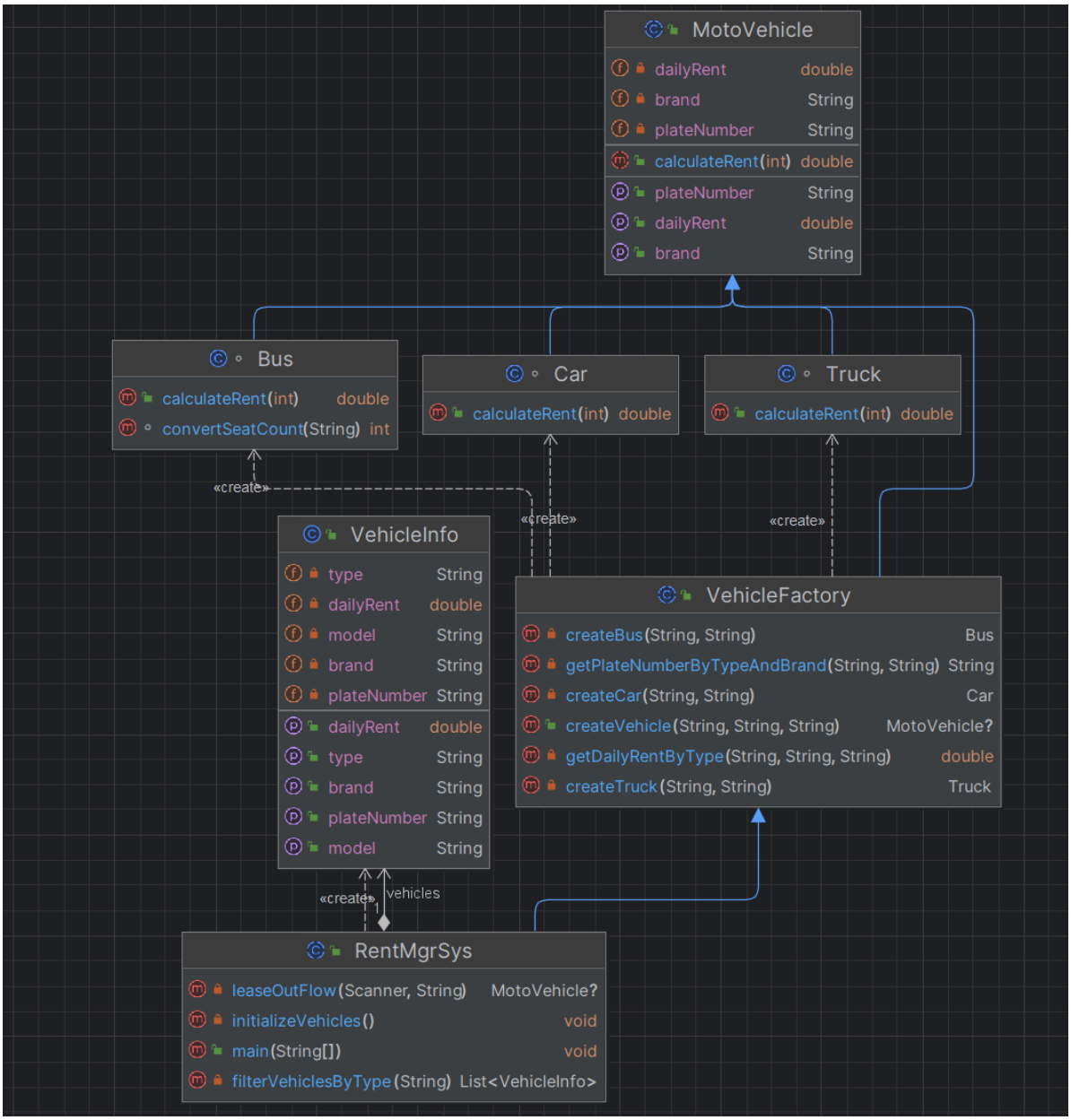
大作业1——汽车租赁系统

21377062 王悦扬

1 UML类设计图



2 车辆信息

- VehicleInfo 类用于存储单个车辆的详细信息。
- 它包含了车辆的类型、品牌、型号、车牌号和日租金等信息。

```
public class VehicleInfo {

    private final String type;            //车辆类型
    private final String brand;           //车辆品牌
    private final String model;           //轿车型号、客车座位数、卡车吨位
    private final String plateNumber;     //车牌号
    private final double dailyRent;       //日租金
```

```
    public VehicleInfo(String type, String brand, String model, String
plateNumber, double dailyRent) {
        this.type = type;
        this.brand = brand;
        this.model = model;
        this.plateNumber = plateNumber;
        this.dailyRent = dailyRent;
    }

    public String getType() {return type;}
    public String getBrand() {return brand;}
    public String getModel() {return model;}
    public String getPlateNumber() {return plateNumber;}
    public double getDailyRent() {return dailyRent;}
}
```

3 汽车基类及其子类

3.1 汽车基类 MotoVehicle类

- MotoVehicle 类是一个抽象基类，定义了汽车的基本属性和功能。

```
public abstract class MotoVehicle {

    private final String plateNumber;  // 车牌号
    private final double dailyRent;    // 每日租金
    private final String brand;        // 轿车型号、客车座位数、卡车吨位

    public MotoVehicle(String plateNumber, double dailyRent, String brand) {
        this.plateNumber = plateNumber;
        this.dailyRent = dailyRent;
        this.brand = brand;
    }

    // 抽象方法：计算租金
    public abstract double calculateRent(int days);

    // Getter 方法
    public String getPlateNumber() { return plateNumber; }
    public double getDailyRent() { return dailyRent; }
    public String getBrand() { return brand; }
}
```

3.2 轿车类 Car类

- Car 类继承自 MotoVehicle 类，代表轿车。

```
class Car extends MotoVehicle {

    public Car(String plateNumber, double dailyRent, String brand, String model)
{
        super(plateNumber, dailyRent, brand);
    }

    //计算租金
    @Override
```

```
    public double calculateRent(int days) {
        double rent = days * getDailyRent();

        if (days > 150) return rent * 0.7;
        else if (days > 30) return rent * 0.8;
        else if (days > 7) return rent * 0.9;
        return rent;
    }
}
```

3.3 客车类 Bus类

- Bus 类继承自 MotoVehicle 类，代表客车。

```
class Bus extends MotoVehicle {

    private static int seatCount;  //客车座位数

    public Bus(String plateNumber, double dailyRent, String brand, String
seatCountStr) {
        super(plateNumber, dailyRent, brand);
        seatCount = convertSeatCount(seatCountStr);
    }

    //计算租金
    @Override
    public double calculateRent(int days) {
        double rent = days * getDailyRent();

        if (days >= 150) return rent * 0.6;
        else if (days >= 30) return rent * 0.7;
        else if (days >= 7) return rent * 0.8;
        else if (days >= 3) return rent * 0.9;
        return rent;
    }

    //座位数有效性检查
    static int convertSeatCount(String seatCountStr) {
        try {
            int seatCount = Integer.parseInt(seatCountStr);
            if (seatCount <= 0) {
                throw new IllegalArgumentException("座位数必须大于0");
            } else if (seatCount <= 16) {
                return 16;
            } else {
                return 34;
            }
        } catch (NumberFormatException e) {
            throw new IllegalArgumentException("座位数必须为有效的整数");
        }
    }
}
```

3.4 卡车类 Truck类

- Truck 类继承自 MotoVehicle 类，代表卡车。

```
class Truck extends MotoVehicle {

    private final double tonnage;  // 载重吨位

    public Truck(String plateNumber, double dailyRent, String brand, double
tonnage) {
        super(plateNumber, dailyRent, brand);
        this.tonnage = tonnage;
    }

    //计算租金
    @Override
    public double calculateRent(int days) {
        double rentPerTon = getDailyRent();
        return tonnage * rentPerTon * days;
    }
}
```

4 静态工厂方法

- VehicleFactory 类提供了创建不同类型的车辆对象的功能。
- 它继承自 MotoVehicle 类，并添加了根据车辆类型创建具体车辆对象的工厂方法。

```
/**
 * VehicleFactory 类提供了创建不同类型的车辆对象的功能。
 * 它继承自 MotoVehicle 类，并添加了根据车辆类型创建具体车辆对象的工厂方法。
 */
public abstract class VehicleFactory extends MotoVehicle {

    public VehicleFactory(String plateNumber, double dailyRent, String brand) {
        super(plateNumber, dailyRent, brand);
    }

    //根据车辆类型、品牌和型号或座位数/吨位创建车辆对象的静态工厂方法
    public static MotoVehicle createVehicle(String type, String brand, String
modelOrSeatsOrTonnage) throws VehicleNotFoundException {
        switch (type) {
            case "car":
                return createCar(brand, modelOrSeatsOrTonnage);
            case "bus":
                return createBus(brand, modelOrSeatsOrTonnage);
            case "truck":
                return createTruck(brand, modelOrSeatsOrTonnage);
            default:
                throw new VehicleNotFoundException("Invalid vehicle type: " +
type);
        }
    }

    private static Car createCar(String brand, String model) throws
VehicleNotFoundException {
        String plateNumber = getPlateNumberByTypeAndBrand("car", brand, model);
```

```
        double dailyRent = getDailyRentByType("car", brand, model);
        return new Car(plateNumber, dailyRent, brand, model);
    }

    private static Bus createBus(String brand, String seatCountStr) throws
VehicleNotFoundException {
        String plateNumber = getPlateNumberByTypeAndBrand("bus", brand,
seatCountStr);
        double dailyRent = getDailyRentByType("bus", brand, seatCountStr);
        return new Bus(plateNumber, dailyRent, brand, seatCountStr);
    }

    private static Truck createTruck(String brand, String tonnage) throws
VehicleNotFoundException {
        String plateNumber = getPlateNumberByTypeAndBrand("truck", brand, "");
        double dailyRent = getDailyRentByType("truck", brand, tonnage);
        return new Truck(plateNumber, dailyRent, brand,
Double.parseDouble(tonnage));
    }

    //根据车辆类型、品牌和型号或座位数/吨位获取日租金。
    private static double getDailyRentByType(String type, String brand, String
modelOrSeatsOrTonnage) throws VehicleNotFoundException {
        int seatCount = type.equalsIgnoreCase("bus") ?
Bus.convertSeatCount(modelOrSeatsOrTonnage) : -1;
        if (type.equalsIgnoreCase("truck")) {
            modelOrSeatsOrTonnage = "";
        }

        for (VehicleInfo vehicle : RentMgrSys.vehicles) {
            if (vehicle.getType().equalsIgnoreCase(type)
                    && vehicle.getBrand().equalsIgnoreCase(brand)
                    && (type.equalsIgnoreCase("bus") ?
vehicle.getModel().equals(String.valueOf(seatCount)) :
vehicle.getModel().equalsIgnoreCase(modelOrSeatsOrTonnage))) {
                return vehicle.getDailyRent();
            }
        }
        throw new VehicleNotFoundException("No vehicle found with type: " + type
+ ", brand: " + brand + ", and model: " + modelOrSeatsOrTonnage);
    }

    //根据车辆类型和品牌获取车牌号。
    private static String getPlateNumberByTypeAndBrand(String type, String
brand, String modelOrSeatsOrTonnage) throws VehicleNotFoundException {
        int seatCount = type.equalsIgnoreCase("bus") ?
Bus.convertSeatCount(modelOrSeatsOrTonnage) : -1;
        if (type.equalsIgnoreCase("truck")) {
            modelOrSeatsOrTonnage = "";
        }

        for (VehicleInfo vehicleInfo : RentMgrSys.vehicles) {
            if (vehicleInfo.getType().equalsIgnoreCase(type)
                    && vehicleInfo.getBrand().equalsIgnoreCase(brand)
                    && (type.equalsIgnoreCase("bus") ?
vehicleInfo.getModel().equals(String.valueOf(seatCount)) :
vehicleInfo.getModel().equalsIgnoreCase(modelOrSeatsOrTonnage))) {
                return vehicleInfo.getPlateNumber();
```

```
            }
        }
        throw new VehicleNotFoundException("No vehicle found with type: " + type
+ ", brand: " + brand + ", and model: " + modelOrSeatsOrTonnage);
    }

    //异常类
    static class VehicleNotFoundException extends Exception {
        public VehicleNotFoundException(String message) {
            super(message);
        }
    }
}
```

5 测试类

- RentMgrSys 类提供了租赁管理系统的核心功能。
- 它继承自 VehicleFactory 类，并添加了车辆信息管理和租赁管理的功能。

```
import java.util.*;

public abstract class RentMgrSys extends VehicleFactory {
    // 存储所有车辆信息的列表
    public static final List<VehicleInfo> vehicles = new ArrayList<>();

    public RentMgrSys(String plateNumber, double dailyRent, String brand) {
        super(plateNumber, dailyRent, brand);
    }

    /**
     * 初始化车辆信息的静态方法。
     * 在系统启动时调用，用于加载车辆的基础信息。
     */
    private static void initializeVehicles() {
        vehicles.add(new VehicleInfo("car", "宝马", "X6", "京NY28588", 800));
        vehicles.add(new VehicleInfo("car", "宝马", "550i", "京CNY3284", 600));
        vehicles.add(new VehicleInfo("car", "别克", "林荫大道", "京NT37465", 300));
        vehicles.add(new VehicleInfo("car", "别克", "GL8", "京NT96968", 600));

        vehicles.add(new VehicleInfo("bus", "金杯", "16", "京6566754", 800));
        vehicles.add(new VehicleInfo("bus", "金杯", "16", "京8696997", 800));
        vehicles.add(new VehicleInfo("bus", "金龙", "34", "京9696996", 1500));
        vehicles.add(new VehicleInfo("bus", "金龙", "34", "京8696998", 1500));

        vehicles.add(new VehicleInfo("truck", "解放", "", "京GD56577", 800));
        vehicles.add(new VehicleInfo("truck", "东风", "", "京GD53456", 700));
    }

    public static void main(String[] args) throws VehicleNotFoundException {
        initializeVehicles();

        Scanner scanner = new Scanner(System.in);
        System.out.println("欢迎使用汽车租赁系统");

        System.out.println("1.轿车 2.客车 3.卡车");
        int vehicleType = scanner.nextInt();
```

```
        MotoVehicle vehicle = null;
        switch (vehicleType) {
            case 1:
                vehicle = leaseOutFlow(scanner, "car");
                break;
            case 2:
                vehicle = leaseOutFlow(scanner, "bus");
                break;
            case 3:
                vehicle = leaseOutFlow(scanner, "truck");
                break;
            default:
                System.out.println("无效的车辆类型！");
                break;
        }

        if (vehicle != null) {
            System.out.println("请输入您要租赁的天数：");
            int days = scanner.nextInt();
            double rent = vehicle.calculateRent(days);
            String plateNumber = vehicle.getPlateNumber();
            System.out.println("分配给您的汽车牌号是" + plateNumber);
            System.out.println("您需要支付的的租赁费用是：" + rent + "元");
        }

        scanner.close();
    }

    // 租车流程方法
    private static MotoVehicle leaseOutFlow(Scanner scanner, String vehicleType)
throws VehicleNotFoundException {
        List<VehicleInfo> vehiclesOfType = filterVehiclesByType(vehicleType);

        System.out.println("请选择你要租赁的" + vehicleType + "品牌：");
        List<String> brands = new ArrayList<>();
        for (VehicleInfo vehicle : vehiclesOfType) {
            if (!brands.contains(vehicle.getBrand())) {
                brands.add(vehicle.getBrand());
            }
        }

        for (int i = 0; i < brands.size(); i++) {
            System.out.println((i + 1) + ". " + brands.get(i));
        }

        int brandChoice = scanner.nextInt();
        String selectedBrand = brands.get(brandChoice - 1);

        switch (vehicleType) {
            case "car":
                System.out.println("请选择你要租赁的" + selectedBrand + "的型号：");
                List<String> models = new ArrayList<>();
                for (VehicleInfo vehicle : vehiclesOfType) {
                    if (vehicle.getBrand().equals(selectedBrand) &&
!models.contains(vehicle.getModel())) {
                        models.add(vehicle.getModel());
                    }
                }
```

```
                for (int i = 0; i < models.size(); i++) {
                    System.out.println((i + 1) + ". " + models.get(i));
                }

                int modelChoice = scanner.nextInt();
                String selectedModel = models.get(modelChoice - 1);

                return VehicleFactory.createVehicle(vehicleType, selectedBrand,
selectedModel);
            case "bus":
                System.out.println("请输入" + selectedBrand + "的座位数：");
                String seatCount = scanner.next();
                return VehicleFactory.createVehicle(vehicleType, selectedBrand,
seatCount);
            case "truck":
                System.out.println("请输入" + selectedBrand + "的吨数：");
                String tonnage = scanner.next();
                return VehicleFactory.createVehicle(vehicleType, selectedBrand,
tonnage);
        }

        return null;
    }

    // 根据车辆类型过滤车辆列表
    private static List<VehicleInfo> filterVehiclesByType(String type) {
        List<VehicleInfo> filteredList = new ArrayList<>();
        for (VehicleInfo vehicle : vehicles) {
            if (vehicle.getType().equals(type)) {
                filteredList.add(vehicle);
            }
        }
        return filteredList;
    }
}
```

6 结果展示

6.1 轿车

- 样例：别克GL8　200天
- 理论输出结果：京NT96968 费用为600 * 200 * 0.7 = 84000

6.2 客车

- 样例：金杯 16座 30天
- 理论输出结果：京6566754 费用为800 * 30 * 0.7 = 16800

```
<terminated> RentMgrSys (1) [Java Application] C:\Program Files\Java\jdk1.8.0_341\bin\javaw.exe  (Nov 23, 2023, 11:25:55 AM – 11:26:00 AM)
欢迎使用汽车租赁系统
1.轿车 2.客车 3.卡车
2
请选择你要租赁的bus品牌：
1. 金杯
2. 金龙
1
请输入金杯的座位数：
16
请输入您要租赁的天数：
30
分配给您的汽车牌号是京6566754
您需要支付的的租赁费用是：16800.0元
```

6.3 卡车

- 样例：东风 20吨 3天
- 理论输出结果：京GD53456 费用为700 * 20 * 3 = 42000

```
<terminated> RentMgrSys (1) [Java Application] C:\Program Files\Java\jdk1.8.0_341\bin\javaw.exe  (Nov 23, 2023, 11:33:01 AM – 11:33:08 AM)
欢迎使用汽车租赁系统
1.轿车 2.客车 3.卡车
3
请选择你要租赁的truck品牌：
1. 解放
2. 东风
2
请输入东风的吨数：
20
请输入您要租赁的天数：
3
分配给您的汽车牌号是京GD53456
您需要支付的的租赁费用是：42000.0元
```

6.4 异常情况

- 样例：金杯 20座
- 理论输出结果：抛出异常 No vehicle found with type: bus, brand: 金杯, and model: 20

```
<terminated> RentMgrSys (1) [Java Application] C:\Program Files\Java\jdk1.8.0_341\bin\javaw.exe  (Nov 23, 2023, 12:19:47 PM – 12:19:55 PM)
欢迎使用汽车租赁系统
1.轿车 2.客车 3.卡车
2
请选择你要租赁的bus品牌：
1. 金杯
2. 金龙
1
请输入金杯的座位数：
20
Exception in thread "main" VehicleFactory$VehicleNotFoundException: No vehicle found with type: bus, brand: 金杯, and model: 20
        at VehicleFactory.getPlateNumberByTypeAndBrand(VehicleFactory.java:74)
        at VehicleFactory.createBus(VehicleFactory.java:32)
        at VehicleFactory.createVehicle(VehicleFactory.java:17)
        at RentMgrSys.leaseOutFlow(RentMgrSys.java:117)
        at RentMgrSys.main(RentMgrSys.java:55)
```

7 技术点总结

7.1 方法/类的封装

- MotoVehicle 类中适用getter方法封装成员变量。

7.2 类的继承

- 子类(Car，Bus，Truck)继承于基类MotoVehicle类。

7.3 多态和抽象方法

- 对于MotoVehicle类和它的子类(Car，Bus，Truck)：MotoVehicle类是一个抽象类，并且有抽象方法（calculateRent），那么在Car，Bus，Truck等子类中对这些抽象方法的实现就是一个抽象方法的应用。这些子类重写了MotoVehicle类的抽象方法，这就体现了多态。
- 对于VehicleFactory类：VehicleFactory 类中有返回MotoVehicle'类型的方法，这些方法根据参数的不同返回不同的子类对象，这属于多态。因为这些不同的子类对象都以被视为“MotoVehicle类型，但它们各自的实现不同。

7.4 工厂模式

- VehicleFactory 类提供了创建不同类型的车辆对象的功能。它继承自 MotoVehicle 类，并添加了根据车辆类型创建具体车辆对象的工厂方法。

7.5 程序注释等编程规范

- 每个类和方法都有明确的注释标明作用。