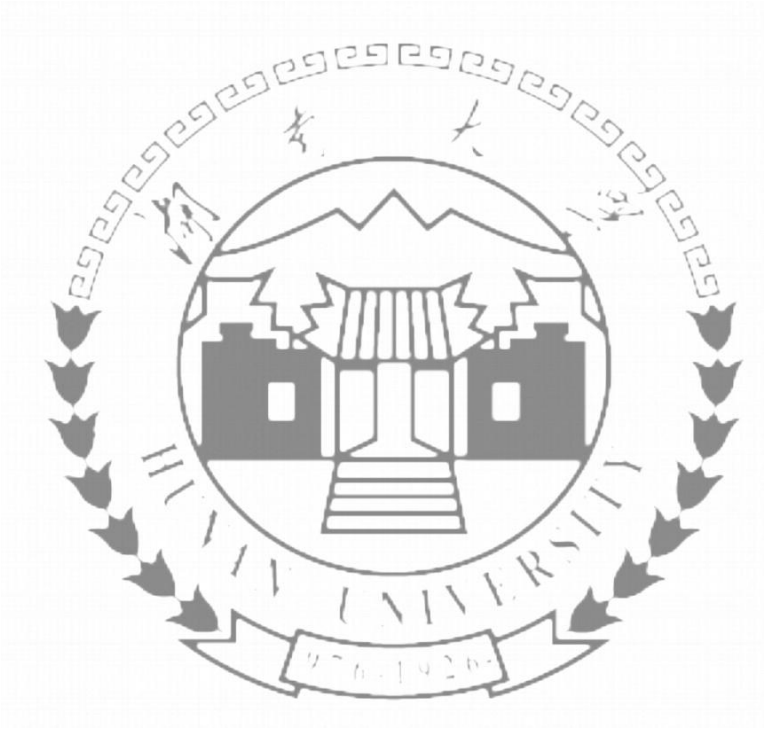
湖南大學



报告名称：	《高级程序设计》Java 租车管理系统实验报告
学生姓名：	郑佳音
学生学号：	202401060101
专业班级：	大数据 2401
学院：	工商管理学院

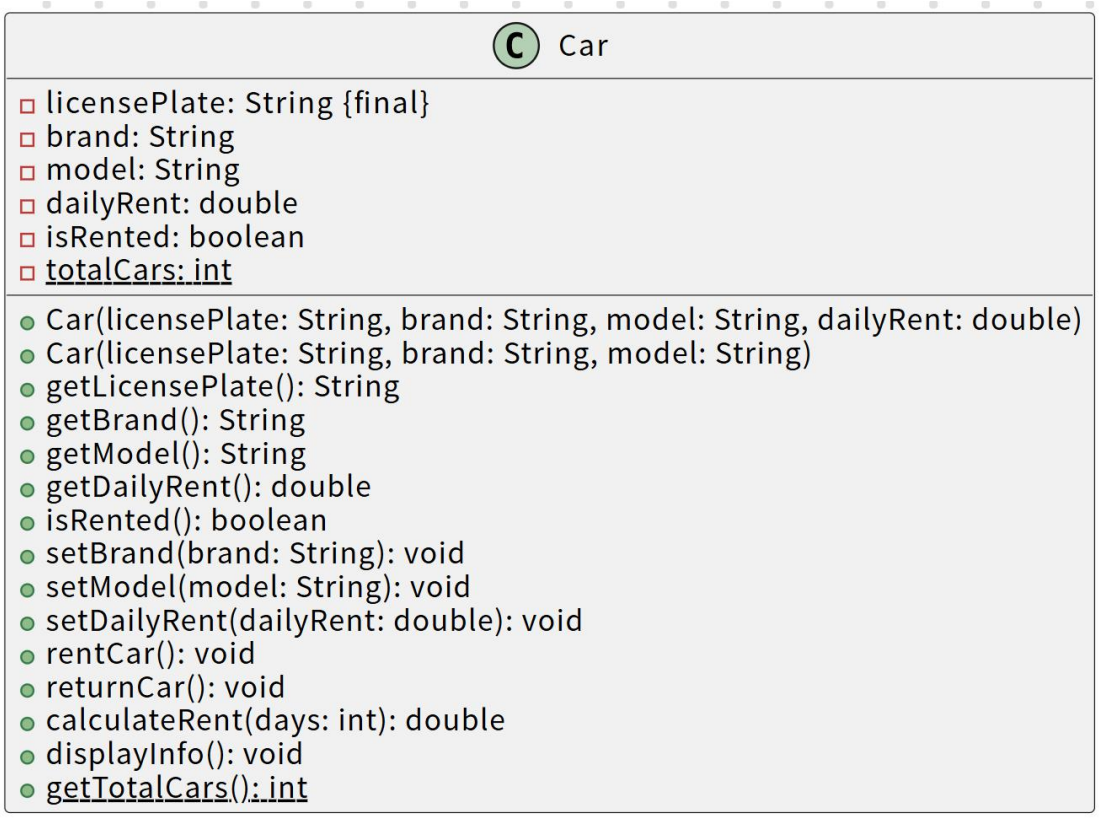
Java 租车管理系统实验报告

1.实验目的

- 学习 Java 类的设计和实现
- 掌握封装的概念和应用
- 学习构造方法的重载和 this() 调用
- 理解静态成员的使用
- 练习基本的业务逻辑实现

2.类图


Car
- licensePlate: String (final)
- brand: String
- model: String
- dailyRent: double
- isRented: boolean
- totalCars: int (static)
+ Car(licensePlate, brand, model, dailyRent)
+ Car(licensePlate, brand, model)
+ getLicensePlate(): String
+ getBrand(): String
+ getModel(): String
+ getDailyRent(): double
+ isRented(): boolean
+ setBrand(brand: String): void
+ setModel(model: String): void
+ setDailyRent(dailyRent: double): void
+ rentCar(): void
+ returnCar(): void
+ calculateRent(days: int): double
+ displayInfo(): void
+ getTotalCars(): int (static)


Car
licensePlate: String {final}
brand: String
model: String
dailyRent: double
isRented: boolean
totalCars: int
Car(licensePlate: String, brand: String, model: String, dailyRent: double)
Car(licensePlate: String, brand: String, model: String)
getLicensePlate(): String
getBrand(): String
getModel(): String
getDailyRent(): double
isRented(): boolean
setBrand(brand: String): void
setModel(model: String): void
setDailyRent(dailyRent: double): void
rentCar(): void
returnCar(): void
calculateRent(days: int): double
displayInfo(): void
getTotalCars(): int

3.核心代码

3.1 构造方法

```
// 全参构造方法
public Car(String licensePlate, String brand, String model, double dailyRent)
{
    this.licensePlate = licensePlate;
    this.brand = brand;
    this.model = model;
    this.dailyRent = dailyRent;
    this.isRented = false;
    totalCars++;
}

// 三参构造方法，使用默认日租金 300 元/天
public Car(String licensePlate, String brand, String model) {
    this(licensePlate, brand, model, 300.0);
}
```

3.2 关键业务方法

```
// 租车方法
public void rentCar() {
    if (isRented) {
        System.out.println("车辆已租出，无法再次租用");
    } else {
        isRented = true;
        System.out.println("车辆租用成功");
    }
}
```

```
// 还车方法
public void returnCar() {
    if (!isRented) {
        System.out.println("车辆未被租用，无需归还");
    } else {
        isRented = false;
        System.out.println("车辆归还成功");
    }
}

// 计算租金方法
public double calculateRent(int days) {
    return dailyRent * days;
}
```

3.3 测试主要片段

```
public class TestCar {
    public static void main(String[] args) {
        // 创建 3 个 Car 对象
        Car car1 = new Car("京A12345", "宝马", "5系", 500.0);
        Car car2 = new Car("京B67890", "奔驰", "C级");
        Car car3 = new Car("京C54321", "奥迪", "A4L", 450.0);

        // 输出所有车辆信息
        System.out.println("所有车辆信息：");
        System.out.println("----------------------");
        car1.displayInfo();
        car2.displayInfo();
        car3.displayInfo();
```

```
        // 测试车辆租用和归还
        System.out.println("测试车辆租用和归还：");
        System.out.println("------------------------");
        System.out.println("测试 car1：");
        car1.rentCar();     // 首次租用
        car1.rentCar();     // 再次租用（应该提示已租出）
        car1.returnCar();   // 归还
        car1.returnCar();   // 再次归还（应该提示未租用）

        // 计算租金
        double rent = car1.calculateRent(5);
        System.out.println("car1 租用 5 天的费用：" + rent + " 元");

        // 测试修改日租金为非法值
        System.out.println("尝试将 car2 的日租金修改为 -100：");
        car2.setDailyRent(-100);
        System.out.println("car2 当前日租金：" + car2.getDailyRent() + " 元
/天");
        System.out.println("尝试将 car2 的日租金修改为 400：");
        car2.setDailyRent(400);
        System.out.println("car2 当前日租金：" + car2.getDailyRent() + " 元
/天");

        // 输出总车辆数
        System.out.println("总车辆数：" + Car.getTotalCars());
    }
}
```

4.运行结果截图

```
所有车辆信息：
------------------------
车牌号：京A12345
品牌：宝马
型号：5系
日租金：500.0 元/天
状态：可租

车牌号：京B67890
品牌：奔驰
型号：C级
日租金：300.0 元/天
状态：可租

车牌号：京C54321
品牌：奥迪
型号：A4L
日租金：450.0 元/天
状态：可租
```

```
测试车辆租用和归还：
------------------------
测试 car1：
车辆租用成功
车辆已租出，无法再次租用
车辆归还成功
车辆未被租用，无需归还

计算租金：
------------------------
car1 租用 5 天的费用：2500.0 元

测试修改日租金：
------------------------
尝试将 car2 的日租金修改为 -100：
日租金必须大于 0，保持原值
car2 当前日租金：300.0 元/天
尝试将 car2 的日租金修改为 400：
car2 当前日租金：400.0 元/天

总车辆数：3
```

5.实验总结

5.1 封装带来的好处

1. 数据安全性 ：通过 private 访问修饰符，确保属性只能通过指定的方法访问和修改，防止外部直接操作数据

2. 代码可维护性 ：当需要修改属性的实现细节时，只需修改对应的方法，而不需要修改所有使用该属性的代码

3. 数据合法性 ：通过 setter 方法中的校验逻辑，确保数据的合法性，例如日租金必须大于 0

4. 接口清晰 ：只暴露必要的方法，隐藏内部实现细节，使类的使用更加简单明了

5.2 遇到的问题及解决方法

1. 构造方法的重载 ：通过使用 this() 调用全参构造方法，减少了代码重复，提高了代码的可维护性

2. 日租金的合法性检查 ：在 setDailyRent() 方法中添加了条件判断，确保

日租金大于 0，否则保持原值并提示错误

3. 车辆状态的管理 ：通过业务方法 rentCar() 和 returnCar() 控制车辆状态的变更，确保状态的一致性，避免了直接修改 isRented 属性可能导致的错误

4. 静态成员的使用 ：使用静态变量 totalCars 统计车辆总数，使用静态方法 getTotalCars() 访问该变量，实现了对车辆总数的全局统计

5.3 实验收获

通过本次实验，我深入理解了 Java 面向对象编程的核心概念，特别是封装的应用。我学会了如何设计一个具有良好封装性的类，如何使用构造方法的重载和 this() 调用，如何使用静态成员，以及如何实现基本的业务逻辑。这些知识将为我后续的 Java 编程学习和实践打下坚实的基础。