目 录

1 第一个 Java 程序

1.1 名字管理

Java 为了给一个类库生成不会与其他名字混淆的名字，让程序员反过来使用自己的 Internet 域名，从而保证它们是独一无二的。比如域名为 MindView.net，那么各种应用工具库就被命名为 net.mindview.utility.foibles。反转域名后，句点就用来代表子目录的划分。

1.2 static 关键字

现在有两个需求，如下所示：

- 只想为某个特定域分配单一存储空间，而不去考虑究竟要创建多少对象，甚至根本就不创建任何对象。
- 希望某个函数不与包含它的类的任何对象关联在一起。也就是说，即使没有创建对象，也可以调用这个函数。

static 关键字可以满足这两方面的需求。当声明一个事物是 static 的时候，就意味着这个域或这个函数不会与包含它的那个类的任何对象实例关联在一起。

引用 static 变量有两种方法。可以通过一个对象去定位它，也可以通过其类名直接引用。

static 函数的重要用法是在不创建任何对象的前提下就可以调用它。Java 中禁止使用全局函数，这时候 static 就派上了用场。程序员可以通过类本身来调用 static 函数和 static 变量。在没有全局变量和全局函数的情况下，可以在类中置入 static 函数和 static 域。在其他类中就可以直接通过类名访问这些 static 函数和 static 域。

1.3 Hello world

下面是第一个完整的程序，可以打印出“hello world”：

```
1  import java.util.*;
2
3  public class HelloWorld
4  {
5      public static void main(String[] args)
6      {
7          System.out.println("hello world");
8      }
9  }
```

1.4 编码风格

代码风格的规定如下：

- 类名的首字母要大些。如果类名由几个单词构成，那么就把它们并在一起，其中每个内部单词的首字母都采用大写形式。
- 其他内容，第一个字母采用小写。如果该内容由几个单词构成，那么就把它们并在一起，其中每个内部单词的首字母都采用大写形式。

2 初始化与清理

2.1 初始化

类似于 C++ 的方式，Java 也采用了构造器进行初始化。例子如下：

```
class Rock
{
    Rock()
    {
        System.out.println("Hello world");
    }
}
```

和 C++ 一样，Java 的构造器也可以进行重载。例子如下：

```
class Tree
{
    int height;
    Tree()
    {
        System.out.println("Planting a seeding");
        height = 0;
    }
    Tree(int inititalHeight)
    {
        height = inititalHeight;
        System.out.println("Creating new Tree that is " + height + " feer tall"
            );
    }
}
```

关于默认构造器，这里需要注意：如果程序员没有提供任何构造器，编译器将自动生成一个默认构造器；如果程序员写了一个构造器，编译器就不会自动生成一个默认构造器。

2.2 this 关键字

Java 的 this 关键字和 C++ 的 this 关键字有区别。C++ 的 this 关键字是该对象的地址，而 Java 中是没有指针的。Java 的 this 关键字表示该对象的引用。例子如下：

```
public class Leaf
{
    int i = 0;
    Leaf increment()
    {
        ++i;
        return this; // 相当于返回当前对象
    }
}
```

2.2.1 利用 this 调用构造器

在构造器中，如果为 this 添加了参数列表，将产生对符合此参数列表的某个构造器的明确调用。例子如下：

```
public class Flower
{
    int petalCount = 0;
    String s = "initial value";
    Flower(int petals)
    {
        petalCount = petals;
    }
    Flower(String ss)
    {
        s = ss;
    }
    Flower(String s, int petals)
    {
        this(petals);
        // 不能再写this(s)，否则会覆盖原有的内容
        this.s = s;
    }
    void print()
    {
        // 在非构造函数中不能使用this(s)
        System.out.println("Hello world!");
    }
}
```

2.3 清理

Java 拥有一个垃圾回收器，用于释放由 new 分配的内存。在其他情况下，如果该内存区域不是由 new 分配，那么垃圾回收器就不知道该如何释放这块特殊内存。

为了解决这种情况，Java 允许在类中定义一个名为 finalize() 的函数。一旦垃圾回收器准备释放对象占用的存储空间，将首先调用其 finalize() 函数，并在垃圾回收动作发生时，真正地回收对象占用的内存。

finalize() 函数与 C++ 中的析构函数有区别，因为 C++ 中的对象一定会被销毁，而 Java 中的对象不一定总是被垃圾回收器回收，而且垃圾回收并不等于析构。需要知道的是，只要程序没有濒临存储空间用完的时刻，垃圾回收器就不会释放程序所创建的任何对象的存储空间。

2.3.1 finalize 函数的用途

finalize 函数不会负责释放对象所占有的内存。无论对象是如何创建的，垃圾回收器都会负责释放对象占据的所有内存。只有当 Java 程序中调用了本地方法分配空间时，finalize 函数才会派上用场。本地方法是一种在 Java 中调用非 Java 代码的方式。在非 Java 代码中，可能会调用 C 的 malloc 函数分配存储空间，此时只有使用了 free 函数，该内存空间才会释放。所以在对象释放时，需要在 finalize 函数中调用 free 函数来释放这块特殊的内存，否则将引起内存泄漏。

下面是使用 finalize() 函数的例子：

```
class Book
{
    boolean checkedOut = false;
    Book(boolean checkOut)
    {
        checkedOut = checkOut;
    }
    void checkIn()
    {
        checkedOut = false;
    }
    protected void finalize()
    {
        if(checkedOut)
            System.out.println("Error: checked out");
    }
}

public class TerminationCondition
{
    public static void main(String[] args)
    {
        Book novel = new Book(true);
        novel.checkIn();
        new Book(true);
        System.gc();
    }
}
```

以上程序通过 finalize() 函数确保所有 Book 对象在被当作垃圾回收钱都应该被 check

in。当有一本书没有 check in 时，程序将输出错误情况。类似的，如果一个对象代表了一个打开的文件，在对象被回收前程序员应该关闭这个文件。finalize() 函数可以用来检查文件是否关闭。finalize 函数更多地被用来发现程序中隐晦的缺陷。

2.3.2 垃圾回收器的工作机制

首先需要意识到，Java 中除了基本类型，所有对象都在堆上分配空间。然而 Java 从堆分配空间的速度，可以和其他语言从堆栈上分配空间的速度相媲美。

对比一下 Java 与 C++ 的堆空间分配机制。C++ 的堆像一个院子，里面的每个对象都负责管理自己的地盘。如果对象被销毁了，这个地盘必须加以重新利用。在 Java 中，堆更像一个堆栈，每分配一个新对象，它就往前移动一格。

Java 这样的实现方式会导致频繁的内存页面调度。为了避免这种情况的出现，Java 使用了垃圾回收器。垃圾回收器工作的时候，一边回收空间，一边讲堆中的对象排列紧凑。通过垃圾回收器对对象重新排列，实现了一种告诉的、有无限空间可供分配的堆模型。