7.7 作用域

一句话解释：变量有效的区域

7.7.1 基本使用

全局变量（属性）

在类中定义的变量，也被称为属性，其作用域为整个类体。这意味着在该类的任何方法中都可以访问和使用这些变量。例如，我们来看下面的`Cat`类：

```
class Cat {
    // 全局变量：也就是属性，作用域为整个类体Cat类：cry eat等方法使用属性
    // 属性在定义时，可以直接赋值
    int age = 10; // 指定的值是10
    // 全局变量(属性)可以不赋值，直接使用，因为有默认值
    double weight; // 默认值是0.0

    public void hi() {
        System.out.println("weight=" + weight);// 属性
    }

    public void cry() {
        System.out.println("在cry中使用属性age=" + age);
    }

    public void eat() {
        System.out.println("在eat中使用属性age=" + age);
    }
}
```

在这个`Cat`类中，`age`和`weight`就是全局变量（属性）。`age`在定义时被初始化为 10，而`weight`虽然没有显式赋值，但它有默认值 0.0。在`hi`、`cry`和`eat`等方法中，都可以直接访问和使用这些属性。

局部变量

局部变量一般是指在成员方法中定义的变量。它们的作用域仅限于声明它们的代码块内。例如：

```
public void cry() {
    // 1. 局部变量一般是指在成员方法中定义的变量
    // 2. n和name就是局部变量
    // 3. n和name的作用域在cry方法中
    int n = 10;
    String name = "jack";
    System.out.println("在cry中使用属性age=" + age);
}
```

在`cry`方法中，`n`和`name`是局部变量。它们只在`cry`方法内部有效，当`cry`方法执行结束，这些局部变量就会被销毁。需要注意的是，局部变量在使用前必须先赋值，因为它们没有默认值。

再看一个更全面的例子：

```java
public class VarScope {
    public static void main(String[] args) {
    }
}

class Cat {
    int age = 10;
    double weight;

    public void hi() {
        // 局部变量必须赋值后，才能使用，因为没有默认值
        int num = 1;
        String address = "北京的猫";
        System.out.println("num=" + num);
        System.out.println("address=" + address);
        System.out.println("weight=" + weight);
    }

    public void cry() {
        int n = 10;
        String name = "jack";
        System.out.println("在cry中使用属性age=" + age);
    }

    public void eat() {
        System.out.println("在eat中使用属性age=" + age);
        // System.out.println("在eat中使用cry的变量name=" + name);//错误
    }
}
```

在`hi`方法中，`num`和`address`是局部变量，它们在方法内被赋值并使用。而在`eat`方法中，尝试访问`cry`方法中的局部变量`name`会导致错误，因为`name`的作用域仅限于`cry`方法内部。

7.7.2 注意事项和细节使用

变量的生命周期

1. **全局变量（属性）**：其生命周期较长，伴随着对象的创建而创建，伴随着对象的销毁而销毁。例如，当创建一个`Person`类的对象时，该对象的属性也随之创建并分配内存空间；当对象不再被引用，被垃圾回收机制回收时，其属性也被销毁。

```java
class Person {
    public int age = 20;
    String name = "jack";
}

public class VarScopeDetail {
    public static void main(String[] args) {
        Person p1 = new Person();
        // 属性生命周期较长，伴随着对象的创建而创建，伴随着对象的销毁而销毁。
    }
}
```

1. **局部变量**：生命周期较短，伴随着它所在的代码块的执行而创建，伴随着代码块的结束而销毁。例如，在一个方法调用过程中，方法内的局部变量在方法开始执行时创建，当方法执行完毕返回时，这些局部变量就会被销毁，释放所占用的内存空间。

```
class Person {
    public void say() {
        String name = "king";
        System.out.println("say() name=" + name);
    }
}

public class VarScopeDetail {
    public static void main(String[] args) {
        Person p1 = new Person();
        p1.say();// 当执行say方法时，say方法的局部变量比如name,会创建，当say执行完毕后
        // name局部变量就销毁,但是属性(全局变量)仍然可以使用
    }
}
```

跨类访问对象属性

在 Java 中，一个类可以通过创建另一个类的对象来访问其属性。例如：

```
class T {
    // 全局变量/属性：可以被本类使用，或其他类使用（通过对象调用）
    public void test() {
        Person p1 = new Person();
        System.out.println(p1.name);// jack
    }

    public void test2(Person p) {
        System.out.println(p.name);// jack
    }
}

class Person {
    String name = "jack";
}

public class VarScopeDetail {
    public static void main(String[] args) {
        T t1 = new T();
        t1.test(); // 第1种跨类访问对象属性的方式
        t1.test2(new Person());// 第2种跨类访问对象属性的方式
    }
}
```

在上述代码中，`T` 类的 `test` 方法通过创建 `Person` 类的对象 `p1`，访问了 `p1` 的 `name` 属性。`test2` 方法则通过接收一个 `Person` 类的对象参数 `p`，访问了 `p` 的 `name` 属性。这展示了两种常见的跨类访问对象属性的方式。

修饰符的使用

1. **属性**：属性可以添加修饰符，如`public`、`protected`、`private`等，这些修饰符用于控制属性的访问权限。`public`修饰的属性可以被任何类访问；`protected`修饰的属性可以被同一包中的类以及该类的子类访问；`private`修饰的属性只能被该类自身访问。

```
class Person {
    // 细节: 属性可以加修饰符(public protected private..)
    public int age = 20;
    protected String hobby = "reading";
    private double salary = 5000.0;
}
```

1. **局部变量**：局部变量不能添加修饰符。例如，下面的代码是错误的：

```
public void hi() {
    public String address = "北京";// 错误，局部变量不能加修饰符
}
```

变量的重名问题

属性和局部变量可以重名，当出现重名时，访问遵循就近原则。即在代码块内，如果局部变量和属性同名，优先访问局部变量。例如：

```
class Person {
    String name = "jack";

    public void say() {
        // 细节 属性和局部变量可以重名，访问时遵循就近原则
        String name = "king";
        System.out.println("say() name=" + name);
    }
}
```

在`say`方法中，定义了一个与属性`name`重名的局部变量`name`，此时`System.out.println("say() name=" + name);`输出的是局部变量`name`的值`"king"`。如果要访问属性`name`，可以使用`this`关键字，即`System.out.println("say() this.name=" + this.name);`，这样输出的就是属性`name`的值`"jack"`。

局部变量的重复定义

在同一个代码块内，不能重复定义同名的局部变量。例如：

```
public void hi() {
    String address = "北京";
    String address = "上海";// 错误,重复定义变量
    String name = "hsp";// 可以
}
```

在`hi`方法中，第二次定义`address`变量会导致编译错误，因为在同一个方法内，局部变量名必须唯一。但定义不同名的局部变量`name`是允许的。

课后练习

一、选择题

1. 以下关于 Java 中全局变量（属性）和局部变量的说法，正确的是（ ）
   A. 全局变量必须在定义时初始化，局部变量可以不初始化
   B. 全局变量的作用域是整个项目，局部变量的作用域是所在方法
   C. 全局变量的生命周期长于局部变量
   D. 局部变量可以使用 public 修饰符

2. 有如下 Java 代码，输出结果是（ ）

```
class Test {
    int num = 10;
    public void show() {
        int num = 20;
        System.out.println(num);
    }
}
public class Main {
    public static void main(String[] args) {
        Test t = new Test();
        t.show();
    }
}
```

   A. 10
   B. 20
   C. 编译错误
   D. 运行时错误

3. 下列变量定义中，属于局部变量的是（ ）
   A. 在类中，方法外定义的变量
   B. 在方法中定义的变量
   C. 在接口中定义的变量
   D. 在构造方法中定义的变量

4. 若在一个类的方法中定义了与类属性同名的局部变量，在该方法内访问此变量名时，访问的是（ ）
   A. 类属性
   B. 局部变量
   C. 编译时会报错
   D. 运行时会报错

5. 以下关于变量生命周期的说法，错误的是（ ）
   A. 全局变量（属性）伴随着对象的创建而创建，对象销毁时销毁
   B. 局部变量在其所在代码块开始执行时创建，代码块结束时销毁
   C. 方法参数作为局部变量，在方法调用时创建，方法结束时销毁
   D. 全局变量在程序启动时创建，程序结束时销毁

6. 有如下代码，关于变量作用域说法正确的是（ ）

```
class ScopeExample {
    int globalVar;
    public void method1() {
        int localVar1 = 1;
        globalVar = localVar1;
    }
    public void method2() {
        int localVar2 = 2;
        // 此处能访问 localVar1 吗?
    }
}
```

A. 在 method2 中能访问 method1 中的 localVar1
B. 在 method2 中不能访问 method1 中的 localVar1
C. 在 method1 中不能访问 globalVar
D. globalVar 的作用域是 method1 和 method2

7. 局部变量不能使用的修饰符是（ ）
A. final
B. static
C. 什么修饰符都不能用
D. private

8. 下列代码中，没有语法错误的是（ ）
A.

```
class A {
    int a;
    public void m() {
        int a = 1;
        a = a + 1;
    }
}
```

B.

```
class B {
    public void n() {
        int b;
        System.out.println(b);
    }
}
```

C.

```
class C {
    int c;
    public void p() {
        static int c = 2;
    }
}
```

D.

```
class D {
    private int d;
    public void q() {
        private int d = 3;
    }
}
```

9. 跨类访问对象属性时，正确的做法是（ ）
   A. 直接访问另一个类的属性
   B. 创建另一个类的对象，通过对象访问其属性
   C. 只能访问另一个类的 public 属性
   D. B 和 C 都对
10. 以下关于作用域的说法，正确的是（ ）
    A. 一个方法中不能有两个同名的局部变量
    B. 全局变量和局部变量不能同名
    C. 局部变量的作用域可以跨方法
    D. 全局变量的作用域可以跨类

二、填空题

1. 在 Java 中，定义在类中、方法外的变量称为__，其作用域为__。
2. 局部变量在使用前必须__，因为它没有__。
3. 当属性和局部变量重名时，在局部变量所在的代码块内，访问变量遵循__原则。
4. 全局变量（属性）的生命周期伴随着__的创建而创建，伴随着__的销毁而销毁。
5. 局部变量的生命周期伴随着它所在的__的执行而创建，伴随着__的结束而销毁。
6. 跨类访问对象属性的方式是创建__，然后通过__访问其属性。
7. 全局变量可以添加__等修饰符来控制访问权限，局部变量__（能 / 不能）添加修饰符。
8. 在一个方法内定义的局部变量，其作用域是__。
9. 若有类 A，在类 A 的方法中访问类 A 的属性，可以使用__关键字。
10. 当在一个方法中定义了与类属性同名的局部变量，若要在该方法内访问类属性，应使用__。

三、判断题

1. 全局变量可以在定义时不赋值，因为有默认值。（ ）
2. 局部变量的作用域可以超出其所在的方法。（ ）
3. 一个类中不能有同名的全局变量和局部变量。（ ）
4. 全局变量的生命周期比局部变量长。（ ）
5. 局部变量可以使用 protected 修饰符。（ ）
6. 在一个方法中，可以先使用局部变量，再定义该局部变量。（ ）
7. 跨类访问对象属性时，只能访问 public 属性。（ ）
8. 全局变量（属性）在整个程序运行期间都占用内存。（ ）
9. 当属性和局部变量重名时，在方法内访问的一定是局部变量。（ ）
10. 方法参数属于局部变量。（ ）

四、简答题

1. 简述 Java 中全局变量和局部变量的区别。
2. 说明变量作用域与变量生命周期的关系。
3. 解释当属性和局部变量重名时，Java 是如何确定访问的变量的。
4. 局部变量和全局变量在修饰符使用上有什么不同？为什么会有这样的不同？
5. 举例说明跨类访问对象属性的方法，并解释其原理。

6. 请描述在一个方法中定义的局部变量的生命周期和作用域。
7. 全局变量在什么情况下会被销毁?
8. 为什么局部变量在使用前必须初始化?
9. 一个方法中定义了多个局部变量，它们的作用域是如何确定的?
10. 简述作用域在 Java 编程中的重要性。

五、编程题

1. 编写一个 Java 类 `Calculator`，包含一个全局变量 `result` 用于存储计算结果。定义两个方法：`add` 方法接收两个整数参数，将它们相加后赋值给 `result`；`displayResult` 方法用于输出 `result` 的值。在 `main` 方法中创建 `Calculator` 类的对象，调用 `add` 方法和 `displayResult` 方法，验证全局变量的使用。
2. 设计一个类 `Employee`，包含全局变量 `name` 和 `salary`。在类中定义一个方法 `calculateBonus`，在该方法内定义局部变量 `bonus`，根据 `salary` 计算奖金（例如奖金为工资的 10%），并将结果赋值给 `bonus`，最后输出员工姓名、工资和奖金。在 `main` 方法中创建 `Employee` 类的对象，设置员工姓名和工资，调用 `calculateBonus` 方法。
3. 编写一个类 `Circle`，包含全局变量 `radius` 表示圆的半径。定义一个方法 `calculateArea`，在方法内定义局部变量 `pi`（值为 3.14），通过半径和 `pi` 计算圆的面积并返回。在 `main` 方法中创建 `Circle` 类的对象，设置半径，调用 `calculateArea` 方法并输出圆的面积。
4. 定义一个类 `Student`，包含全局变量 `grades`（一个整数数组，用于存储学生的成绩）。编写一个方法 `calculateAverageGrade`，在方法内定义局部变量 `sum` 和 `average`，计算学生成绩的总和与平均值，最后返回平均值。在 `main` 方法中创建 `Student` 类的对象，初始化 `grades` 数组，调用 `calculateAverageGrade` 方法并输出平均成绩。
5. 编写一个 Java 类 `BankAccount`，包含全局变量 `balance` 表示账户余额。定义两个方法：`deposit` 方法接收一个双精度浮点数参数 `amount`，将其加到 `balance` 上；`withdraw` 方法接收一个双精度浮点数参数 `amount`，如果 `balance` 大于等于 `amount`，则从 `balance` 中减去 `amount`，否则输出提示信息 “余额不足”。在 `main` 方法中创建 `BankAccount` 类的对象，进行存款和取款操作，并输出每次操作后的余额。
6. 设计一个类 `Book`，包含全局变量 `title` 和 `author`。编写一个方法 `printBookInfo`，在方法内定义局部变量 `info`，将书名和作者信息拼接成一个字符串赋值给 `info`，然后输出 `info`。在 `main` 方法中创建 `Book` 类的对象，设置书名和作者，调用 `printBookInfo` 方法。
7. 编写一个类 `TemperatureConverter`，包含全局变量 `celsius` 表示摄氏温度。定义一个方法 `convertToFahrenheit`，在方法内定义局部变量 `fahrenheit`，根据公式（华氏温度 = 摄氏温度 * 1.8 + 32）将摄氏温度转换为华氏温度并赋值给 `fahrenheit`，最后返回 `fahrenheit`。在 `main` 方法中创建 `TemperatureConverter` 类的对象，设置摄氏温度，调用 `convertToFahrenheit` 方法并输出华氏温度。
8. 定义一个类 `Rectangle`，包含全局变量 `length` 和 `width`。编写一个方法 `calculatePerimeter`，在方法内定义局部变量 `perimeter`，通过长和宽计算矩形的周长并赋值给 `perimeter`，最后返回 `perimeter`。在 `main` 方法中创建 `Rectangle` 类的对象，设置长和宽，调用 `calculatePerimeter` 方法并输出矩形的周长。
9. 编写一个 Java 类 `ShoppingCart`，包含全局变量 `items`（一个字符串数组，用于存储购物车中的商品）。定义一个方法 `addItem`，接收一个字符串参数 `item`，将其添加到 `items` 数组中（可以先创建一个新的更大的数组，将原数组元素复制到新数组，再将新商品添加到新数组）。在 `main` 方法中创建 `ShoppingCart` 类的对象，调用 `addItem` 方法添加商品，并输出购物车中的商品列表。
10. 设计一个类 `GameCharacter`，包含全局变量 `health` 表示角色生命值。编写一个方法 `takeDamage`，接收一个整数参数 `damage`，在方法内定义局部变量 `newHealth`，根据受到的伤害计算新的生命值（新生命值 = 当前生命值 - 伤害值，如果新生命值小于 0 则设为 0），并将新生命值赋值给 `newHealth`，最后更新全局变量 `health`。在 `main` 方法中创建 `GameCharacter` 类的对象，设置初始生命值，调用 `takeDamage` 方法并输出角色当前生命值。