

个人所得税计算器设计文档（MVC架构）

作者：胡瑞康

学号：2020123456

1 项目概述

个人所得税计算器是一款基于MVC架构的Java命令行应用程序，支持动态调整起征点和税率表，提供税款计算与配置功能。通过分层设计实现高内聚低耦合，满足未来税法变更的扩展需求。

2 文件结构

```
src/
├── main/
│   └── java/
│       └── com/tax/
│           ├── controller/ # 控制器层
│           ├── model/     # 模型层
│           ├── view/      # 视图层
│           └── PersonalTaxApp.java # 应用入口
└── test/
    └── java/
        └── com/tax/
            ├── controller/ # 控制器测试
            └── model/      # 模型测试
```

3 系统架构（MVC模式）

3.1 分层说明

层级	组件	职责
视图层	TaxView	处理用户输入输出
控制层	TaxController	处理业务流程与请求转发
模型层	TaxCalculator	税款计算核心逻辑
	TaxBracket	税率区间数据实体

4 UML类图



5 类设计

5.1 模型层（Model）

5.2 TaxBracket 类

职责：封装单个税率区间的上下限和税率，提供数据验证与格式化输出。

属性：

- `lowerBound`：税率生效的最低应纳税所得额
- `upperBound`：税率生效的最高应纳税所得额（`Double.MAX_VALUE` 表示无上限）
- `rate`：税率值（如0.05表示5%）

方法说明：

- 构造函数：初始化税率区间，确保上下限和税率的合法性
- **Getter/Setter**：提供属性访问控制，支持动态调整税率参数
- **toString()**：格式化输出税率信息，自动处理无上限情况

设计考量：

// 示例：无上限的税率区间显示

@Override

```
public String toString() {
    String upperBoundStr = upperBound == Double.MAX_VALUE ? "无上限"
        : String.format("%.2f", upperBound);
    return String.format("区间 [%.2f, %s], 税率 %.1f%%",
        lowerBound, upperBoundStr, rate * 100);
}
```

5.3 TaxCalculator 类

职责：管理税率表和起征点，执行累进税款计算逻辑。

核心属性：

- `threshold`：个税起征点（默认1600元）
- `brackets`：有序的税率区间列表（`List<TaxBracket>`）

关键方法：

- **calculateTax()**:

采用级差累进算法：

```
public double calculateTax(double salary) {
    double taxable = salary - threshold;
    if (taxable <= 0) return 0;

    double tax = 0;
    for (TaxBracket bracket : brackets) {
        if (taxable > bracket.getLowerBound()) {
            double upper = bracket.getUpperBound();
            double delta = Math.min(taxable, upper) - bracket.getLowerBound();
            tax += delta * bracket.getRate();
        }
    }
    return tax;
}
```

- **addBracket()**：维护税率区间顺序，确保区间连续性

设计模式：

- 使用策略模式封装不同税率计算规则
- 开放-封闭原则：通过扩展税率区间实现税制变更

5.4 视图层（View）

5.4.1 TaxView 类

职责：处理所有用户交互，包括菜单显示、输入捕获和结果展示。

核心功能：

- **displayMenu():**

提供清晰的命令行界面：

```
public void displayMenu() {
    System.out.println("\n===== 个人所得税计算器 =====");
    System.out.println("1. 输入工资并计算税额");
    System.out.println("2. 设置起征点");
    System.out.println("3. 修改税率表");
    System.out.println("4. 显示当前税率表");
    System.out.println("5. 退出");
    System.out.print("请选择操作 (1-5) : ");
}
```

- **displayTaxBrackets():**

格式化输出税率表：

```
public void displayTaxBrackets(double threshold, List<TaxBracket> brackets) {
    System.out.println("\n当前税率表 (起征点: " + threshold + "元) :");
    System.out.println("-----");
    System.out.printf("%-5s %-15s %-15s %-10s\n", "级别", "下限", "上限", "税率");
    for (int i = 0; i < brackets.size(); i++) {
        TaxBracket b = brackets.get(i);
        String upper = b.getUpperBound() == Double.MAX_VALUE ? "无上限"
            : String.format("%.2f", b.getUpperBound());
        System.out.printf("%-5d %-15.2f %-15s %-10.1f%%\n",
            i+1, b.getLowerBound(), upper, b.getRate()*100);
    }
}
```

输入验证：

- 使用 `Scanner` 捕获输入
- 通过 `NumberFormatException` 处理非法输入

5.5 控制层（Controller）

5.5.1 TaxController 类

职责：协调视图和模型的交互，处理业务流程。

核心逻辑：

- **run():** 主循环控制：

```
public void run() {
    while (!exit) {
        view.displayMenu();
        String choice = view.getInput();
        switch (choice) {
            case "1": calculateTaxForSalary(); break;
        }
    }
}
```

```

        case "3": modifyTaxBracket(); break;
        case "4": view.displayTaxBrackets(...); break;
        case "5": exit = true; break;
        default: view.displayMessage("无效选项");
    }
}
}

```

- **modifyTaxBracket():**

包含完整的区间验证逻辑:

```

protected void modifyTaxBracket() {
    // 显示当前税率表
    view.displayTaxBrackets(...);

    // 输入验证与边界检查
    int level = Integer.parseInt(view.getInput());
    if (level < 1 || level > brackets.size()) {
        view.displayMessage("无效级别");
        return;
    }

    // 级联更新相邻区间
    updateNextBracketIfNeeded(...);
}

```

设计亮点:

- 依赖注入构造函数支持单元测试
- 级联更新机制保证税率区间连续性
- 输入边界双重验证 (视图层+控制层)

5.6 主程序入口

指责: 调用控制器, 启动应用程序。

```

public class PersonalTaxApp {
    public static void main(String[] args) {
        TaxController controller = new TaxController();
        controller.run();
    }
}

```