个人所得税计算器设计文档

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）

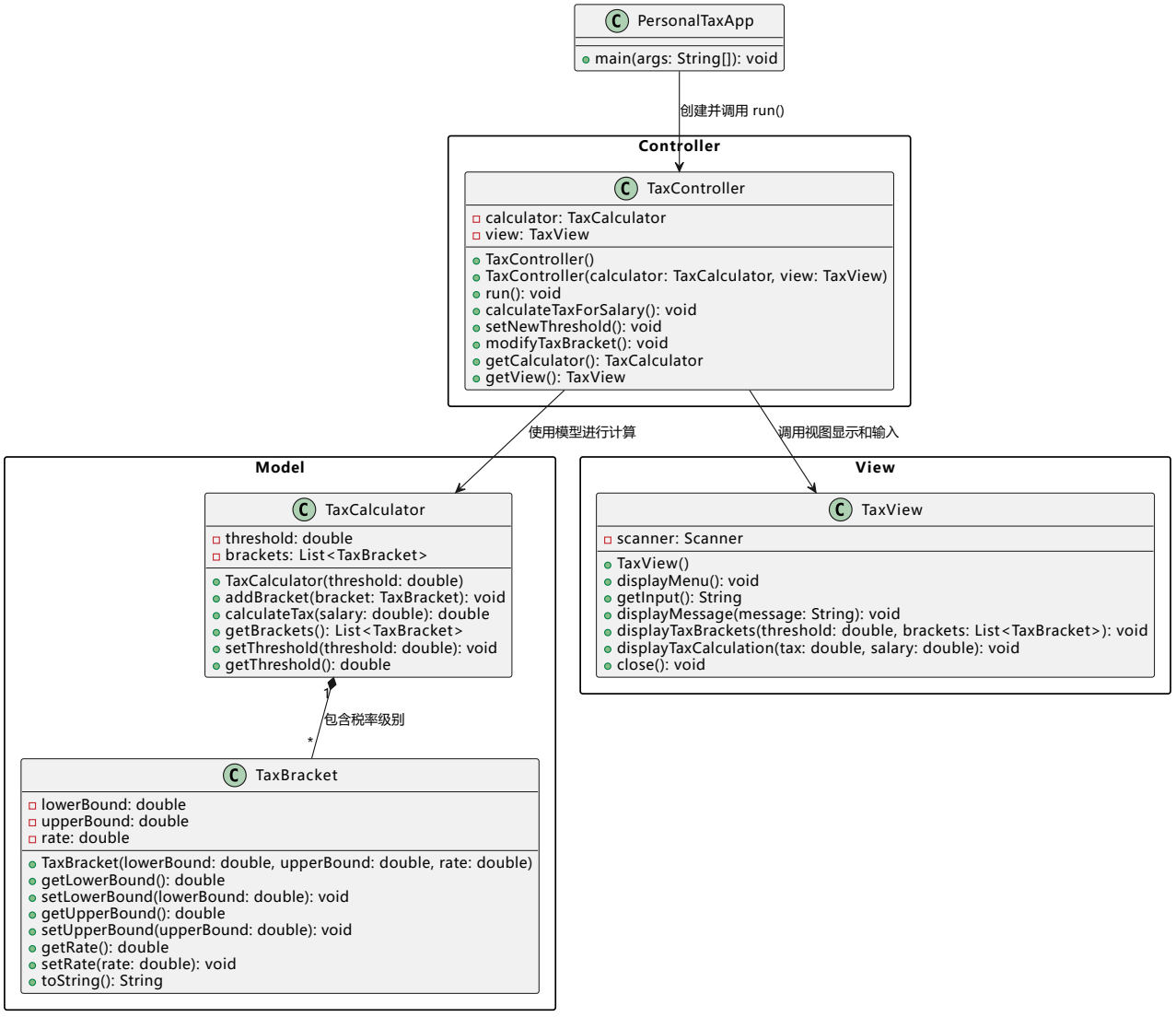
3.1　分层说明

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

3.2　UML类图

下图展示了个人所得税计算器的**MVC架构设计**：

图中箭头表示类之间的依赖关系，例如`TaxController`依赖于`TaxCalculator`和`TaxView`，而`TaxCalculator`包含多个`TaxBracket`对象。通过这种分层设计，系统实现了高内聚低耦合，便于未来扩展和维护。



3.3 模型层（Model）

3.3.1 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

代码片段：

```
// 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);
}
```

3.3.2 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

  ```
  /**
  * 添加一个税率级别
  *
  * @param bracket 要添加的税率级别
  */
  public void addBracket(TaxBracket bracket) {
    brackets.add(bracket);
  }
  ```

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

3.4 视图层（View）

3.4.1 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);
    }
}
```

- **displayTaxCalculation()：**

  展示税款计算结果

```
public void displayTaxCalculation(double tax, double salary) {
  System.out.printf("应缴纳的个人所得税为: %.2f 元\n", tax);
  System.out.printf("税后实际收入: %.2f 元\n", salary - tax);
}
```

3.5 控制层（Controller）

3.5.1 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 "2": setNewThreshold(); break;
            case "3": modifyTaxBracket(); break;
            case "4": view.displayTaxBrackets(...); break;
            case "5": exit = true; break;
            default: view.displayMessage("无效选项");
        }
    }
}
```

- **TaxController 类**

  构造函数，默认构造在内部实例化一个 `TaxCalculator` 和 `TaxView`；带参数的构造函数提供依赖注入便于单元测试

```
// 依赖注入构造函数，便于单元测试注入模拟对象
public TaxController(TaxCalculator calculator, TaxView view) {
    this.calculator = calculator;
    this.view = view;
}

public TaxController() {
    // 初始化税率计算器，默认起征点为 1600 元
    calculator = new TaxCalculator(1600);
    // 添加默认的 5 个税率级别
    calculator.addBracket(new TaxBracket(0, 500, 0.05));
    calculator.addBracket(new TaxBracket(500, 2000, 0.10));
    calculator.addBracket(new TaxBracket(2000, 5000, 0.15));
    calculator.addBracket(new TaxBracket(5000, 20000, 0.20));
    calculator.addBracket(new TaxBracket(20000, Double.MAX_VALUE, 0.25));

    view = new TaxView();
}
```

- **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(...);
}
```

- **calculateTaxForSalary**:

  从视图获取用户输入并调用模型计算税款

```
protected void calculateTaxForSalary() {
    try {
        view.displayMessage("请输入月工资薪金总额：");
        double salary = Double.parseDouble(view.getInput());
        double tax = calculator.calculateTax(salary);
        view.displayTaxCalculation(tax, salary);
    } catch (NumberFormatException e) {
        view.displayMessage("输入格式错误，请输入正确的数字。");
    }
}
```

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）

3.6 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();
    }
}
```

4 测试模块（Test）

4.1 TaxBracketTest 类

职责：
验证 `TaxBracket` 类的正确性，确保税率区间的数据表示、格式化输出等功能符合预期。

说明：

- 重点测试 `toString()` 方法，确保税率区间信息能够正确格式化。
- 覆盖不同上限情况，包括**正常上限**和**无上限**（`Double.MAX_VALUE`）。
- 使用 **JUnit** 进行断言，确保输出与期望结果一致。

测试代码：

```java
@Test
public void testToString_NormalUpperBound() {
    TaxBracket bracket = new TaxBracket(500, 2000, 0.10);
    String expected = "区间 [500.00, 2000.00], 税率 10.0%";
    assertEquals(expected, bracket.toString());
}

@Test
public void testToString_InfiniteUpperBound() {
    TaxBracket bracket = new TaxBracket(20000, Double.MAX_VALUE, 0.25);
    String expected = "区间 [20000.00, 无上限], 税率 25.0%";
    assertEquals(expected, bracket.toString());
}
```

4.2 TaxCalculatorTest 类

职责：
测试 `TaxCalculator` 类的税款计算逻辑，确保其可以正确执行累进税制计算。

说明：

- 测试工资低于起征点时的行为，确保不应缴税款。
- 测试多个税率级别累进计算，验证不同工资下的应纳税额是否正确。
- 通过模拟计算过程，对照实际计算结果与预期值。

测试代码：

```java
@Test
public void testCalculateTax_NoTaxWhenSalaryBelowThreshold() {
    TaxCalculator calculator = new TaxCalculator(1600);
    calculator.addBracket(new TaxBracket(0, 500, 0.05));

    double salary = 1500; // 低于起征点
    double tax = calculator.calculateTax(salary);

    assertEquals(0, tax); // 低于起征点不应缴税
}

@Test
public void testCalculateTax_WithTax() {
    TaxCalculator calculator = new TaxCalculator(1600);
    calculator.addBracket(new TaxBracket(0, 500, 0.05));
    calculator.addBracket(new TaxBracket(500, 2000, 0.10));
    calculator.addBracket(new TaxBracket(2000, 5000, 0.15));
    calculator.addBracket(new TaxBracket(5000, 20000, 0.20));
    calculator.addBracket(new TaxBracket(20000, Double.MAX_VALUE, 0.25));

    double salary = 4000;
    // 应纳税所得额 = 4000 - 1600 = 2400
    // 第一级：0~500，纳税 500 * 0.05 = 25
    // 第二级：500~2000，纳税 1500 * 0.10 = 150
    // 第三级：2000~2400，纳税 400 * 0.15 = 60
    // 总税款 = 25 + 150 + 60 = 235
    double tax = calculator.calculateTax(salary);
```

```
    assertEquals(235, tax, 0.001);
}
```

4.3 TaxControllerTest 类

职责：
测试 `TaxController` 逻辑，确保控制层能正确协调视图和模型的交互。

说明：

- 采用 **Mockito** 模拟 `TaxView`，避免依赖实际用户输入，提高测试效率。
- 测试工资输入与税额计算流程，确保控制器能正确调用模型计算税额，并将结果传递至视图层。
- 验证 `displayTaxCalculation()` 方法是否被正确调用，保证最终输出符合预期。

测试代码：

```
@Test
public void testCalculateTaxForSalary() {
    // 创建模拟的 TaxView
    TaxView mockView = mock(TaxView.class);
    // 设定模拟行为：模拟用户输入 4000 元工资
    when(mockView.getInput()).thenReturn("4000");

    // 创建 TaxCalculator，并添加税率级别
    TaxCalculator calculator = new TaxCalculator(1600);
    calculator.addBracket(new TaxBracket(0, 500, 0.05));
    calculator.addBracket(new TaxBracket(500, 2000, 0.10));
    calculator.addBracket(new TaxBracket(2000, 5000, 0.15));
    calculator.addBracket(new TaxBracket(5000, 20000, 0.20));
    calculator.addBracket(new TaxBracket(20000, Double.MAX_VALUE, 0.25));

    // 构造 TaxController 并传入模拟对象
    TaxController controller = new TaxController(calculator, mockView);
    controller.calculateTaxForSalary();

    // 验证是否正确调用了视图层的方法，显示计算结果
    verify(mockView, atLeastOnce()).displayTaxCalculation(anyDouble(), eq(4000.0));
}
```