湖南大學

HUNAN UNIVERSITY

高级程序设计实验一

实验名称： 实现一个封装完整的实体类

学号： 202401070210

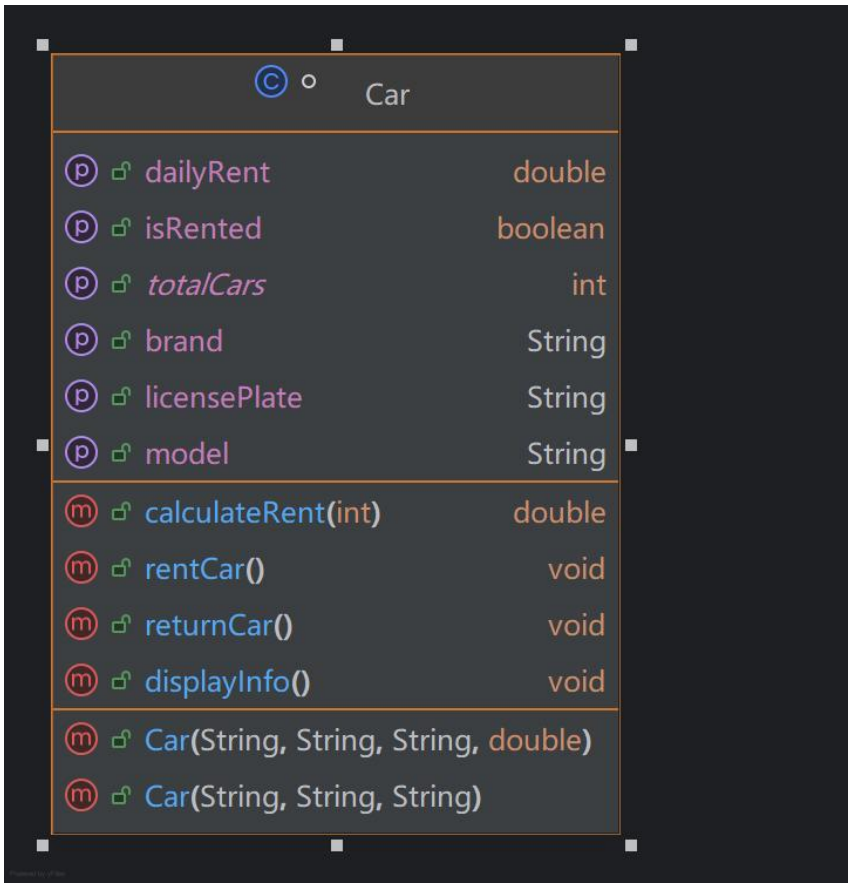
姓名： 郑诗艺

一、实验目的

1. 掌握 Java 面向对象封装特性：通过 private 修饰符封装类的属性，限制外部直接访问，理 解封装对数据安全和代码可维护性的意义。
2. 熟练构造方法编写：实现全参构造方法与重载构造方法，理解 this() 调用本类其他构造方法的语法规则与复用优势。
3. 规范 Getter/Setter 方法设计：根据业务需求为不同属性设计访问器与修改器，掌握属性合法性校验（如日租金必须大于 0）的实现方法。
4. 实现业务逻辑方法：编写租车、还车、租金计算等业务方法，理解状态管理与业务流程控制的实现方式。
5. 理解静态成员的作用：通过静态变量统计车辆总数，掌握静态变量与静态方法的定义、使用场景及生命周期。
6. 完成类的测试与验证：编写测试类验证类功能的正确性，包括正常业务流程、异常操作提示与边界条件处理。

二、类图

该图展示了 Car 类的封装结构：所有属性均为私有（private），通过公共构造方法、业务方法及访问器方法对外提供操作，实现了数据安全与业务逻辑的分离。



三、核心代码

1. Car 类：构造方法

全参构造用于给车辆所有属性赋值，并校验日租金是否合法，不合法则使用默认值 300 元，同时将车辆状态设为未租出，每创建一辆车就让总车辆数 totalCars 加 1。三参构造通过 this()调用全参构造，实现代码复用，默认日租金 300 元。

```java
    // 全参构造
    public Car(String licensePlate, String brand, String model, double dailyRent) {  3 个用法
        this.licensePlate = licensePlate;
        this.brand = brand;
        this.model = model;
        if (dailyRent > 0) {
            this.dailyRent = dailyRent;
        } else {
            this.dailyRent = 300;
            System.out.println("日租金非法，使用默认300元");
        }
        this.isRented = false;
        totalCars++;
    }

    // 三参构造
    public Car(String licensePlate, String brand, String model) {  1 个用法
        this(licensePlate, brand, model,  dailyRent: 300.0);
    }
```

2. Car 类：关键业务方法

(1) rentCar()：判断车辆是否已租，未租则标记为已租出；

(2) returnCar()：判断车辆是否在租，在租则标记为可租；

(3) calculateRent()：根据天数计算总租金；

(4) setDailyRent()：保证日租金必须大于 0，否则不允许修改，保证数据安全。

```java
    public void rentCar() {  2 个用法
        if (isRented) {
            System.out.println("车辆已租出，无法再次租用");
        } else {
            isRented = true;
            System.out.println("租车成功");
        }
    }

    public void returnCar() {  2 个用法
        if (!isRented) {
            System.out.println("车辆未被租用，无需归还");
        } else {
            isRented = false;
            System.out.println("还车成功");
        }
    }

    public double calculateRent(int days) {  1 个用法
        return dailyRent * days;
    }
```

```java
    public void setDailyRent(double dailyRent) {  1 个用法
        if (dailyRent > 0) {
            this.dailyRent = dailyRent;
        } else {
            System.out.println("设置失败：日租金必须大于0");
        }
    }
```

3. TestCar 测试类：主要片段

(1) 测试类通过创建不同构造方法的对象，验证构造方法的正确性。

(2) 调用 displayInfo() 输出车辆信息，验证属性初始化。

(3) 重复调用 rentCar() 和 returnCar()，测试业务流程的异常提示。

(4) 计算 5 天租金，验证租金计算逻辑。

(5) 测试非法日租金修改，验证数据校验逻辑。

(6) 输出静态变量 totalCars，验证对象计数功能。

```
public class TestCar {
    public static void main(String[] args) {
        // 创建3辆车
        Car car1 = new Car( licensePlate: "京A12345",  brand: "丰田",  model: "凯美瑞",  dailyRent: 500.0);
        Car car2 = new Car( licensePlate: "沪B67890",  brand: "大众",  model: "帕萨特");
        Car car3 = new Car( licensePlate: "粤C54321",  brand: "宝马",  model: "3系",  dailyRent: 800.0);

        // 输出车辆信息
        System.out.println("=====车辆信息=====");
        car1.displayInfo();
        car2.displayInfo();
        car3.displayInfo();

        // 测试租车还车
        System.out.println("=====租赁测试=====");
        car1.rentCar();
        car1.rentCar();
        car1.returnCar();
        car1.returnCar();
```

```
        // 计算5天租金
        System.out.println("\n=====5天租金=====");
        double money = car3.calculateRent( days: 5);
        System.out.println("5天总租金：" + money);

        // 测试非法租金
        System.out.println("\n=====测试非法日租金=====");
        car2.setDailyRent(-100);

        // 车辆总数
        System.out.println("\n=====总车辆数=====");
        System.out.println("总车辆数：" + Car.getTotalCars());
    }
}
```

四、运行结果截图

截图完整展示了程序运行后的控制台输出，分为车辆信息初始化、租赁流程测试、租金计算、非法数据校验和车辆总数统计五个部分，全面验证了 Car 类的功能正确性。

1. 车辆信息模块

展示了 3 辆汽车的初始信息：

```
=====车辆信息=====
车牌号：京A12345
品牌：丰田
型号：凯美瑞
日租金：500.0
状态：可租
--------------------
车牌号：沪B67890
品牌：大众
型号：帕萨特
日租金：300.0
状态：可租
--------------------
车牌号：粤C54321
品牌：宝马
型号：3系
日租金：800.0
状态：可租
```

前一辆和第三辆使用全参构造创建，第二辆使用三参构造创建，日租金自动设为默认值 300.0 元，所有车辆初始状态均为 “可租”，符合构造方法设计预期。

2. 租赁测试模块

输出结果：

```
=====租赁测试=====
租车成功
车辆已租出，无法再次租用
还车成功
车辆未被租用，无需归还
```

(1) 第一次调用 rentCar()：车辆状态变为 “已租出”，提示 “租车成功”。

(2) 重复调用 rentCar()：检测到已租出状态，提示 “车辆已租出，无法再次租用”，避免重复租赁。

(3) 第一次调用 returnCar()：状态恢复为 “可租”，提示 “还车成功”。

(4) 重复调用 returnCar()：检测到未租用状态，提示 “车辆未被租用，无需归还”，避免无效操作。

3. 5 天租金计算模块

输出结果：

```
=====5天租金=====
5天总租金：4000.0
```

第三辆车（粤 C54321）日租金为 800.0 元，计算 800.0 × 5 = 4000.0，结果与预期一致，验证了租金计算方法 calculateRent() 的正确性。

4. 非法日租金测试模块

输出结果：

```
=====测试非法日租金=====
设置失败：日租金必须大于0
```

尝试将日租金设置为 -100（非法负值），程序拒绝修改并给出提示，验证了 setDailyRent() 方法的数据合法性校验功能。

5. 总车辆数统计模块

输出结果：

```
=====总车辆数=====
总车辆数：3
```

程序共创建 3 个 Car 对象，静态变量 totalCars 准确统计了对象数量，验证了静态成员计数功能。

五、实验总结

1. 封装带来的好处

(1) 数据安全性提升：将所有属性声明为 private 私有类型，外部无法直接修改，必须通过 setter 或业务方法访问。例如：

(2) 车牌号 licensePlate 只提供 getter，彻底禁止修改；

(3) 日租金 dailyRent 在 setDailyRent() 中做了 >0 校验，避免非法负值；

(4) 租赁状态 isRented 只能通过 rentCar() 和 returnCar() 改变，防止状态错乱。

(5) 代码可维护性增强：内部实现细节被隐藏，外部只需调用公开方法。若后续修改租金计算规则或状态判断逻辑，不会影响调用方代码，降低了耦合度。

(6) 业务逻辑更清晰：将租车、还车、租金计算等操作封装为独立方法，代码可读性更高，也便于复用和扩展。

(7) 对象状态可控：通过封装保证对象始终处于合法状态（如日租金始终为正、租赁状态不会被随意篡改），减少了程序出错的可能。

2. 遇到的问题及解决方法

(1) 构造方法代码冗余

最初编写全参构造和三参构造时，重复写了大量属性赋值代码，维护麻烦。

解决方法：使用 this() 在三参构造中调用全参构造，将默认日租金 300.0 作为参数传入，实现代码复用，简化了构造方法。

(2) 日租金可能被赋值为负数

直接暴露 dailyRent 属性或未做校验的 setter 会导致负数租金，业务逻辑错误。

解决方法：在全参构造和 setDailyRent() 中添加 if (dailyRent > 0) 判断，非法值则保持原值或使用默认值，并输出提示信息，保证数据合法性。

(3) 租赁状态被随意修改

若为 isRented 提供 setter，外部可直接篡改状态，导致重复租车或重复还车。

解决方法：取消 isRented 的 setter，仅提供查询方法 isRented()，状态变更必须通过 rentCar() 和 returnCar() 完成，在方法内部做状态判断和提示。

(4) 静态变量计数不准确

最初在多个构造方法中分别写 totalCars++，容易遗漏或重复计数。

解决方法：将 totalCars++ 统一放在全参构造方法中，三参构造通过 this() 调用全参构造，保证每次创建对象都会准确计数。

3. 实验收获

通过本次实验，我深入理解了 Java 封装的核心思想，掌握了构造方法复用、getter/setter 设计、业务方法封装及静态成员的使用，体会到封装对代码安全性、可维护性和可读性的显著提升，为后续面向对象编程打下了坚实基础。