蓝桥 LAN QIAO

Java程序设计基础

包和访问控制

包和访问控制主要内容

- 7.1 Java 包概述
- 7.2 引用包（重点）
- 7.3 访问控制（重点）
- 7.4 static 关键字

教学目标

- 能理解包概念
- 能正确使用包避免类重名问题
- 能正确使用包对多个类进行归类
- 能描述包的命名规则
- 能理解引用包的作用和场景
- 能正确的在程序中打包并导入包
- 能区分四种访问修饰符
- 能准确使用四种访问修饰符
- 能理解 static 关键字的作用
- 能正确使用 static 关键字修饰类的成员
- 能理解 Java 静态块
- 能正确使用初始化块和静态初始化块初始化程序
- 能理解并编写单例设计模式

包的简介

- 计算机中存放了若干类型的文档，为了管理方便，操作系统采用了树形结构的文件夹形式存放这些文档，并对文档进行管理。
- 类似的，为了更好地组织类，Java 提供了包机制。
- 包是类的容器，用于分隔类名空间。如果没有指定包名，所有的类都属于一个默认的无名包。Java 中将实现相关功能的类组织到一个包中。例如，Java 中通用的工具类，一般都放在 java.util 包中。
- 总的来说，包有以下三个方面的作用：
  - 提供了类似于操作系统树形文件夹的组织形式，能分门别类地存储、管理类，易于查找并使用类。
  - 解决了同名类的命名冲突问题。
  - 包允许在更广的范围内保护类、属性和方法。

包的语法

- 程序员可以使用 package 关键字指明源文件中的类属于哪个具体的包，包的语法形式如下。

  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可执行语句，它的前面只能有注释或空行。
- 一个文件中最多只能有一条 package 语句，即只能把一个类放在一个包中。
- 包的名字应该有层次关系，各层之间以.分隔。

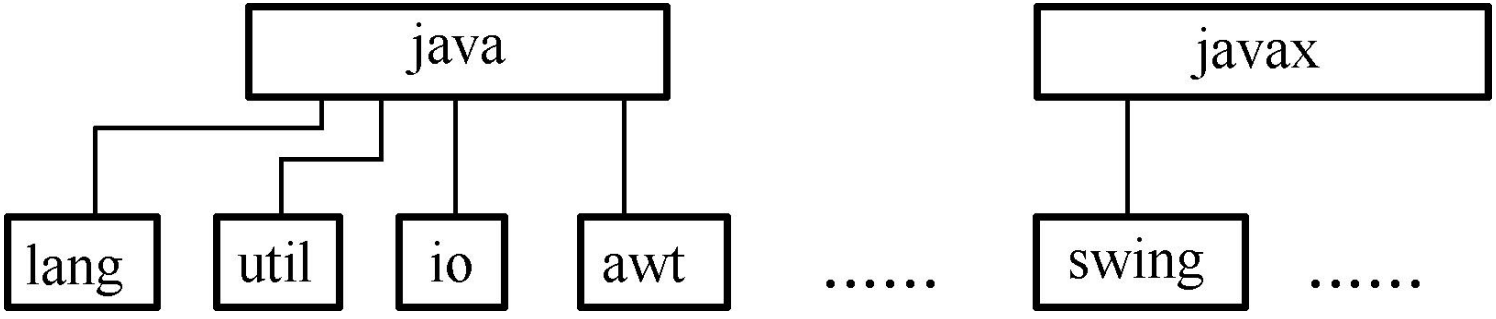
包的命名规则

- 命名规则

  通常包名全部用小写字母，这与类名以大写字母开头且各单词的首字母亦大写的命名约定有所不同。关于包的命名，现在使用最多的规则是使用翻转的 internet 域名（不含 www、ftp 等访问协议）。

  例如 abc 公司的域名为 abc.com，该公司开发部门正开发一个名为 fly 的项目，在这个项目中有一个工具类的包，则这个工具包的包名可以为：com.abc.fly.tools。

JDK 类库里的包



- java.lang：lang 是 language 的简写，这个包提供 Java 语言的基础类，例如 String、Math、Integer、System 和 Thread 等。
- java.util：util 是 utility 的简写，组织了 Java 的工具类，包含集合、事件模型、日期和时间设置、国际化和各种实用工具类。
- java.io：io 是 input 和 output 的合并简写，指输入和输出，组织了数据流、序列化和文件系统相关的类。
- java.net：net 即网络，这个包组织了为实现网络应用程序而提供的类。
- java.awt：抽象窗口工具集(Abstract Window Toolkit)，包含用于创建用户界面和绘制图形图像的类。

引用包的作用

- 思考：如果两个 Java 类不在同一个包中，这两个类之间如何相互引用

  引用包才能解决上述问题

引用不同包中类的两种方式

- 一种非常直观的方法就是使用完整类名引用类，即 包名+类名（也称为类的全限定名）。
- 另一种是导入包的形式。使用类的全限定名的方法虽然直观，但书写的内容多，且当使用的类比较多时，编辑和阅读都非常困难，因此并不推荐。接下来学习的是采用导入包的形式引用类，导入包的语法形式如下。

  import 包名.类名;

- 这里的包名、类名既可以是 JDK 提供的包和类的名称，也可以是用户自定义的包名和类名。
- 如果要使用一个包中的多个类，可以使用 import 包名.* ; 的形式导入这个包中所有的类。不过，包的导入只能导入当前目录中的类，而不能导入其子目录中的类。
- 例如在导入 java.util.* 时，只会导入 java.util 包中的所有类，但不能导入 java.util.function 包中的类。另外，import 语句需要放在 package 语句后，在类定义之前。

同时导入多个包

- 如果一个程序同时存在 Date 类和 Scanner 类，就可以通过两条 import 语句分别导入二者，如下所示。

```
import java.util.Date;                    //导入java.util包中的Date类
import java.util.Scanner;                 //导入java.util包中的Scanner类
```

- java 还提供了一种批量导入的方式：如果要导入的多个类存在于同一个包中，那么可以使用通配符 * 代表包中的所有类。例如可以使用 import java.util.* 代表导入了 java.util 包中的所有类，如下所示。

```
import java.util.*;
```

访问权限修饰符

- 有些类并不希望被其他类使用，有些属性和方法需要对外界不可见或仅在有限的范围内可见。
- Java 语言中的访问权限修饰符有 4 种，但却只有 3 个关键字。因为不写访问权限修饰符时，在 Java 中被称为默认权限（包权限），本课程中以 default 代替。其他 3 个访问权限修饰符分别为 private、protected 和 public。

如何实现对类的访问控制

- 对于类而言，目前能使用的访问权限修饰符只有 public 和 default
- 如果使用 public 修饰某个类，则表示该类在任何地方都能被访问，如果不写访问权限修饰符，则该类只能在本包中使用。

4 种类成员访问权限修饰符

- 对于类的成员（属性和方法）而言，4 种访问权限修饰符都可以使用。下面按照权限从小到大的顺序（即 private < default < protected < public）对四种访问权限修饰符分别进行介绍。

- 私有权限 private

  private 可以修饰属性、构造方法、普通方法。被 private 修饰的类成员只能在定义它们的类中使用，在其他类中都不能访问。

- 默认权限 default

  不写任何权限关键字就代表使用默认权限，属性、构造方法、普通方法都能使用默认权限。默认权限也称为同包权限。同包权限的元素只能在定义它们的类中以及同包的类中被调用。

4 种类成员访问权限修饰符

- 受保护权限 protected

  protected 可修饰属性、构造方法、普通方法，能在定义它们的类中以及同包的类中调用被 protected 修饰的成员。如果有不同包中的类想调用它们，那么这个类必须是这些成员所属类的子类。关于子类及相关概念，将会在后续讲解继承的时候详细介绍。

- 公共权限 public

  public 可以修饰属性、构造方法和普通方法。被 public 修饰的成员，可以在任何一个类中被调用，是权限最大的访问权限修饰符。

4 种类成员访问权限修饰符

修饰符	类内部	同一个包中	子类	任何地方
private	Yes			
default	Yes	Yes		
protected	Yes	Yes	Yes	
public	Yes	Yes	Yes	Yes

Static 关键字的作用

- 对象的成员变量有两种级别的使用范围：对象级别和类级别。
- “对象名.变量名”的形式访问，实际就是对象级别的访问形式。对象级别的成员变量只能在当前对象的范围内使用。
- 在类成员的声明前，加上 static（静态的）关键字，就能创建出类级别的成员变量。声明为 static 的变量称为静态变量或类变量。可以直接通过类名引用静态变量，也可以通过实例名来引用静态变量，但推荐采用前者，因为采用后者容易混淆静态变量和实例变量。
- 除了修饰变量以外，声明为 static 的方法称为静态方法或类方法，最常见的例子是 main 方法。和静态变量一样，静态方法也可以被类名直接引用。
- 静态方法可以直接调用静态方法，访问静态变量，但是不能直接访问实例变量和实例方法。静态方法中不能使用 this 关键字，因为静态方法不属于任何一个实例。

静态方法不能操作实例变量

- 静态方法可以操作静态变量，不能操作实例变量，通过下面的例子看出(编译时报错)。

```
public class Student{
public int avgAge = 22;            //实例变量，存放平均年龄
public static void showAvgAge(){
    //静态方法调用实例变量—编译出错
System.out.println("静态方法输出所在班平均年龄为：" + avgAge);
    }
}
```

```
shiyanlou:project/ $ javac Student.java
Student.java:5: 错误: 无法从静态上下文中引用非静态 变量 avgAge
System.out.println("静态方法输出所在班平均年龄为: " + avgAge);
                                                  ^
1 个错误
```

Java 静态块

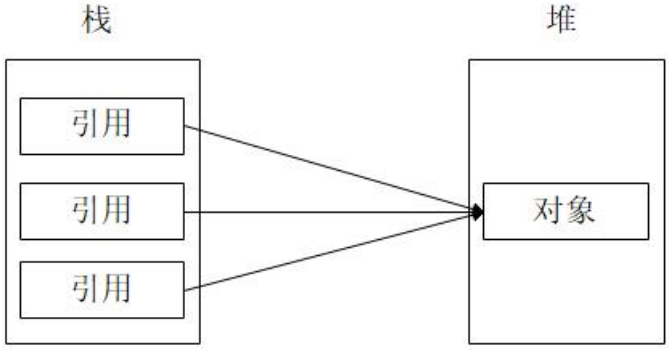
- 静态块的语法形式如下。

```
static{
        语句块
    }
```

- Java 类首次装入 JVM 时，会对静态成员或静态块进行一次初始化，注意此时还没有产生对象。因此，静态成员和静态块都是和类绑定的，会在类加载时就进行初始化操作。
- 当类加载完毕后，才能实例化出对象，并在对象产生的同时对实例成员进行初始化。因此，实例成员是和对象绑定的，会在实例化对象时一并进行初始化。

单例模式

- 单例模式是指：无论创建了多少个引用，在堆中仅仅只有一个实例对象，如图所示。



单例模式

- 实现单例模式的核心是思路是将构造方法私有化，即使用 private 修饰构造方法，然后利用 static 成员变量的“一次性”，如下所示。

```
public class Singleton {
    private Singleton() {
    }
}
```

单例模式

- 完整代码，如下所示。

```
public class Singleton {
    // 在类加载时就创建实例
    private static final Singleton instance = new Singleton();
    // 私有构造方法，防止外部实例化
    private Singleton() {
    }
    // 提供一个公共方法来获取实例
    public static Singleton getInstance() {
        return instance;
    }
}
```

单例模式

- 在 main 方法中调用并验证单例模式的实现

```
public static void main(String[] args) {
    // 获取两个实例
    Singleton instance1 = Singleton.getInstance();
    Singleton instance2 = Singleton.getInstance();

    // 验证两个实例是否相同
    if (instance1 == instance2) {
        System.out.println("两个实例是相同的。");
    } else {
        System.out.println("两个实例是不同的。");
    }
}
```

笔试题

- 下面那种权限是同一包可以访问，不同包的子类可以访问，不同包的非子类不可以访问（　）？

  A　private

  B default（默认）

  C　protected

  D　public

笔试题

- 下面对static的描述错误的是（ ）？

  A 静态修饰的成员变量和成员方法随着类的加载而加载

  B 静态修饰的成员方法可以访问非静态成员变量

  C 静态修饰的成员可以被整个类对象所共享

  D 静态修饰的成员变量和成员方法随着类的消失而消失

笔试题

- 下面对单例设计模式的描述错误的是（ ）？

A 在单例模式中，不论创建多少个引用，都只有一份堆对象

B 单例模式可以通过构造方法私有化实现

C 单例模式可以减少程序中堆对象的个数

D 单例模式是工厂模式的另一种别名