第十三章 包的管理和静态导入

1. 包（非重点）

1.1 包的概述

1. 包在硬盘上就是一个文件夹
2. 包是用来管理类的，让类具有模块化，使其结构更加清晰
3. 包的层级关系在java中就是小圆点
    a. com.powernode.itf
    b. com文件夹下有一个文件夹powernode，在powernode文件夹下有一个itf文件夹
4. 包的使用我们需要掌握两个关键字
    a. package：告诉编译器，类或者接口属于哪个包
    b. import:导入类（当前类使用了其他的类，需要导入）
5. 注意：java.lang包的类可以直接使用，不用导入

代码块

```
package com.powernode.package14;


import java.util.Arrays;

public class Test {
    public static void main(String[] args) {
        Arrays.asList(1, 2);
        String str = "123";
    }
}
```

1.2 类的三种导入方式

代码块

```
package com.powernode.package15.itf;

public interface Flyer {
    void land();
```

```
}


package com.powernode.package15.itf;

public interface Swimming {
    void swim();
}


package com.powernode.package15.amodel;

public abstract class Pet {
    public abstract void eat();
    public abstract void sleep();
}



package com.powernode.package15.model;


import com.powernode.package15.amodel.Pet;
import com.powernode.package15.itf.Flyer;
import com.powernode.package15.itf.Swimming;

public class Bird extends Pet implements Flyer, Swimming {
    @Override
    public void eat() {

    }

    @Override
    public void sleep() {

    }

    @Override
    public void land() {

    }

    @Override
    public void swim() {

    }
}
```

```
52
53
54
55
56
57  package com.powernode.package15.test;
58
59
60  //1.导入方式（一）：包名 + 类名
61  import com.powernode.package15.model.Bird;
62  /*import com.powernode.package15.itf.Flyer;
63  import com.powernode.package15.itf.Swimming;*/
64  //2.导入方式（二）：包名 + *（*代表包下所有的成员）
65  import com.powernode.package15.itf.*;
66  public class Test {
67      public static void main(String[] args) {
68          Flyer flyer = new Bird();
69          Swimming swimming = new Bird();
70          //3.导入方式（三）:使用类是直接带上包名
71          com.powernode.package15.amodel.Pet pet = new Bird();
72      }
73  }
74
75
76
```

1.3 静态导入

代码块

```
1   package com.powernode.package16.tools;
2
3   public class IMath {
4       public static final double PI = 3.14;
5       public static double cleArea(int r){
6           return PI * r * r;
7       }
8   }
9
10
11  package com.powernode.package16;
12
13  import com.powernode.package16.tools.IMath;
14
15  public class Test01 {
16      public static void main(String[] args) {
```

```
        System.out.println(IMath.PI);
        System.out.println(IMath.cleArea(10));
    }
}


package com.powernode.package16;
//静态导入：所导入的成员必须是静态的
/*import static com.powernode.package16.tools.IMath.PI;
import static com.powernode.package16.tools.IMath.cleArea;*/

import static com.powernode.package16.tools.IMath.*;

public class Test02 {
    public static void main(String[] args) {
        //静态导入可读性差，在实际的开发过程中使用不多
        System.out.println(PI);
        System.out.println(cleArea(10));
    }
}
```

2. 方法权限修饰符（非重点）

2.1 私有的不可以夸类访问

代码块

```
package com.powernode.package17.model;

 class Teacher {
     /**
      * 1.public:公共的
      * 2.protected: 受保护的
      * 3.default: 默认的（不写）
      * 4.private :私有的
      * 注意：protected和private不可以修饰类
      * 类的访问修饰符主要用于限定类在其他类或包中的可见性，以实现封装和模块化设计。
      * Java 中外部类仅支持两种访问修饰符：
      * public：所有包中的类都可访问。
      * 默认（无修饰符）：仅同一包中的类可访问。
      */
     public String name = "zs";
     protected int age = 23;
     char sex = '男';
     private String address = "北京";
```

```
}


package com.powernode.package17.model;

public class Test {
    public static void main(String[] args) {
        Teacher teacher = new Teacher();
        System.out.println(teacher.name);
        System.out.println(teacher.age);
        System.out.println(teacher.sex);
        //System.out.println(teacher.address);
    }
}
```

2.2 权限修饰符的总结

代码块

```
package com.powernode.package17.test;
//跨包访问类，该类必须是public的
import com.powernode.package17.model.Teacher;

public class Test extends Teacher{
    public static void main(String[] args) {
        /**
         * 访问权限修饰符的总结（记住）
         *    修饰符        同类      同包      子类      全局
         *    private      可以      不可以    不可以    不可以
         *    default      可以      可以      不可以    不可以
         *    protected    可以      可以      可以      不可以
         *    public       可以      可以      可以      可以
         *    从大到小的权限排序：public > protected > default > private
         */
        Teacher teacher = new Teacher();
        System.out.println(teacher.name);
        //System.out.println(teacher.age);

    }
    public void method(){
        System.out.println(name);
        System.out.println(age);
        //System.out.println(sex);
        //System.out.println(address);
    }
}
```

作业

1. 练习题接口（一个类继承了另一个类并实现接口）；
    a. 声明Clothing（ 衣服）接口 ，在接口中声明calcArea(double size)方法；(size * 1.3)
    b. 声明一个抽象类Frock（女装），定义public变量color，创建构造器为color赋值，定义抽象方法getColor();
    c. 定义一个Shirt（衬衫）类，声明一个price实例变量，创建构造器为price赋值，并提供get和set方法
    d. **Shirt继承Frock抽象类并实现Clothing接口**。
    e. 在Test类的main方法中：
        i. 使用本态引用创建Shirt对象，并调用calcArea(double size)方法，打印计算结果。
        ii. 使用Frock多态引用创建Shirt对象，并调用calcArea(double size)方法，打印计算结果。
        iii. 使用Clothing多态引用创建Shirt对象，并调用calcArea(double size)方法，打印计算结果。
2. 练习题接口（**Shirt继承了Frock，Frock实现了Clothing**）
3. 练习题（接口继承接口）
    a. 声明一个接口Collection 有一个add(Object obj)方法 【传入一个对象直接输出】
    b. 声明一个接口 Set 继承Collection 接口 ，Set接口有一个 方法 max(int x ,int y)
    c. 声明一个实现类HashSet 实现了Set接口
    d. 使用多态分别调用，add方法和max方法
4. 练习题（接口多实现）
    a. 实现类HashSet实现Set,实现Collection
    b. 使用多态分别调用，add方法和max方法