser(String currentDirectoryPath), 构造一个使用给定路径的文件对话框。

第十章 多线程

第一节 线程和多线程

[单选、填空]在程序要投入运行时,系统从程序入口开始按语句的顺序(包括顺序、分支和循环结构)完成相应指令直至结尾,再从出口退出,整个程序结束。这样的语句结构称为进程,它是程序的一次动态执行,对应了从代码加载、执行至执行完毕的一个完整过程;或者说进程就是程序在处理机中的一次运行。在这样一个结构中不仅包括了程序代码,同时也包括了系统资源的概念。具体来说,一个进程既包括其所要执行的指令,又包括执行指令所需的任何系统资源,如 CPU、内存空间、I/O 端口等,不同进程所占用的系统资源相对独立。

[单选、填空]线程是进程执行过程中产生的多条执行线索,是比进程单位更小的执行单位,在形式上同进程十分相似——都是用一个按序执行的语句序列来完成特定的功能。不同的是,它没有入口,也没有出口,因此其自身不能自动运行,而必须栖身于某一个

第一章 Java 概述

第一章 Java 概述

第一节 Java 语言简介

[填空]Java 语言的前身是Oak 语言,是美国 Sun Microsystems 公司于 1991 年推出的,仅限于公司内部使用的语言。

[单选、填空]Java 是一种功能强大的程序设计语言,既是开发环境,又是应用环境,它代表一种新的计算模式。

[简答]Java 语言的特点:

- (1) 语法简单,功能强大,安全可靠。
- (2) 与平台无关。
- (3) 解释编译两种运行方式。
- (4) 多线程。
- (5) 动态执行兼有丰富的 API 文档及类库。

第二节 Java 开发环境的安装与设置

[单选、填空]JDK(Java SE Development Kit, Java 语言软件开发工具包)是原 Sun 公司(现已被 Oracle 公司收购)提供的软件包,其中含有编写和运行 Java 程序的所有工具,包括组成 Java 环境的基本构件。

[单选、填空]下载完毕,找到下载文件所在的目录,双击 jdk-8u131-windows-x64.exe 直接运行,开始安装 JDK。按照安装向导进行安装即可。

[单选、填空]安装完成后,需要设置环境变量。环境变量设置完毕。重启计算机让这些设置生效。

第三节 Java 程序示例

[单选、填空]Java 程序分为两种,一种是Java 应用程序(Java Application),另一种是Java 小应用程序(Java Applet),或叫 Java 小程序。

[填空]Java 程序由类构成,含有一个 main() 方法,称为主方法或主函数。程序是通过 Java 解释器来执行的独立程序,可以使用命令行命令直接运行。整个程序的运行入口是 main() 方法,main() 方法执行完毕,整个程序也即结束。

[单选、填空]一个程序可以包含一个或多个.java 文件。不论文件个数有多少,其中只能有一个 main() 方法。

[单选、填空]源文件是文本形式的文件,Java 的执行系统是不能识别的,它必须经过编译,生成字节码的类文件后才能运行。类文件是二进制格式的,它有统一的格式,JVM 可以识别类文件并执行它。

[填空]编译一个程序的命令格式是:

`javac [选项] 源文件名`

如果在命令行中输入 `javac`, 则系统会显示所有选项。这些选项都是可选的。如果源文件不在当前目录, 则需要在文件名的前面加上目录。

[单选、填空] 运行一个 Java 程序的命令格式是:

`java [选项] 程序名 [参数列表]`

`java` 是解释器的名字, 表示要运行一个由“程序名”指定的程序。程序名也就是类的名字, 后面的参数列表是可选的。如果想向程序传递参数值, 则可以把这些参数依次列在程序名的后面, 参数个数不限。

[单选、填空] IDE 是集成开发环境的缩写, 这是一个提供给开发人员使用的程序开发环境, 通常包括了代码编辑器、编译器、调试器和图形用户界面等工具。

第四节 使用 Java 核心 API 文档

[单选、填空] JDK 文档中有许多 HTML 文件, 这些是 JDK 提供的应用程序编程接口 (Application Programming Interface, API) 文档, 可使用浏览器查看。API 是原 Sun 公司提供的使用 Java 语言开发的类集合, 用来帮助程序员开发自己的类和程序。最基本的是 Java 核心 API。

[填空] 核心 API 文档是按层设计的, 以主页方式提供给用户。

第五节 Java 中的面向对象技术

[填空] 所谓面向对象的方法学, 就是使分析、设计和实现一个系统的方法尽可能地接近人们认识一个系统的方法。通常包括 3 个方面: 面向对象的分析 (Object-Oriented Analysis, OOA)、面向对象的设计 (Object-Oriented Design, OOD) 和面向对象的程序设计 (Object-Oriented Programming, OOP)。

[单选、填空] OOP 技术把问题看成是相互作用的事物的集合, 也就是对象的集合。对象具有两个特性, 一是状态, 二是行为。状态是指对象本身的信息, 行为是实现对对象的操作。

[单选、填空] OOP 中采用了三大技术: 封装、继承和多态。封装体现的特点是将对象的属性及实现细节隐藏起来, 只给出如何使用的信息。将数据及对数据的操作捆绑在一起成为类, 这就是封装技术。

第二章 数据和表达式

第一节 基本语法元素

[单选、填空] 在 Java 程序中, 换行符及回车符都可以表示一行的结束, 它们可被看作是空白。另外, 空格键、水平定位键 (Tab) 亦是空白。为了增加程序的可读性, Java 程序的元素之间可以插入任意数量的空白, 编译器会忽略掉多余的空白。

[简答] Java 中的 3 种注释形式:

(1) 表示从 “`/*`” 开始一直到行尾均为注释, 一般用它对声明的变量、一行程序的作用

[程序分析、程序设计] 列表 (JList) 是可供用户进行选择的一系列可选项, 常用的构造方法如下:

(1) `JList()`, 构造一个空列表。

(2) `JList(Object[] listData)`, 构造一个列表, 列表的可选项由对象数组 `listData` 指定。

(3) `JList(Vector<?> listData)`, 构造一个列表, 使其显示指定 `Vector` 中的元素。

第二节 文本组件

[单选、填空] 文本域是一个单行的文本输入框, 可用于输入少量文本。

[简答] 文本区是一个多行多列的文本输入框, 常用的构造方法如下:

(1) `JTextArea()`, 构造一个空文本区。

(2) `JTextArea(String text)`, 构造一个显示指定初始字符串的文本区, `String` 型参数 `text` 指定要显示的初始字符串。

(3) `JTextArea(int rows, int columns)`, 构造一个具有指定行数和列数的空文本区, `int` 型参数 `rows` 和 `columns` 分别指定文本区的行数和列数。

(4) `JTextArea(String text, int rows, int columns)`, 构造一个具有指定行数和列数并显示指定初始字符串的文本区, `String` 型参数 `text` 指定要显示的初始字符串, `int` 型参数 `rows` 和 `columns` 指定文本区的行数和列数。

第三节 菜单组件

[填空] 菜单栏是窗口中的主菜单, 用来包容一组菜单。

[单选、填空] 如果将整个菜单系统看作是一棵树, 那么菜单项就是这棵树的叶子, 是菜单系统的最下面一级。

[简答] 常用的菜单项构造方法有:

(1) `JMenuItem()`, 创建不带有设置文本或图标的菜单项。

(2) `JMenuItem(Icon icon)`, 创建一个只显示图标的菜单项, 图标由 `Icon` 型参数 `icon` 指定。

(3) `JMenuItem(String text)`, 创建一个只显示文本的菜单项, 文本由 `String` 型参数 `text` 指定。

(4) `JMenuItem(String text, Icon icon)`, 创建一个同时显示文本和图标的菜单项, 文本由 `String` 型参数 `text` 指定, 图标由 `Icon` 型参数 `icon` 指定。

(5) `JMenuItem(String text, int mnemonic)`, 创建一个显示文本并且有快捷键的菜单项, 文本由 `String` 型参数 `text` 指定, 快捷键由 `int` 型参数 `mnemonic` 指定。

[单选、填空] 复选菜单项和单选菜单项是两种特殊的菜单项, 在复选菜单项前面有一个小方框, 在单选菜单项前面有一个小圆圈。可以对这两类菜单项进行选中或不选中的操作, 使用方法与复选按钮和单选按钮类似。

义的颜色。另一种方法是通过红、绿、蓝三原色的值来组合。每种颜色由 3 个值来指定，它们一起称为 RGB 值，RGB 分别代表红、绿、蓝。各个值表示对应原色的相对值，使用 1 个字节(8 位)来保存，取值范围为 0 ~ 255。3 种原色的值合在一起决定实际的颜色值。

[程序设计]显示文字的方法主要有以下 3 种：

(1) public void drawChars(char[] data, int offset, int length, int x, int y)，使用此图形上下文的当前字体和颜色显示字符数组 data 中从 offset 位置开始、最多 length 个字符。首字符的基线位于此图形上下文坐标系统的(x,y)处。

(2) public void drawString(String aString, int x, int y)，在指定位置显示字符串 aString。

(3) public void drawBytes(byte[] data, int offset, int length, int x, int y)，使用此图形上下文的当前字体和颜色显示由指定的 byte 数组 data 中从 offset 位置开始、最多 length 个字符。首字符的基线位于此图形上下文坐标系统的(x,y)处。

[填空]文字字形有字体、样式及字号 3 个要素。

[单选、填空]Graphics 类是所有图形上下文的抽象父类，允许应用程序在组件以及屏幕图像上进行绘制。这个类提供的功能有：建立字体、设定显示颜色、显示图像和文本、绘制和填充各种几何图形等。由 Graphics 对象记录针对绘制图形和文本的一系列设置。

[填空]绘图模式主要有两种，分别是正常模式和异或模式。正常模式下，后绘制的图形覆盖先绘制的图形，使得先绘制的图形被重叠的部分不再可见；异或模式下，当前绘制的颜色、先前绘制的颜色及所选定的某种颜色之间进行某种处理，使用得到的新颜色值进行绘制。

[单选、填空]Graphics2D 拥有更强大的二维图形处理能力，提供对几何形状、坐标转换、颜色管理以及文字布局等更复杂的控制。

[简答]使用 Graphics2D 类的新方法画一个图形的步骤通常是：

(1) 先在重画方法 paintComponent() 或 paint() 中，把参数对象 g 强制转换成 Graphics2D 对象。

(2) 用上述各图形类提供的静态方法 Double() 创建该图形的对象。

(3) 以图形对象为参数调用 Graphics2D 对象的 draw() 方法绘制这个图形。

第九章 Swing 组件

第一节 组合框与列表

[填空]组合框(JComboBox)是一个下拉式菜单，它有两种形式：不可编辑的和可编辑的。

[程序设计]JComboBox 常用的构造方法有以下两种：

(1) JComboBox()，创建一个没有任何可选项的默认组合框。

(2) JComboBox(Object[] items)，根据 Object 数组创建组合框，Object 数组的元素即为组合框中的可选项。

进行简短说明。“//”是注释的开始，行尾表示注释结束。

(2) 可用于多行注释，“/*”是注释的开始，“*/”表示注释结束，“/*”和“*/”之间的所有内容均是注释内容。这种注释多用来说明方法的功能、设计逻辑及基本思想等。

(3) 文档注释，以“/**”开头，以“*/”结束。一般地，在公有类定义或公有方法头的前面会使用这样的注释。

[填空]语句是 Java 程序的最小执行单位，程序的各语句间以分号“；”分隔。

[填空]Java 语言定义了许多关键字，关键字也称为保留字。

[单选、填空]在 Java 语言中，标识符是由字母、数字、下划线(_)或美元符(\$)组成的字符串，其中数字不能作为标识符的开头。标识符区分大小写，长度没有限制。除以上所列几项之外，标识符中不能含有其他符号，例如 +、=、* 及 % 等，当然也不允许插入空白。在程序中，标识符可用作变量名、方法名、接口名和类名等。

第二节 基本数据类型

[单选、填空]Java 的数据类型共分为两大类，一类是基本数据类型，另一类是复合数据类型。基本数据类型共有 8 种，分为 4 小类，分别是整型、浮点型、字符型和布尔型。整型和浮点型有时也合称为数值型。复合数据类型包括数组、类和接口。其中，数组是一个很特殊的概念，它是对象，而不是一个类，一般地把它归为复合数据类型。

[单选、填空]Java 语言提供了 4 种整型量，对应的关键字分别是：byte、short、int 和 long。下表列出了 4 种整数类型的字节大小和可表示的范围。

表 Java 整数类型的表示范围

| 整数类型 | 整数长度 | 字节数 | 表示范围 |
|-------|------|-----|---|
| byte | 8 位 | 1 | -2 ⁷ ~ 2 ⁷ - 1 (-128 ~ 127) |
| short | 16 位 | 2 | -2 ¹⁵ ~ 2 ¹⁵ - 1 (-32768 ~ 32767) |
| int | 32 位 | 4 | -2 ³¹ ~ 2 ³¹ - 1 (-2,147,483,648 ~ 2,147,483,647) |
| long | 64 位 | 8 | -2 ⁶³ ~ 2 ⁶³ - 1 (-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807) |

[单选、填空]整型量可用十进制、八进制或十六进制形式表示，以 1 ~ 9 开头的数为十进制数，以 0 开头的数为八进制数，以 0x 或 0X 开头的数为十六进制数。

[填空]Java 浮点数类型遵从标准的浮点规则。浮点数类型有两种：一种是单精度浮点数，用 float 关键字说明；另一种是双精度浮点数，用 double 关键字说明，它们都是有符号数。

[单选、填空]单个字符用 char 类型表示。一个 char 表示一个 Unicode 字符，其值用 16 位无符号整数表示，范围为 0 ~ 65535。char 类型的常量值必须用一对单引号('')括起来，分为普通字符常量和转义字符常量两种。

[单选、填空]逻辑值有两个状态，它们常被写作 on 和 off、true 和 false、yes 和 no 等。

在 Java 中,这样的一个值用 boolean(布尔)类型表示,布尔类型也称作逻辑类型。boolean 类型有两个常量值:true 和false,它们全是小写,计算机内部使用8位二进制数表示。

第三节 表达式

[单选、填空] 表达式由运算符和操作数组成,对操作数进行运算符指定的操作,并得出运算结果。

[填空] 常量操作数很简单,只有简单数据类型和String 类型才有相应的常量形式。

[单选、填空] 变量是存储数据的基本单元,它可以用作表达式中的操作数。变量在使用之前要先声明。变量声明的基本格式为:

类型 变量名 1 [= 初值 1] [, 变量名 2[= 初值 2]] ...;

其中,类型是变量所属的类型,既可以是简单数据类型,如 int 和 float 等,又可以是类类型。有时也把类类型的变量称为引用。方括号中的初值是可选的。如果没有,则表明仅是声明了一个变量,否则是在声明变量的同时,给变量赋了初值,称为对变量进行了初始化。

[填空] 具有 null 值的引用不指向任何对象。如果使用它指向的对象,则将导致一个异常。异常是程序运行时发生的一个错误。

[填空] 变量的作用域是指可访问该变量的代码范围。类中定义的成员变量的作用域是整个类。方法中定义的局部变量的作用域是从该变量的声明处开始到包含该声明的语句块结束处,块外是不可使用的。

[单选、填空、程序设计] 运算符:

(1) 算术运算符。算术运算符包括通常的加(+)、减(-)、乘(*)、除(/)、取模(%),完成整型和浮点型数据的算术运算。许多语言中的取模运算只能用于整型数,Java 对此有所扩展,它允许对浮点数进行取模操作。

(2) 关系运算符。关系运算符用来比较两个值,包括大于(>)、大于等于(>=)、小于(<)、小于等于(<=)、等于(==)和不等于(!=)6 种。关系运算符都是二元运算符,也就是每个运算符都带有两个操作数。运算的结果是一个逻辑值。Java 允许“=”和“!=”两种运算用于任何数据类型。

(3) 逻辑运算符。逻辑运算符包括逻辑与(&&)、逻辑或(||)和逻辑非(!)。前两个是二元运算符,后一个是一元运算符。Java 对逻辑与和逻辑或提供“短路”操作功能。

(4) 位运算符。位运算符用来对二进制位进行操作,包括按位取反(~)、按位与(&)、按位或(|)、异或(^)、右移(>>)、左移(<<)及无符号右移(>>>)。位运算符只能对整型和字符型数据进行操作。

(5) 其他运算符。Java 中的运算符还包括扩展赋值运算符(+ =、- =、* =、/ =、% =、& =、| =、^ =、> > =、< < =和> > > =)、条件运算符(?:)、点运算符(.)、实例运算符(instanceof)、new 运算符及数组下标运算符([])等。

(6) JToggleButton(String text, Icon icon), 创建一个既有显示文本又有图标的切换按钮。

(7) JToggleButton(String text, Icon icon, boolean selected), 创建一个既有显示文本又有图标和指定初始状态的切换按钮。

第四节 布局管理器

[单选、填空] FlowLayout 定义在 java.awt 包中,这个布局管理器对容器中组件进行布局的方式是将组件逐个地放置在容器中的一行上,一行放满后就另起一个新行。

[程序设计] FlowLayout 布局管理器有 3 种构造方法:

(1) FlowLayout(), 创建一个默认的 FlowLayout 布局管理器,居中对齐,默认的水平和垂直间距是 5 个像素。

(2) FlowLayout(int align), 创建一个新的 FlowLayout 布局管理器,对齐方式是指定的,默认的水平和垂直间距是 5 个像素。

(3) FlowLayout(int align,int hgap,int vgap), 创建一个新的 FlowLayout 布局管理器,具有指定的对齐方式以及指定的水平和垂直间距。

[单选、填空] BorderLayout 是顶层容器中内容窗格的默认布局管理器,它提供了一种较为复杂的组件布局管理方案。每个由 BorderLayout 管理的容器被划分成 5 个区域,分别代表容器的上部(North)、下部(South)、左部(West)、右部(East)和中部(Center),分别使用常量 BorderLayout.NORTH、BorderLayout.SOUTH、BorderLayout.WEST、BorderLayout.EAST 和 BorderLayout.CENTER 来表示。在容器的每个区域,可以加入一个组件。

[单选、填空] GridLayout 是一种网格式的布局管理器,它将容器空间划分成若干行乘若干列的网格,组件依次放入其中,每个组件占据一格。

[单选、填空] 卡片的顺序由组件对象本身在容器内部的顺序决定。CardLayout 定义了一组方法,这些方法允许应用程序按顺序浏览这些卡片,或者显示指定的卡片。

[单选、填空] BoxLayout 是定义在 javax.swing 包中的另一种常用布局管理器,它将容器中的组件按水平方向排成一行或按垂直方向排成一列。当组件排成一行时,每个组件可以有不同的宽度;当组件排成一列时,每个组件可以有不同的高度。

第五节 事件处理

[单选、填空] Java 中,为了便于管理,系统将事件分类,称为事件类型。系统为每个事件类型提供一个侦听程序接口,它规定了接收并处理该类事件的方法的规范。为了接收并处理某类用户事件,组件必须注册相应的事件处理程序,这种事件处理程序称为事件侦听程序(Listener,也称为侦听器),它是实现了对应侦听程序接口的一个类。要作为侦听程序对象的类必须实现相应的接口,并实现接口中规定的响应事件的方法。

[填空] 在适配器类中实现了相应接口中的全部方法,只是方法的内容为空。

第六节 绘图基础

[单选、填空] 可以使用 java.awt 包中的 Color 类来定义和管理颜色。Color 类的每个对象表示一种颜色。可以有两种方法生成颜色。一种方法是使用 Java 的 Color 类中预定

第三节 标签及按钮

[填空] 标签(JLabel)对象是最简单的 Swing 组件,通常用于显示提示性的文本信息或图标,不可被编辑。

[简答、程序填充] 标签对象的构造方法有以下 6 种形式:

- (1) `JLabel()`, 创建一个既不显示文本信息又不显示图标的空标签。
- (2) `JLabel(Icon image)`, 创建一个显示图标的标签。
- (3) `JLabel(String text)`, 创建一个显示文本信息的标签。

(4) `JLabel(Icon image, int horizontalAlignment)`, 创建一个显示图标的标签,水平对齐方式由 int 型参数 `horizontalAlignment` 指定。

(5) `JLabel(String text, int horizontalAlignment)`, 创建一个显示文本信息的标签,水平对齐方式由 int 型参数 `horizontalAlignment` 指定。

(6) `JLabel(String text, Icon icon, int horizontalAlignment)`, 创建一个同时显示文本信息和图标的标签,水平对齐方式由 int 型参数 `horizontalAlignment` 指定。

构造方法中,表示水平对齐方式的 int 型参数 `horizontalAlignment` 的取值可以为 `JLabel.LEFT`、`JLabel.RIGHT` 和 `JLabel.CENTER` 常量,分别表示左对齐、右对齐和居中对齐。

[单选、填空] 按钮(JButton)是 Java 图形用户界面的基本组件之一,经常用到的按钮有 4 种形式:`JButton`、`JToggleButton`、`JCheckBox` 和 `JRadioButton`,它们均是 `AbstractButton` 的子类或间接子类。各种按钮上都可以设置文本、设置图标、注册事件侦听程序。在 `AbstractButton` 中定义了按钮所共有的一些方法,例如 `addActionListener()`、`setEnabled()`、`setText()` 和 `setIcon()` 等。

[程序设计] `JButton` 是最简单的按钮,常用的构造方法有以下几种:

- (1) `JButton()`, 创建一个既没有显示文本又没有图标的按钮。
- (2) `JButton(Icon icon)`, 创建一个没有显示文本但有图标的按钮。
- (3) `JButton(String text)`, 创建一个有显示文本但没有图标的按钮。
- (4) `JButton(String text, Icon icon)`, 创建一个既有显示文本又有图标的按钮。

[单选]除了普通按钮 `JButton` 外,还有切换按钮(JToggleButton)、复选按钮(JCheckBox)和单选按钮(JRadioButton)。

[填空] `JToggleButton` 是具有两种状态的按钮,即选中状态和未选中状态。

[程序分析、程序设计] `JToggleButton` 的构造方法主要有以下几种格式:

- (1) `JToggleButton()`, 创建一个既没有显示文本又没有图标的切换按钮。
- (2) `JToggleButton(Icon icon)`, 创建一个没有显示文本但有图标的切换按钮。
- (3) `JToggleButton(Icon icon, boolean selected)`, 创建一个没有显示文本但有图标和指定初始状态的切换按钮。
- (4) `JToggleButton(String text)`, 创建一个有显示文本但没有图标的切换按钮。
- (5) `JToggleButton(String text, boolean selected)`, 创建一个有显示文本和指定初始状态但没有图标的切换按钮。

第三章 流程控制语句

[单选、填空] Java 是一种强类型语言,不支持变量类型间的自动任意转换,有时必须显式地进行变量类型的转换。每个数据都与特定的类型相关,允许整型、浮点型、字符串型数据进行混合运算。运算时,不同类型的数据先转换为同一种类型,然后再进行运算。转换的一般原则是位数少的类型转换为位数多的类型,这称作自动类型转换。这样做的目的是保证转换时不丢失有用信息。

第三章 流程控制语句

第一节 Java 程序的结构

[单选、填空] 一个源文件中如果有多个类,则最多只能有一个是 `public` 类型的类,且该源文件的名字即为这个公有类的名字,且大小写也要一致。其他非 `public` 的类的个数不限。

[简答]一个 Java 程序的结构包含以下内容:

- (1) `package` 语句,包语句,每个文件最多只有一个,且必须放在文件开始的地方。
- (2) `import` 语句,引入语句,可以没有,也可以有多个,如果有 `import` 语句的话,必须放在所有类定义的前面。
- (3) 具有 `public` 权限的类定义,每个文件中最多有一个。
- (4) 类定义,每个文件中包含的非 `public` 权限的类定义的个数没有限制。
- (5) 接口定义,每个文件中包含的接口定义个数没有限制。

[填空] 包是类的容器,包的设计人员利用包来划分名字空间,以避免类名冲突。

[程序填充] 包语句的格式为:

`package pkg1[. pkg2[. pkg3...]];`

程序中如果有 `package` 语句,该语句一定是源文件中的第一条非注释语句,它的前面只能有注释或空行。另外,一个文件中最多只能有一条 `package` 语句。

[程序填充、程序分析] 引入语句的格式如下:

`import pkg1[. pkg2[. pkg3...]].(类名 | *)`;

其中,格式语句中如果指明具体的类名,则表示引入的是这个具体的类;要引入包中的所有类时,可以使用通配符“*”。

第二节 流程控制

[单选、填空] 在 Java 程序中,表达式可以当作一个值赋给某个变量,这样的语句称为赋值语句。有的表达式也可单独当作语句,这样的语句称为表达式语句。

[简答] 语句与表达式的相同点和不同点:

- (1) 有的表达式可以当作语句,但并不是所有的语句都是表达式。
- (2) 每个表达式都会得到一个值,即表达式的计算结果。

虽然语句也会有一个值,但这个值并不是语句的计算结果,而是执行结果。

[单选、填空] 分支语句根据一定的条件, 动态决定程序的流程方向, 从程序的多个分支中选择一个或几个来执行。分支语句有if语句和switch语句两种。

[单选] if语句是单重选择, 最多只有两个分支。

[程序设计] if语句的基本格式是:

```
if(条件表达式)
```

```
    语句1;
```

```
[ else
```

```
    语句2;
```

```
]
```

if关键字之后的条件表达式必须得到一个逻辑值, 不能像其他语言那样以数值来代替。语句1和语句2还可以是if语句, 这样的if语句称为嵌套的if语句。使用嵌套的if语句可以实现多重选择, 也就是可以有多个分支。

[程序设计] switch语句的语法格式如下:

```
switch(表达式) {
    case c1:
        语句组1;
        break; //break语句可选
    case c2:
        语句组2;
        break; //break语句可选
    .....
    case ck:
        语句组k;
        break; //break语句可选
    default:
        语句组;
        break; //break语句可选
}
```

[单选、填空] 循环语句控制程序流多次执行一段代码。Java语言提供3种循环语句, 分别是for语句、while语句和do语句。

[程序填充、程序设计] for语句:

for语句的语法格式是:

```
for(初始语句; 条件表达式; 迭代语句)
```

```
    循环体语句;
```

初始语句和迭代语句中可以含有多个语句, 各语句间以逗号分隔。for语句括号内的3个部分都是可选的, 条件表达式为空时, 默认规定为恒真。

被包含在浏览器窗口中。创建对话框时使用JDialog。JWindow是一个不带有标题行和控制按钮的窗口, 通常很少使用。

[程序填充、程序分析] JFrame类常用的构造方法有以下几种:

(1) JFrame(), 构造一个初始时不可见、无标题的新框架窗体。

(2) JFrame(String title), 创建一个初始时不可见、具有指定标题的新框架窗体。

[程序填充、程序设计] JFrame类中定义了一些相关方法, 另外也从祖先类中继承了一些方法。常用的方法有以下几种:

(1) void setBounds(int x, int y, int width, int height), 移动并调整框架大小。左上角位置的横纵坐标分别由x和y指定, 框架的宽高分别由width和height指定。

(2) void setSize(int width, int height), 设置框架的大小, 宽度是width, 高度是height。

(3) void setBackground(Color bg), 使用颜色bg设置框架的背景色。

(4) void setVisible(boolean aFlag), 设置框架可见或不可见。

(5) void pack(), 调整框架的大小, 以适合其子组件的首选大小和布局。

(6) void setTitle(String title), 设置框架的标题为字符串title。

(7) Container getContentPane(), 返回此框架窗体的内容窗格对象。

(8) void setLayout(LayoutManager manager), 设置布局管理器。

[单选、填空] 4个顶层容器中的每一个都有一个内容窗格。除菜单之外, 顶层容器中的组件都放在这个内容窗格中。有两种方法可以将组件放入内容窗格中, 一种方法是通过顶层容器的getContentPane()方法获得其默认的内容窗格。getContentPane()方法的返回类型为java.awt.Container, 它仍然是一个容器。然后将组件添加到内容窗格中。

另一种方法是创建一个新的内容窗格, 以取代顶层容器默认的内容窗格。通常的做法是创建一个 JPanel 的实例, 它是 java.awt.Container 的子类。然后将组件添加到 JPanel 实例中, 再通过顶层容器的setContentPane()方法将 JPanel 实例设置为新的内容窗格。

[单选、填空] 普通面板(JPanel)和滚动面板(JScrollPane)都是用途广泛的容器。与顶层容器不同的是, 面板不能独立存在, 必须被添加到其他容器内部。面板可以嵌套, 由此可以设计出复杂的图形用户界面。

[填空] 当容器中组件过多而不能在显示区域内全部显示时, 可以让容器带滚动条, 从而显示出全部的组件, 使用滚动面板可以实现这个功能。

[程序填充] JPanel类常用的构造方法有以下几种:

(1) JPanel(): 创建具有FlowLayout布局的新面板。

(2) JPanel(LayoutManager layout): 创建具有指定布局管理器的新面板。

使用public Component add(Component comp)方法可以将指定组件追加到面板中。

[单选、填空] JScrollPane是带有滚动条的面板, 它是Container类的子类。但是只能添加一个组件。所以当有多个组件需要添加时, 一般是先将多个组件添加到 JPanel 中, 然后再将这个 JPanel 添加到 JScrollPane 中。

(3) boolean canRead():文件对象是否可读。
 (4) boolean isFile():文件对象是否是文件。
 (5) boolean isDirectory():文件对象是否是目录。
 (6) boolean isAbsolute():文件对象是否是绝对路径。

[程序填充、程序设计]常用文件信息和方法:

- (1) long lastModified():获取文件最后修改时间。
- (2) long length():获取文件长度。
- (3) boolean delete():删除文件对象指向的文件,成功则返回 true,否则返回 false。

[程序填充、程序设计]目录工具:

- (1) boolean mkdir():创建新目录。
- (2) boolean mkdirs():创建新目录。
- (3) String[] list():列出符合模式的文件名。

[程序填充、程序设计]创建一个随机访问文件有以下两种方法供选择:

(1) 使用文件名。

```
myRAFile = new RandomAccessFile(String name, String mode);
```

(2) 使用文件对象。

```
myRAFile = new RandomAccessFile(File file, String mode);
```

第八章 图形界面设计

第一节 AWT 与 Swing

[单选、填空]设计图形用户界面时一般有 3 个步骤,分别是选取组件、设计布局及响应事件。

[单选、填空]Swing 组件与 AWT 组件最大的不同是 Swing 组件在实现时不包含任何本地代码,因此 Swing 组件可以不受硬件平台的限制,而具有更多的功能。基于 AWT 的界面可能会因运行平台的不同略有差异,而基于 Swing 的界面在任何平台上的显示效果都是一致的。不包含本地代码的 Swing 组件被称为“轻量级”组件,而包含本地代码的 AWT 组件被称为“重量级”组件。当“重量级”组件与“轻量级”组件一同使用时,如果组件区域有重叠,则“重量级”组件总是显示在上面。在 Java 2 平台上推荐使用 Swing 组件。

[填空]Swing 组件比 AWT 组件拥有更多的功能。

第二节 容器

[单选、填空]组件可以分为容器组件和非容器组件。所谓容器组件是指可以包含其他组件的组件,又分为顶层容器和一般用途容器。而非容器组件则必须要包含在容器中。

[单选、填空]Swing 中提供了 4 种顶层容器,分别为 JFrame、JApplet、JDialog 和 JWindow。JFrame 是一个带有标题栏和控制按钮(最小化、恢复/最大化、关闭)的独立窗口,有时称为框架,创建应用程序时需要使用 JFrame。创建小应用程序时使用 JApplet,它

[程序填充、程序设计]while 循环的语法格式是:

```
while(条件表达式)
```

循环体语句;

和 if 语句一样,while 语句中的条件表达式亦不能用数值来代替。

[程序填充、程序设计]do 语句:

do 语句与 while 语句很相似。它把 while 语句中的条件表达式移到循环体之后。do 语句的语法格式是:

```
do
```

语句;

```
while(条件表达式);
```

[单选、填空]标号可以放在任意语句之前,通常与 for、while 或 do 语句配合使用。

[单选、填空]break 语句可用于 3 类语句中,第一类是在 switch 语句中,第二类是在 for、while 及 do 等循环语句中,第三类是在语句块中。在 switch 语句及循环语句中,break 的语义是跳过本块中余下的所有语句,转到块尾,执行其后的语句。

[单选、填空]在循环语句中,continue 语句可以立即结束当次循环,开始执行下一次循环,当然执行前要先判断循环条件是否满足。

第三节 简单的输入/输出

[单选、填空]Scanner 类的构造方法接收一个参数,这个参数代表了输入源。System.in 对象代表标准输入流,默认是指键盘。Scanner 对象用空白(空格、水平制表符及回车换行符)作为输入的分隔元素。这些空白称为分隔符。也可以指定用其他的符号作为分隔符。

[单选、填空]Scanner 类的 next() 方法读入下一个输入对象,将它作为字符串返回。如果输入的是一串用空白分开的多个字,则每次调用 next() 时都会得到下一个字。nextLine() 方法读入当前行的所有输入,直到行尾,然后作为字符串返回。

[填空]NumberFormat 类提供对数值进行格式化操作的一般功能。不能使用 new 运算符实例化一个 NumberFormat 对象,只能直接使用类名调用一个特殊的静态方法来得到一个对象。

[单选、填空]和 NumberFormat 类不一样,DecimalFormat 类按惯例使用 new 运算符来实例化对象。它的构造方法要带一个 String 类型的参数,这个参数表示格式化处理模式。然后可以使用 format() 对一个具体的值进行格式化。之后,还可以调用 applyPattern() 方法来改变对象要使用的模式。

第四节 处理异常

[单选、填空]Java 语言把程序运行中可能遇到的错误分为两类,一类是非致命性的,通过某种修正后程序还能继续执行。这类错误称作异常(Exception)。如打开一个文件时,发现文件不存在。又比如除零溢出、数组越界等。这一类的错误可以借助于程序员的

处理来恢复。另一类是致命性的,即程序遇到了非常严重的不正常状态,不能简单地恢复执行,这就是错误。比如程序运行过程中内存耗尽。

[简答] 异常分为以下 3 种:

- (1) 受检异常,必须被处理。
- (2) 运行时异常,不需要处理。
- (3) 错误,不需要处理。

[单选、填空] 受检异常是程序执行期间发生的严重事件的后果。

[单选、填空] 运行时异常通常是程序中逻辑错误的结果。

[单选、填空] 运行时异常和错误称为不检异常。

[单选、填空] 当发生异常时,程序通常会中断执行,并输出一条信息。对所发生的异常进行的处理就是异常处理。

[程序分析] 几个常用到的公共异常:

(1) `ArithmaticException`。整数除法中,如果除数为 0,则发生该类异常。

(2) `NullPointerException`。如果一个对象还没有实例化,那么访问该对象或调用它的方法将导致 `NullPointerException` 异常。

(3) `NegativeArraySizeException`。按常规,数组的元素个数应是一个大于等于 0 的整数。创建数组时,如果元素个数是负数,则会引发 `NegativeArraySizeException` 异常。

(4) `ArrayIndexOutOfBoundsException`。Java 把数组看作是对象,并用 `length` 变量记录数组的大小。访问数组元素时,运行时环境根据 `length` 值检查下标的大小。如果数组下标越界,则将导致 `ArrayIndexOutOfBoundsException` 异常。

[单选、填空] 程序员处理异常有两种方法。一种是使用 `try` 块和 `catch` 块,捕获到所发生的异常类,并进行相应的处理。当然,`catch` 块可以为空,表示对发生的异常不进行处理。另一种方法是,程序员不在当前方法内处理异常,而是把异常抛出到调用方法中。当不能使用合理的方式来解决不正常或意外事件的情形下,才抛出异常。

第四章 面向对象程序设计

第一节 类和对象

[填空] 类的定义也称为类的声明。类中含有两部分元素,分别是数据成员变量和成员方法。

[程序设计] 类定义的一般格式如下:

修饰符 `class` 类名 [`extends` 父类名] {

 修饰符 类型 成员变量 1;

 修饰符 类型 成员变量 2;

....

 修饰符 类型 成员方法 1(参数列表) {

第七章 输入和输出流

(3) `void write(char[] cbuf)`。

(4) `void write(char[] cbuf, int off, int len)`。

(5) `void write(int c)`。

(6) `void write(String str)`。

(7) `void write(String str, int off, int len)`。

[程序分析、程序设计] 缓冲区读者和缓冲区写者:

像其他 I/O 操作一样,如果格式转换以较大数据块为单位进行,那么会提高效率。为此,java.io 中提供了缓冲流 `BufferedReader` 和 `BufferedWriter`。其构造方法与 `BufferedInputStream` 和 `BufferedOutputStream` 类似。

另外,除了 `read()` 和 `write()` 方法外,还提供了整行字符的处理方法。

(1) `public String readLine()`: `BufferedReader` 的方法,从输入流中读取一行字符,行结束标志为 '`\n`'、'`\r`' 或两者一起。

(2) `public void newLine()`: `BufferedWriter` 的方法,向输出流中写入一个行结束标志。

把 `BufferedReader` 或 `BufferedWriter` 正确连接到 `InputStreamReader` 或 `OutputStreamWriter` 的末尾是一个很好的方法。但是要在 `BufferedWriter` 中使用 `flush()` 方法,以强制清空缓冲区中的剩余内容,防止遗漏。

第四节 文件的处理

[程序设计] 创建一个新的 `File` 对象可以使用的构造方法:

第 1 种方法为:

`File myFile;`

`myFile = new File("mymotd");`

第 2 种方法为:

`myFile = new File("/", "mymotd");`

第 3 种方法为:

`File myDir = new File("/");`

`myFile = new File(myDir, "mymotd");`

根据文件对象的具体情况,选择使用何种构造方法。

[程序填充、程序设计] 与文件名相关的方法:

(1) `String getName()`: 获取文件名。

(2) `String getPath()`: 获取文件路径。

(3) `String getAbsolutePath()`: 获取文件绝对路径。

(4) `String getParent()`: 获取文件父目录名称。

(5) `boolean renameTo(File newName)`: 更改文件名,成功返回 `true`,否则返回 `false`。

[程序填充、程序设计] 文件测定方法:

(1) `boolean exists()`: 文件对象是否存在。

(2) `boolean canWrite()`: 文件对象是否可写。

[单选、填空]能够记录自己的状态以便将来得到复原的能力,称为对象的持久性(Persistence)。称一个对象是可持久的,意味着可以把这个对象存入磁盘、磁带,或传入另一台计算机保存在它的内存或磁盘中。也就是说,把对象存为某种永久存储类型。

[单选、填空]对象通过数值来描述自己的状态,记录对象也就是记录下这些数值。把对象转换为字节序列的过程称为对象的序列化,把字节序列恢复为对象的过程称为对象的反序列化。序列化的主要任务是写出对象实例变量的数值。序列化是一种用来处理对象流的机制,所谓对象流也就是将对象的内容进行流化。序列化是为了解决在对对象流进行读写操作时所引发的问题。

[单选、填空]当数据变量是一个对象时,该对象的数据成员也可以被持久化。对象的数据结构或结构树,包括其子对象树在内,构成了这个对象的结构表。

如果一个对象结构表中包含了一个对不可持久化对象的引用,而这个引用已用关键字 transient 加以标记,则这个对象仍可以被持久化。

第三节 基本字符流

[单选、填空]Java 通过读者和写者,实现了对不同平台之间数据流中的数据进行转换。同其他程序设计语言使用 ASCII 字符集不同,Java 使用 Unicode 字符集来表示字符串和字符。ASCII 字符集以一个字节(8bit)表示一个字符,可以认为一个字符就是一个字节(byte)。但 Java 使用的 Unicode 是一种大字符集,用两个字节(16bit)来表示一个字符,这时字节与字符就不再相同。为实现与其他程序语言及不同平台的交互,Java 提供一种新的数据流处理方案,称作读者(Reader)和写者(Writer)。像数据流一样,在 java.io 包中有许多不同类对其进行支持,其中最重要的是 InputStreamReader 和 OutputStreamWriter 类。这两个类是字节流和读者、写者的接口,用来在字节流和字符流之间作为中介。使用这两个类进行字符处理时,在构造方法中应指定一定的平台规范,以便把以字节方式表示的流转换为特定平台上的字符表示。

[简答、程序填充]读者提供的方法包括以下几种:

- (1) void close()。
- (2) void mark(int readAheadLimit)。
- (3) boolean markSupported()。
- (4) int read()。
- (5) int read(char[] cbuf)。
- (6) int read(char[] cbuf, int off, int len)。
- (7) boolean ready()。
- (8) void reset()。
- (9) long skip(long n)。

[简答、程序填充]写者提供的方法包括以下几种:

- (1) void close()。
- (2) void flush()。

```

    方法体
}
修饰符 类型 成员方法 2(参数列表) {
    方法体
}
.....
}
```

其中, class 是关键字,表明其后定义的是一个类。含有 class 的这一行称为类头,后面大括号括住的部分称为类体。class 前的修饰符可以有多个,用来限定所定义的类的使用方式。类名是用户为该类所起的名字,它必须是一个合法的标识符,并尽量遵从命名约定。extends 是关键字。如果所定义的类是从某一个父类派生而来,那么,父类的名字要写在 extends 之后。

[单选、填空]类定义中的修饰符是访问权限修饰符,包括 public、private 和 protected,也可以不写,表示是默认修饰符。它们既可以用来修饰类,又可以修饰类中的成员,修饰符决定所修饰成员在程序运行时被访问的方式。访问权限关键字与访问能力之间的关系见表4-1。

表 4-1 访问权限修饰符

| 类型 | 无修饰符 | private | protected | public |
|----------|------|---------|-----------|--------|
| 同一类 | 是 | 是 | 是 | 是 |
| 同一包中的子类 | 是 | 否 | 是 | 是 |
| 同一包中的非子类 | 是 | 否 | 是 | 是 |
| 不同包中的子类 | 否 | 否 | 是 | 是 |
| 不同包中的非子类 | 否 | 否 | 否 | 是 |

[填空]构造方法的名字与类名相同,没有返回值,在创建对象实例时通过 new 运算符自动调用。同时为了便于创建实例,一个类可以有多个具有不同参数列表的构造方法,即构造方法可以重载。

[单选]构造方法不能说明为 native、abstract、synchronized 或 final 类型,通常说明为 public 类型的。

[单选、填空]每个类都必须至少有一个构造方法。如果程序员没有为类定义构造方法,系统就会自动为该类生成一个默认的构造方法。默认构造方法的参数列表及方法体均为空,所生成的对象的各属性值也为零或空。如果类定义中已经含有一个或多个构造方法,则系统不会再自动生成默认的构造方法了。

[填空]this 除了可以用在构造方法中之外,还可以用来指明要操作的对象自身。

[单选、填空]创建对象实例的格式如下:

变量名 = new 类名(参数列表);

也可以与变量声明合在一起使用。

类名 变量名 = new 类名(参数列表);

当通过 new 为一个对象分配内存时,如果构造方法中没有为成员变量提供初值,则 Java 进行自动初始化。对于数值变量,赋初值 0;对于布尔变量,赋初值 false;对于引用,即对象类型的任何变量,使用一个特殊的值 null 作为初值。

[填空] 定义类时,可以对成员变量进行显式的初始化,即在声明这些变量的同时,直接给出它们的初值。

第二节 定义方法

[单选、填空] 对对象的操作体现在成员方法上。说明为 private 的成员变量,在类外,不能通过点操作符直接访问,必须通过成员方法才能访问。

[简答] 关于方法定义的说明:

(1) 方法名必须是一个合法的标识符。

(2) 返回类型是方法返回值的类型。如果方法不返回任何值,则应该声明为 void。Java 对待返回值的要求很严格,方法返回值必须与所声明的类型相匹配。如果方法声明有返回值,比如说是 int,那么方法从任何一个分支返回时都必须返回一个整数值。

(3) 修饰符段可以含有几个不同的修饰符,其中限定访问权限的修饰符包括 public、protected 和 private。

(4) 参数列表是传送给方法的参数表。表中各元素间以逗号分隔,每个元素由一个类型和一个标识符表示的参数组成。

(5) 块表示方法体,是要实际执行的代码段,是由一对大括号括起来的语句序列。方法体中一般使用 return 语句表示方法的结束。如果方法的返回类型不是 void,则需要在 return 语句中指明方法的返回值。

[填空] 调用方法时,通常会给方法传递一些值。传给方法的值称为实参,方法的参数列表中列出的值称为形参。

[程序分析、程序设计] Java“按值”传送实参:

(1) 如果形参是基本数据类型的,则调用方法时,将实参的“值”复制给形参。返回时,形参的值并不会带回给实参,即在方法内对形参的任何修改,都不会影响实参的值。

(2) 如果形参是引用,则调用方法时传递给形参的是一个地址,即实参指向的对象的首地址。方法返回时,这个地址也不会被改变,但地址内保存的内容是可以改变的。因此,当从方法中退出时,所修改的对象内容可以保留下来。

[填空] 允许多个方法使用同一个方法名,这就是方法名的重载。

[单选、填空] 一般地,方法名称加上方法的参数列表(包括方法中参数的个数、顺序和类型)称为方法签名。方法重载时,方法签名一定不能相同。

[简答] 重载方法有两条规则:

(1) 调用语句的实参列表必须能够判断要调用的是哪个方法。实参的类型可能要进行正常的扩展提升(如浮点数变为双精度数),但在有些情况下会引起混淆。

[简答] 输出数据流中提供的主要数据操作方法:

(1) void write(int i): 将字节 i 写入到数据流中,它只输出所读入参数的最低 8 位。该方法是抽象方法,需要在其输出流子类中加以实现,然后才能使用。

(2) void write(byte b[]): 将数组 b[] 中的全部 b.length 个字节写入数据流。

(3) void write(byte b[], int off, int len): 将数组 b[] 中从下标 off 开始的 len 个字节写入数据流。元素 b[off] 是此操作写入的第一个字节,b[off + len - 1] 是此操作写入的最后一个字节。

以上这些方法用于向输出数据流中写数据。在实际应用中,和操作输入数据流一样,通常以系统允许的最大数据块长度为单位进行写操作。

第二节 基本字节数据流类

[程序分析、程序填充] 文件数据流:

文件数据流包括 FileInputStream 和 FileOutputStream,这两个类用来进行文件的 I/O 处理,其数据源或数据终点都应当是文件。通过所提供的方法可以对本机上的文件进行操作,但是不支持方法 mark() 和 reset()。在构造文件数据流时,可以直接给出文件名。

[单选、填空] 一个过滤器数据流在创建时与一个已经存在的数据流相连,这样在从这样的数据流中读取数据时,它提供的是对一个原始输入数据流的内容进行了特定处理的数据。

[单选、填空] 缓冲区数据流有 BufferedInputStream 和 BufferedOutputStream,它们是在数据流上增加了一个缓冲区,都属于过滤器数据流。当读写数据时,数据以块为单位先进入缓冲区(块的大小可以进行设置),其后的读写操作则作用于缓冲区。采用这个办法可以降低不同硬件设备之间速度的差异,提高 I/O 操作的效率。与此同时,这两个流还提供了对 mark()、reset() 和 skip() 等方法的支持。

[单选、填空] DataInputStream 和 DataOutputStream 是这样的两个过滤器数据流,它们允许通过数据流来读写 Java 基本类型,包括布尔型(boolean)、浮点型(float)等。假设 is 和 os 分别是前面已经建立好的输入/输出数据流对象,则数据流的创建方式如下:

```
DataInputStream dis = new DataInputStream(is);
```

```
DataOutputStream dos = new DataOutputStream(os);
```

在这两个类中之所以能够对这些基本类型进行操作,是因为它们提供了一组特定的方法来操作不同的基本类型。

[单选、填空] DataInputStream 的方法与 DataOutputStream 的方法都是成对出现的。如果查询 API 文档,就会发现在这两个数据流中也都定义了对字符串进行读写的方法,但是,由于字符编码的原因,应该避免使用这些方法。

[单选、填空] Java 中的数据流不仅能对基本数据类型的数据进行操作,而且也提供了把对象写入文件数据流或从文件数据流中读出的功能,这一功能是通过 java.io 包中的 ObjectInputStream 和 ObjectOutputStream 两个类实现的。能够输入/输出对象的流称为对象流。

定义中继承任何行为。在实现该接口的类的任何对象中，都能够调用这个接口中定义的方法。一个类可以同时实现多个接口。

要实现接口，可以在类的声明中用关键字 `implements` 来表示。接口中的所有抽象方法必须在类或子类中实现。

[单选、填空] Java 程序中，可以在 `implements` 后面声明多个接口名，也就是一个类可以实现多个接口。接口实际上就是一个特殊的抽象类，同时实现多个接口也就意味着具有多重继承的能力。由于在接口中的方法都是抽象方法，并不包含任何的具体代码，对这些抽象方法的实现都在具体的类中完成，因此，即使不同的接口中有同名的方法，类的实例也不会混淆。这正是 Java 取消了显式的多重继承机制，但还保留了多重继承的能力之所在。

第七章 输入和输出流

第一节 数据流的基本概念

[单选、填空] 在 Java 中，把这些不同类型的输入、输出源抽象为流（Stream），其中输入或输出的数据称为数据流（Data Stream），用统一的接口来表示。

[单选、填空] 数据流是指一组有顺序的、有起点和终点的字节集合。程序从键盘接收数据或向文件中写数据，都可以使用数据流来完成。

[单选、填空] 流被组织成不同的层次。数据流分为输入数据流和输出数据流。输入数据流只能读不能写，而输出数据流只能写不能读。从数据流中读取数据时，必须有一个数据源与该数据流相连。

[填空] 输入数据流是指只能读不能写的数据流，用于向计算机内输入信息使用。

[简答] 输入数据流中提供的主要数据操作方法：

(1) `int read()`：从输入流中读取一个字节的二进制数据。

(2) `int read(byte[] b)`：将多个字节读到数组中，填满整个数组。

(3) `int read(byte[] b, int off, int len)`：从输入流中读取长度为 `len` 的数据，从数组 `b` 中下标为 `off` 的位置开始放置读入的数据，读毕返回读取的字节数。

这 3 个方法提供了访问数据流中数据的方法，所读取的数据都默认为字节类型。

[简答] 在支持回推操作的数据流中经常用到如下几个方法：

(1) `boolean markSupported()`：用于测试数据流是否支持回推操作，当一个数据流支持 `mark()` 和 `reset()` 方法时返回 `true`，否则返回 `false`。

(2) `void mark(int markarea)`：用于标记数据流的当前位置，并划出一个缓冲区，其大小至少为指定参数的大小。

(3) `void reset()`：将输入流重新定位到对此流最后调用 `mark` 方法时的位置。

[填空] 输出数据流是指只能写不能读的流，用于从计算机中输出数据。

(2) 方法的返回类型可以相同也可以不同。两个同名方法仅有返回类型不同，而参数列表完全相同，这是不够的，因为在方法执行前不知道能得到什么类型的返回值，因此也就不能确定要调用哪个方法。重载方法的参数列表必须不同。

第三节 静态成员

[单选、填空] 在类的定义中还可以定义一种特殊的成员，用 `static` 修饰，称为静态成员或类成员，包括静态变量和静态方法。静态成员是不依赖于特定对象的内容。

[填空] 将一个变量定义为静态变量的方法就是将这个变量标记上关键字 `static`。

[单选、填空] 与静态变量类似，如果需要在尚未创建一个对象实例的时候就去引用方法的程序代码，那么标记上关键字 `static` 即可实现。这样的方法称为静态方法，或称类方法。与之相对的，非静态方法有时也称为实例方法。静态方法不依赖于特定对象的行为。

[简答] 使用静态方法时，有两个特别的限制必须注意：

(1) 由于静态方法可以在没有定义它所从属的类的对象的情况下加以调用，故不存在 `this` 值。因此，一个静态方法只能使用其内部定义的参数或静态变量，如果想使用非静态变量将引起编译错误。

(2) 静态方法不能被重写。也就是说，在这个类的后代类中，不能有相同名称、相同参数列表的方法。

第四节 包装类

[单选、填空] 对于 Java 中的每种基本数据类型，Java 类库中都有一个对应的包装类。所有的包装类都定义在 `java.lang` 包中。表 4-2 列出了每种基本数据类型及其对应的包装类。

表 4-2 java.lang 包中的包装类

| 基本数据类型 | 包装类 | 基本数据类型 | 包装类 |
|--------------------|----------------------|----------------------|------------------------|
| <code>byte</code> | <code>Byte</code> | <code>double</code> | <code>Double</code> |
| <code>short</code> | <code>Short</code> | <code>char</code> | <code>Character</code> |
| <code>int</code> | <code>Integer</code> | <code>boolean</code> | <code>Boolean</code> |
| <code>long</code> | <code>Long</code> | <code>void</code> | <code>Void</code> |
| <code>float</code> | <code>Float</code> | | |

注意，对应于 `void` 类型的包装类是 `Void`。但和其他的包装类不一样的是，`Void` 类不能被实例化，只表示 `void` 引用的概念。

[填空] 自动将基本数据类型转换为对应的包装类的过程称为自动装箱。逆向的转换称为拆箱，需要时也是自动完成的。自动装箱与自动拆箱仅能用在基本数据类型与对应的包装类之间。其他的情况，如将基本数据类型赋给对象引用变量，或是相反的过程，都会导致编译错误。

第五章 数组和字符串

第一节 数组

[填空]一个数组是相同数据类型的元素按一定顺序排列的集合。使用数组可以将同一类型的数据存储在连续的内存位置。数组中各元素的类型相同,通过下标来访问数组中的元素,下标从0开始。

[程序设计]一维数组的定义格式为:

类型 数组名[];

其中,类型是数组元素的类型。数组名为合法的标识符,[]指明定义的是一个数组类型变量。

[填空]初始化的过程就是数组的创建过程。

[单选、填空]数组的初始化分为静态初始化和动态初始化两种。所谓静态初始化就是在定义数组的同时给数组元素赋初值。静态初始化使用一对大括号将初值括起来,每个元素对应一个引用。

[单选、填空]在 Java 中,数组下标从0开始,数组中的元素个数 length 是数组类中唯一的数据成员变量。使用 new 创建数组时系统自动给 length 赋值。数组一旦创建完毕,其大小就固定下来。程序运行时可以使用 length 进行数组边界检查。如果发生越界访问,则抛出一个异常。

[程序分析、程序填充]二维数组的定义格式:

类型 数组名[][];

也可以采用另外两种定义方式。

类型[][] 数组名;

类型[] 数组名[];

与一维数组一样,二维数组定义时对数组元素没有分配内存空间,同样要进行初始化后,才可以访问每个元素。

[填空]与一维数组一样,多维数组的初始化也分为静态和动态两种。静态初始化时,在定义数组的同时为数组元素赋初值。

[单选、填空]对二维数组进行动态初始化时,有两种分配内存空间的方法:直接分配与按维分配。直接分配就是直接为每一维分配空间,声明数组时,给出各维的大小。按维分配是从最高维起(而且必须从最高维开始),分别为每一维分配内存,创建二维数组的一般格式为:

```
类型 数组名[][] = new 类型[数组第一维大小][];
```

```
数组名[0] = new 类型[数组第二维大小];
```

```
数组名[1] = new 类型[数组第二维大小];
```

```
.....
```

第六章 继承与多态

[单选、填空]调用稍后可能被覆盖的方法的这种处理方式,称为动态绑定或后绑定。动态绑定一定要到运行时才能确定要执行的方法代码。在编译过程中能确定调用方法的处理方式,称为静态绑定或前绑定。

第三节 终极类与抽象类

[填空]Java 中有一个重要的关键字 final,它表示终极,既可以修饰一个类,也可以修饰类中的成员变量或成员方法。

[填空]被标记为 final 的类将不能被继承,这样的类称为终极类或终态类,其声明的格式为:

```
final class 终极类名 {  
    类体  
}
```

[单选、填空]成员方法也可以被标记为 final,从而成为终极方法或终态方法。被标记为 final 的方法将不能被覆盖,从而可以确保被调用的方法是最原始的方法,而不是已被更改的子类中的方法。另外,把方法标记为 final 有时也被用于优化,从而提高编译运行效率。终极方法的定义格式为:

```
final 返回值类型 终极方法名([参数列表]) {  
    方法体  
}
```

[填空]一个变量被标记为 final,称为终极变量或终态变量。实际上它会成为一个常量。企图改变终极变量的值将引起编译错误。

[填空]在 Java 中可以通过关键字 abstract 把一个类定义为抽象类,每一个未被定义具体实现的方法也应标记为 abstract,这样的方法称为抽象方法。

[程序设计]抽象类和抽象方法的定义格式为:

```
public abstract class 抽象类名 {  
    //抽象类
```

类体

```
}
```

public abstract 返回值类型 抽象方法名([参数列表]); //抽象方法

第四节 接口

[填空、程序设计]接口的定义格式为:

```
[接口修饰符] interface 接口名 [ extends 父接口列表 ] {
```

```
.....//方法原型或静态常量
```

接口与一般类一样,本身也具有数据成员变量与方法,但数据成员变量一定要赋初值,且此值不能再更改,而方法必须是“抽象方法”。

[单选、填空]接口的实现与类的继承是相似的,不过,实现接口的类不能从该接口的

最高点。所有其他的类都是从 Object 类派生而来的。

[简答] Object 类包含了所有 Java 类的公共属性,其构造方法是 Object(),类中主要的方法如下:

- (1) public final Class getClass(): 获取当前对象所属的类信息,返回 Class 对象。
- (2) public String toString(): 按字符串对象返回当前对象本身的有关信息。
- (3) public Boolean equals(Object obj): 比较两个对象是否是同一个对象,是则返回 true。

[填空]关于对象相等的判别,在 Java 中有两种方式。一种是使用 == 运算符,另一种是使用 equals() 方法。

[填空] Java 是完全的面向对象语言,具有完全的 OOP 能力。在类的继承机制中,它抛弃了多重继承功能,仅实现了单重继承机制。

[填空] 多重继承是指从多个类共同派生一个子类,即一个类可以有多个父类。

[填空] 和大多数面向对象的语言一样,Java 允许使用对象的父类型的一个变量指向该对象,比如对于前面定义的 Employee 类和 Manager 类,可以将子类的对象赋给父类的变量:

```
Employee e = new Manager(); // 子类 Manager 的实例赋给父类变量 e
```

这称为对象转型(Casting)。

[简答] 一般地,进行对象引用转型时的规则:

(1) 沿类层次向“上”转型总是合法的,例如,把 Manager 引用转型为 Employee 引用。这种方式下不需要转型运算符,只用简单的赋值语句就可以完成。

(2) 对于向“下”转型,只能是祖先类转型到后代类,其他类之间是不允许的。

第二节 方法覆盖与多态

[单选、填空] 在面向对象语言程序设计中,方法覆盖是经常用到的概念。通过方法覆盖,可以达到语言多态性的目的。当子类中要做的事情(某个方法)与父类中不完全相同时,就要重写父类中的相关方法。

[单选、填空] 如果方法名相同,而参数列表不同,则是对方法的重载。调用重载方法时,编译器将根据参数的个数和类型,选择对应的方法执行。重载的方法属于同一个类,覆盖的方法分属于父类、子类中。

[简答] 应用覆盖时必须注意的两条重要规则:

- (1) 覆盖方法的允许访问范围不能小于原方法。
- (2) 覆盖方法所抛出的异常不能比原方法更多。

以上两条规则均源于多态性和 Java 所具有的“类型安全性”的要求。

[填空] 如果在子类构造方法的定义中调用了父类的构造方法,则调用语句必须出现在子类构造方法的第一行。

[单选、填空] 在 Java 中,多态是一个重要概念。有了多态,能够允许同一条方法调用指令在不同的上下文中做不同的事情。

数组名[数组第一维大小 - 1] = new 类型[数组第二维大小];

如果创建数组时第二维大小是一样的,则创建的是一个矩阵数组。

[单选、填空] 数组下标也称为索引,它们都可以是整型常数和表达式,都是从 0 开始。第一维也称为行,第二维也称为列。

第二节 字符串类型

[单选、填空] 字符串是由有限个字符组成的序列,Java 中的字符串是一个对象,而不是一个以 ‘\0’ 结尾的字符数组。Java 的标准包 java.lang 中封装了 String 类和 StringBuffer 类,可以方便地处理字符串。其中 String 类用于处理不变字符串, StringBuffer 类用于处理可变字符串。

[填空] 字符串是内存中连续排列的 0 个或多个字符。不变字符串是指字符串一旦创建,其内容就不能改变。

[填空] Java 程序中的字符串分为常量和变量两种,其中字符串常量是用双引号括起来的一串字符。系统为程序中出现的字符串常量自动创建一个 String 对象。

[程序填充、程序设计] String 类和 StringBuffer 类中共有的常用方法:

- (1) length(): 返回字符串的长度,即字符个数。
- (2) charAt(int index): 返回字符串中 index 位置的字符。
- (3) subString(int beginIndex): 截取当前字符串中从 beginIndex 开始到末尾的子串。

[程序填充、程序设计] String 类中的常用方法:

(1) replace(char oldChar, char newChar): 将当前字符串中出现的所有 oldChar 转换为 newChar。

(2) toLowerCase(): 将当前字符串中所有字符转换为小写形式。

(3) toUpperCase(): 将当前字符串中所有字符转换为大写形式。

(4) concat(String str): 将 str 连接在当前字符串的尾部。

(5) startsWith(String prefix): 测试 prefix 是否是当前字符串的前缀。

(6) trim(): 去掉字符串前面及后面的空白。

(7) valueOf(type value): 将 type 类型的参数 value 转换为字符串形式。

[程序填充、程序设计] StringBuffer 类中的常用方法:

(1) append(String str): 将参数 str 表示的字符串添加到当前串的最后。

(2) replace(int start, int end, String str): 使用给定的 str 替换从 start 到 end 之间的子串。

(3) capacity(): 返回当前的容量。

[单选、填空] 系统为 String 类对象分配内存时,按照对象中所含字符的实际个数等量分配。而为 StringBuffer 类对象分配内存时,除去字符所占空间外,再另加 16 个字符大小的缓冲区。对于 StringBuffer 类对象,length() 方法获得的是字符串的长度, capacity() 方法返回当前的容量,即字符串长度再加上缓冲区的大小。

第三节 Vector 类

[单选、填空] Vector 是 java.util 包提供的一个非常重要的工具类, 它类似于数组, 可以使用整数下标来访问各个元素, 但是比数组的功能更强大。

[简答] Vector 类包含的成员变量有 3 个:

(1) protected int capacityIncrement; 增量的大小。如果值为 0, 则缓冲区的大小每次倍增。

(2) protected int elementCount; Vector 对象中元素的数量。

(3) protected Object elementData[]; 元素存储的数组缓冲区。

[简答、程序设计] 常用的 Vector 类的 3 个构造方法:

(1) public Vector(); 构造一个空向量。

(2) public Vector(int initialCapacity); 以指定的初始存储容量 initialCapacity 构造一个空的向量 Vector。

(3) public Vector(int initialCapacity, int capacityIncrement); 以指定的初始存储容量 initialCapacity 和容量增量 capacityIncrement 构造一个空的向量 Vector。

[简答、程序设计] 向 Vector 类对象中添加元素的常用方法:

(1) addElement(Object obj); 将新元素 obj 添加到序列尾部。

(2) insertElementAt(Object obj, int index); 将指定对象 obj 插入到指定位置 index 处。

(3) add(int index, Object obj); 在向量的指定位置 index 插入指定的元素 obj。

[简答、程序设计] 使用以下方法可以修改或删除 Vector 类对象序列中的元素:

(1) setElementAt(Object obj, int index); 将向量序列 index 位置处的对象元素修改为 obj。如果下标 index 是负数或超出实际元素的个数, 则抛出异常 ArrayIndexOutOfBoundsException。

(2) removeElement(Object obj); 删除向量序列中第一个与指定的 obj 对象相同的元素, 同时将后面的所有元素均前移一个位置。这个方法返回的是一个布尔值, 表示删除成功与否。

(3) removeElementAt(int index); 删除 index 位置的元素, 同时将后面的所有元素均前移一个位置。如果下标 index 是负数或超出实际元素的个数, 则抛出异常 ArrayIndexOutOfBoundsException。

(4) removeAllElements(); 清除向量序列中的所有元素, 同时向量的大小置为 0。

[简答、程序设计] Java 还提供了在向量序列中进行查找的操作, 常用的查找方法如下:

(1) elementAt(int index); 返回指定位置处的元素。如果下标 index 是负数或超出实际元素的个数, 则抛出异常 ArrayIndexOutOfBoundsException。这个方法的返回值是 Object 类型的对象, 在使用之前通常需要进行强制类型转换, 将返回的对象引用转换成 Object 类的某个具体子类的对象。

(2) contains(Object obj); 检查向量序列中是否包含指定的对象元素 obj。

(3) indexOf(Object obj, int start_index); 从指定的 start_index 位置开始向后搜索, 返回所找到的第一个与指定对象 obj 相同的元素的下标位置。若指定的对象不存在, 则返回 -1。

(4) lastIndexOf(Object obj, int start_index); 从指定的 start_index 位置开始向前搜索, 返回所找到的第一个与指定对象 obj 相同的元素的下标位置。若指定的对象不存在, 则返回 -1。

第六章 继承与多态

第一节 子类

[程序分析、程序设计] 具有一般性和特殊性的两个类:

public class Employee { // 具有一般性的类

 private String name, jobTitle;

 private Date hireDate, dateOfBirth;

 private int grade;

 ...

}

public class Manager { // 具有特殊性的类

 private String name, jobTitle;

 private Date hireDate, dateOfBirth;

 private int grade; // 以上是与 Employee 共有的属性

 private String department; // 特有的属性

 private Employee[] subordinates; // 特有的属性

 ...

}

从上面的定义可以看出, Manager 类和 Employee 类之间存在重复部分。实际上, 适用于 Employee 的很多属性和方法可以不经修改就被 Manager 所使用。Manager 与 Employee 之间存在“is a”关系, 即 Manager“is a”Employee。

[单选、填空] 与一般的面向对象语言一样, Java 提供了派生机制, 允许程序员用以前已定义的类来定义一个新类。新类称作子类, 原来的类称作父类, 也称为基类或超类。两个类中共同的内容放到父类中, 特殊的内容放到子类中。在定义类时可以表明一个类是不是另一个类的子类。在 Java 中, 用关键字 extends 表示派生, 格式如下:

修饰符 class 子类名 extends 父类名 |

 类体

|

[单选、填空] Object 类是 Java 程序中所有类的直接或间接父类, 处在类层次的