Refactoring and Strategy Pattern

> Designed by ZHU Yueming in Dec. 15th
>
> This tutorial is based on materials from Dr. Christine Julien.

Manipulation platform

> IDEA version: Ultimate 2023.2.5

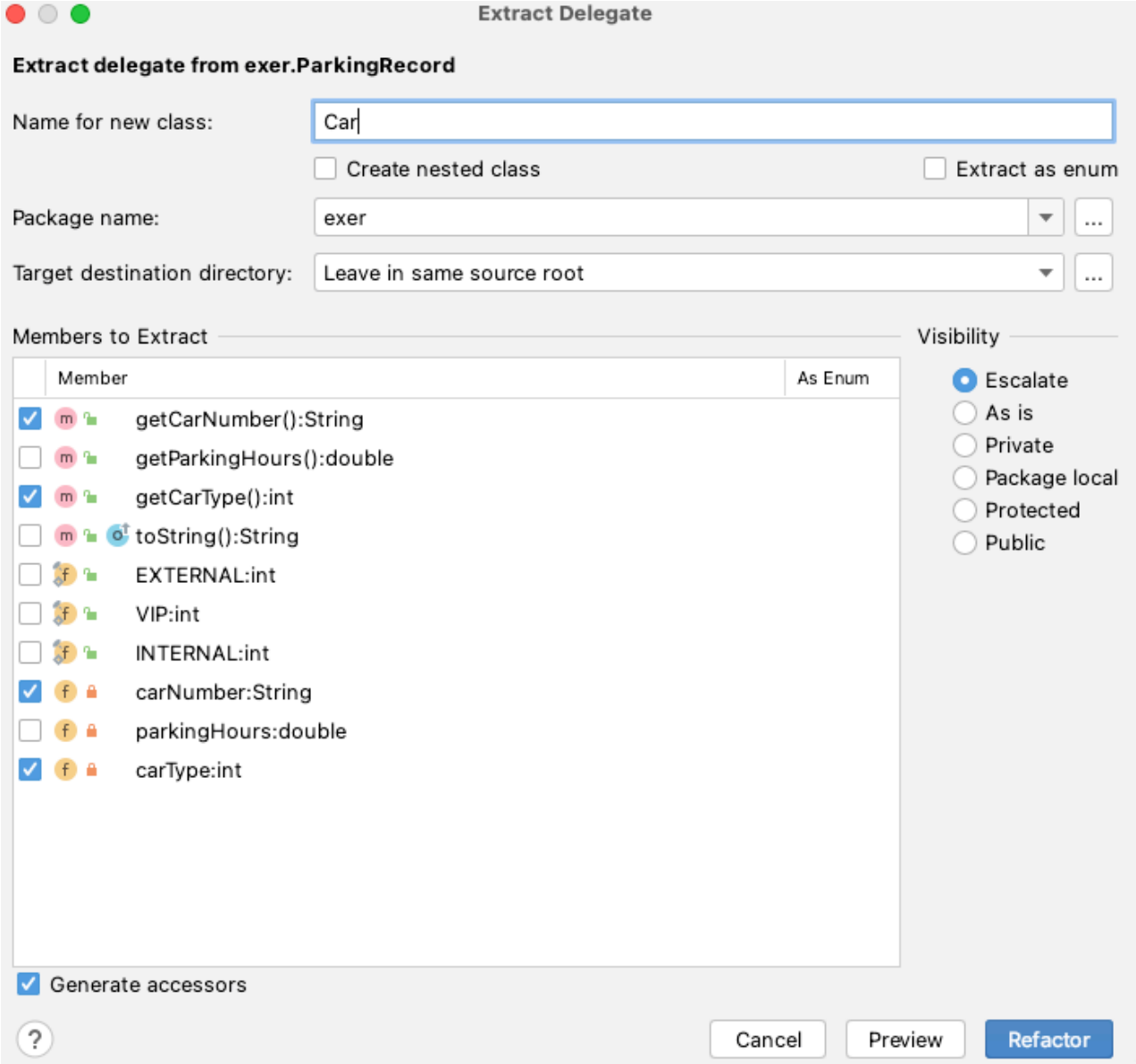
Learning Objective:

- Understand basic refactoring methods including:
  - Extract Class
  - Move Members
  - Change Method Signature
  - Introduce Parameter Object
  - Extract Method / Inline Method
  - Move Method
  - Extract Super Class
  - Push members down / Pull members up
  - Etc.
- Understand Strategy Design Pattern.

Extract Class

In `ParkingRecord` class, there is no need to overly describe the car's information. The best way is to extract some of the attributes and methods related to the car and turn it into a car class.

Open the class `ParkingRecord`, Right click and select “Refactor” > “Extract Delegate”.



Even though there is a problem with the process, you can ignore it and do following works:

- **Remove** following two statements in constructor of `ParkingRecord`

```
this.car.setCarNumber(carNumber);
this.car.setCarType(carType);
```

- **Change** the attribute car to be

```
private Car car;
```

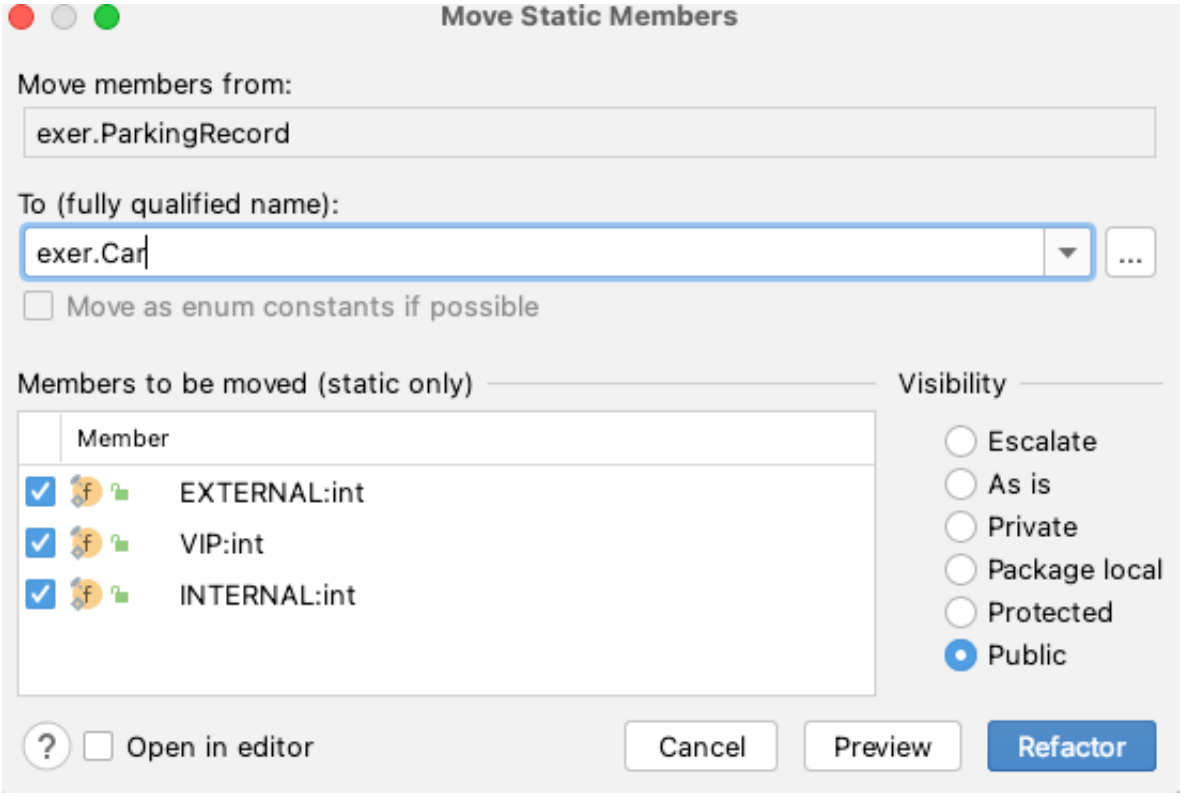
- **Add** a **getter** method of attribute car in `ParkingRecord`

```
public Car getCar() {
        return this.car;
    }
```

Move Field / Move Members

EXTERNAL, VIP, INTERNAL are better to describe `Car`, we'd bettern move them to the `Car` class.

Move the cursor to one of those three variables and right click "Refactor" -> "Move Members"



In this case, the class or Car would be :

```
package exer;

public class Car {
    public static final int EXTERNAL = 0;
    public static final int VIP = 1;
    public static final int INTERNAL = 2;
    String carNumber;
    int carType;

    public Car() {
    }

    public String getCarNumber() {
        return carNumber;
    }

    public int getCarType() {
        return carType;
    }
}
```

Then the method `getCarType()` is useless in `ParkingRecord`, because we have a similar method in `Car`

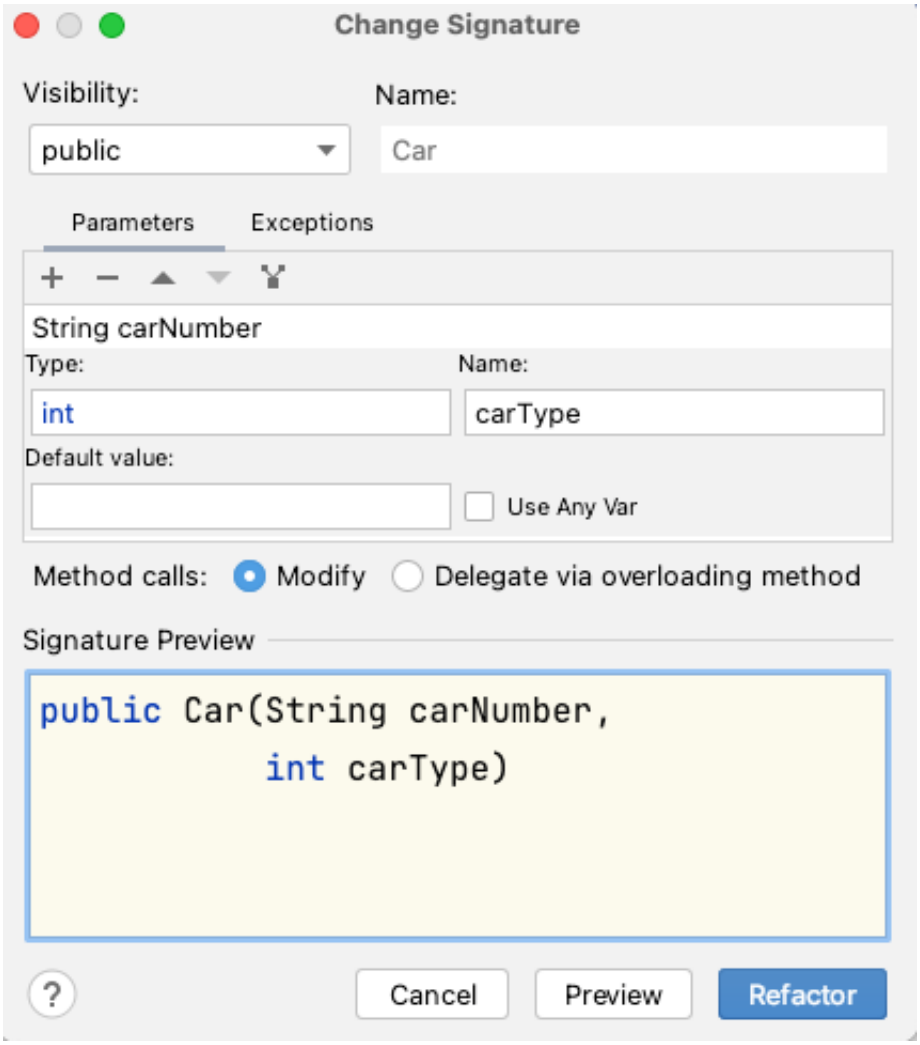
- Remove following code in `ParkingRecord`

```
public int getCarType() {
        return car.getCarType();
    }
```

- Modify all compile error code in `CarOwner`, and change `parkingRecord.getCarType()` to be `car.getCarType()`

Change Method Signature

Open the car class, and we want to set the carNumber and the carType when an object of a car is created. Then we use "Change Method Signature". Move cursor to the constructor of the Car, and then right click "Change Signature"



After that:

- You need complete the constructor of Car(String carNumber, int carType).

```
public Car(String carNumber, int carType) {
        this.carNumber = carNumber;
        this.carType = carType;
    }
```

Exercise 1:

Open the ParkingRecord class, and we want to passing a reference of Car instead of passing two parameters including carNumber and carType. Finish it in a similar way by Change Method Signature about the exercise above.

After that:

- Change the Junit Test to be:

  In the field, add three objects of Car

  ```
  Car c1, c2, c3;
  ```

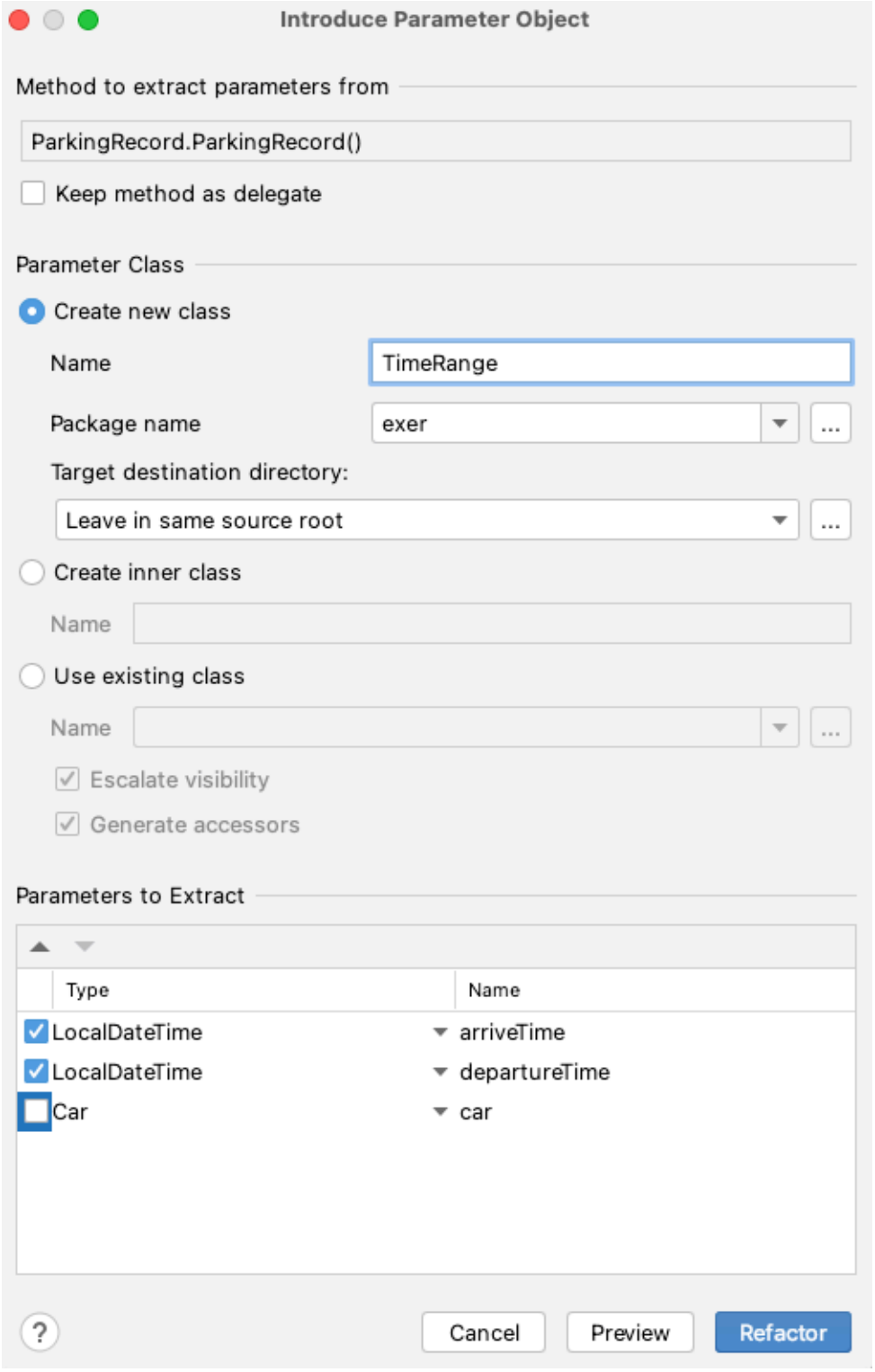
  In the method `public void TestData()` , add and modify by

  ```
        c1 = new Car("粤B11111", 0);
        c2 = new Car("粤B11112", 1);
        c3 = new Car("粤B11113", 2);
        p1 = new ParkingRecord(t1, t2, c1);
        p2 = new ParkingRecord(t5, t6, c1);
        p3 = new ParkingRecord(t7, t10, c1);
        p4 = new ParkingRecord(t2, t3, c2);
        p5 = new ParkingRecord(t5, t6, c2);
        p6 = new ParkingRecord(t8, t10, c2);
        p7 = new ParkingRecord(t3, t4, c3);
        p8 = new ParkingRecord(t7, t8, c3);
        p9 = new ParkingRecord(t9, t10, c3);
  ```

Introduce Parameter Object

In `ParkingRecord` class, move the cursor to the constructor, right click and select "Refactor" -> "Introduce Parameter Object".

It will be success in Intellij IDEA only when we design the exercise in a package.



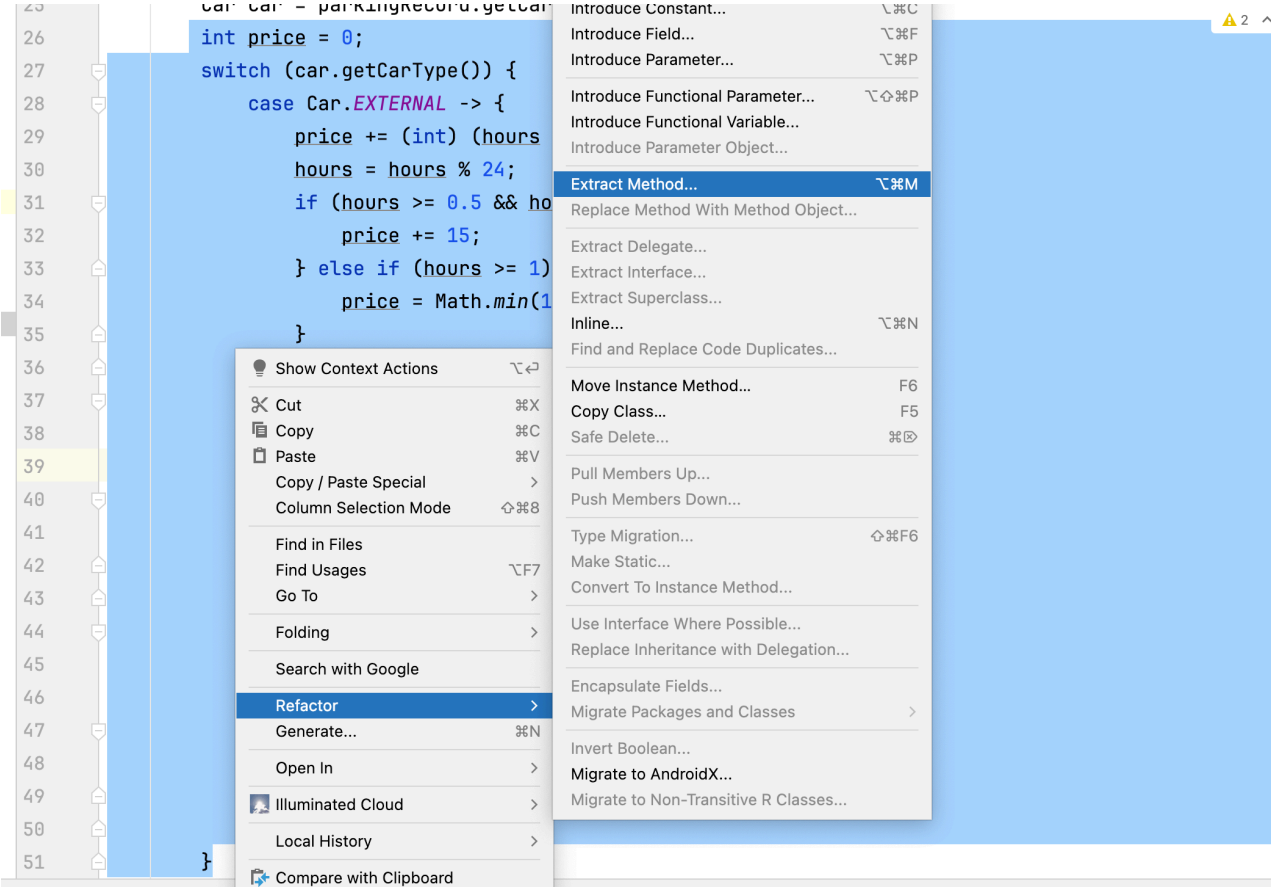
After that, Intellij IDEA will create a new class named `TimeRange` , which is a type of `record`

```
public record TimeRange(LocalDateTime arriveTime, LocalDateTime departureTime)
{
}
```

Extract Method

In `addParkingRecord` method in `CarOwner` class, it is too complex. First of all, all calculation methods about price and active points is related to `Car`. And secondly, it is not suitable to implement two different type of calculate method in only one method. To improve the code, the `addParkingRecord` method is only to record the increment of price and active points, and we can using other method to implement how to calculate the increments.

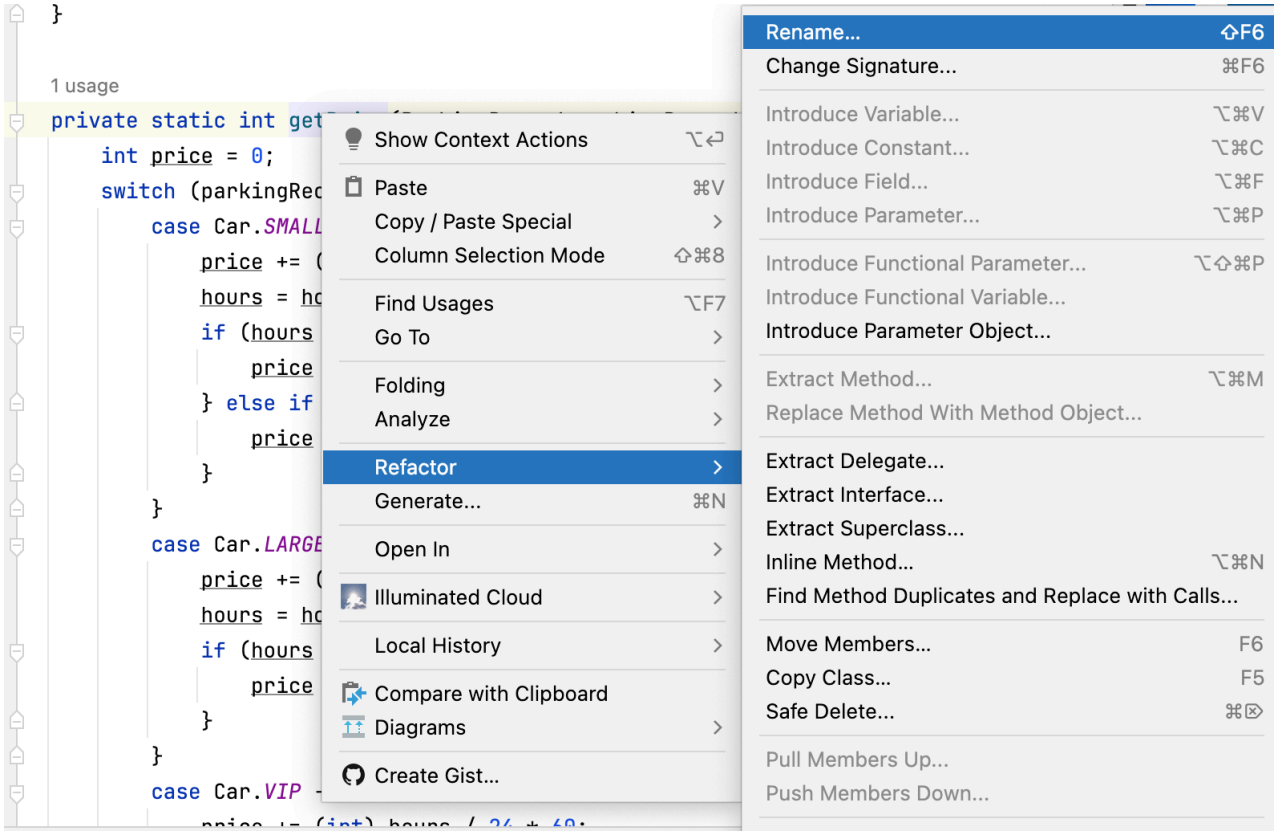
Open the class `CarOwner`, and select the statements lines from 26 to 51, then right click "Refactor" -> "Extract Method"



After that, a static method `private static int getPrice(ParkingRecord parkingRecord, double hours)` will be created, and this method only to calculate the parking price.

Rename

The name of method `getPrice` is the generated name by extract method, we need change it to another one like `parkingPrice` . Move the cursor to the name of `getPrice` method, then right click and select "Refactor" -> "Rename".



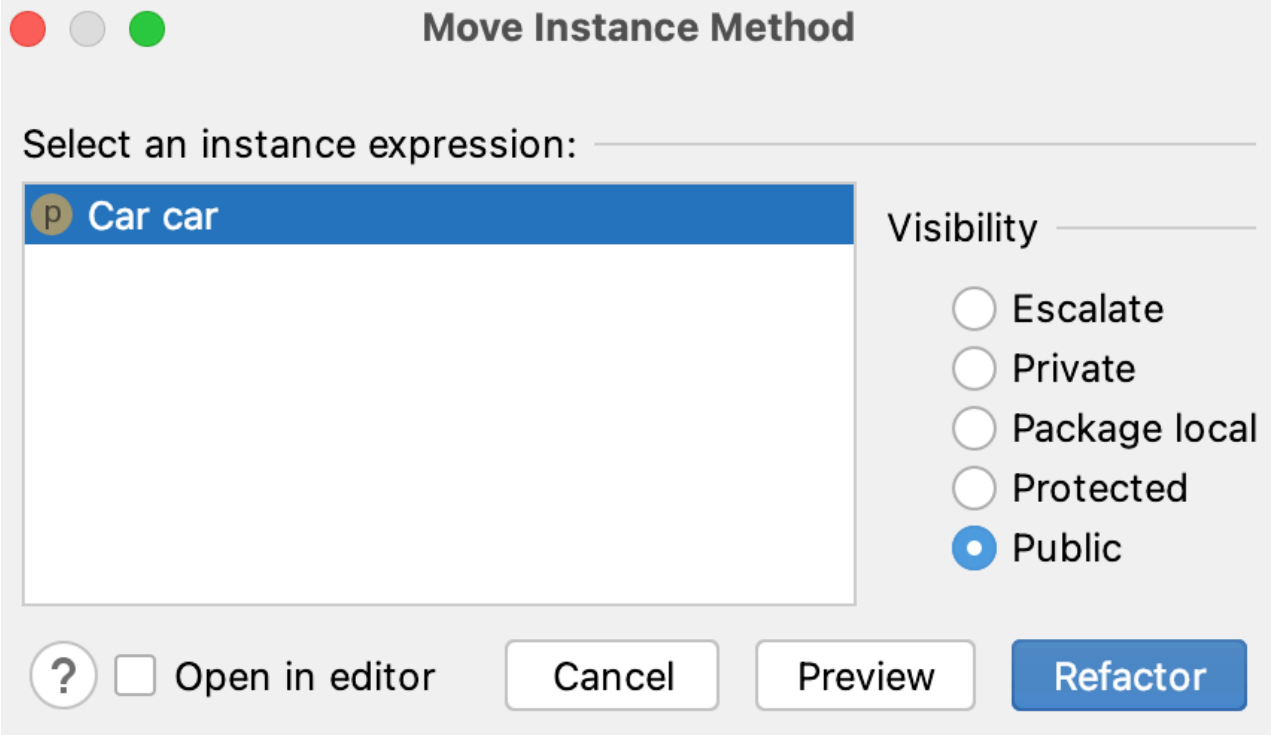
Then change it to `parkingPrice`

Move Method

Before this task, the `static` keyword of `parkingPrice` method is useless, we can remove it. Then the signature of method `parkingPrice` would be:

```
private int parkingPrice(Car car, double hours)
```

The method `parkingPrice` is more appropriately designed in `Car` then in `CarOwner`, so that we need to move it to `Car` . Move the cursor to the name of `parkingPrice` method, and then right click and select "Refactor" -> "Move Instance Method"



After that, the method `parkingPrice` is appeared in `Car`

Exercise 2:

Do similar way to exact a new method from addParkingRecord method in CarOwner class, and give the method a new name `increasePoints`

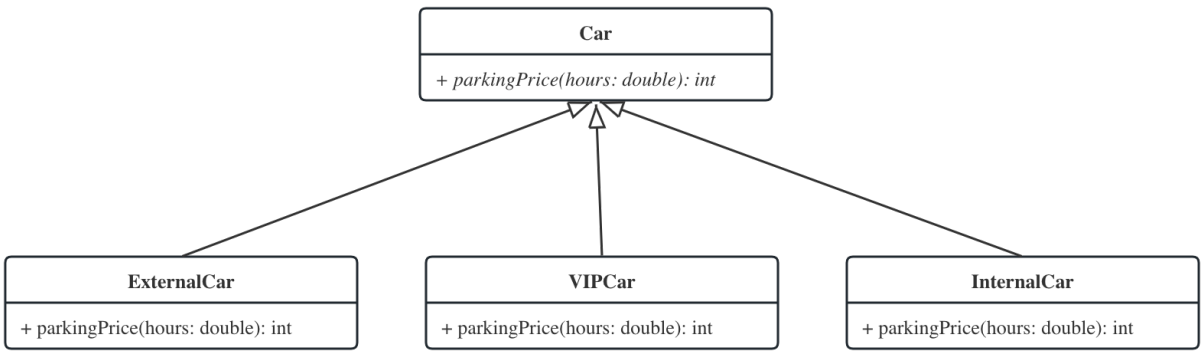
Exercise 3:

Move method `increasePoints` from CarOwner class to `Car` .

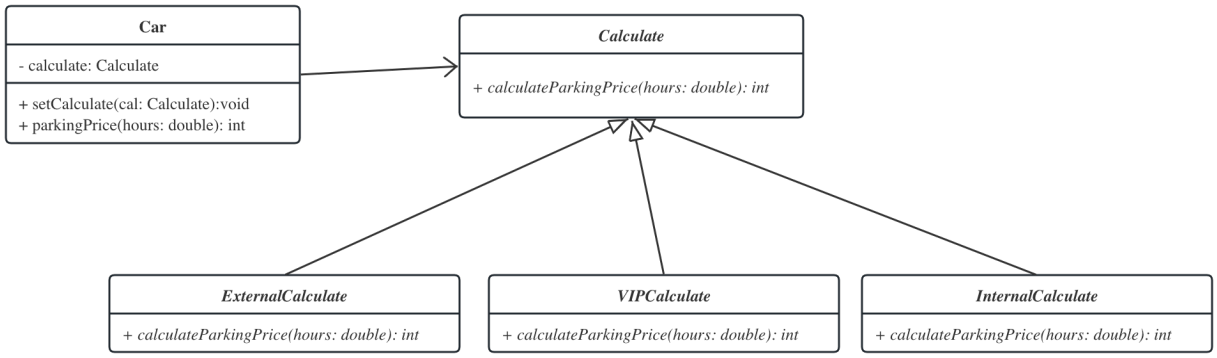
Strategy Design Pattern

In the method `parkingPrice` in `Car` class, there is a bad design, because we strongly couple three different price calculation way in only one method.

To decouple the calculation methods, we can design three subclasses of `Car` and implement the paringPrice method by using polymorphism. An UML diagarm could be designed as below:



This can decouple the switch statement by using polymorphism. But it has one problem: A car can change its classification during its lifetime. For example, a owner can change his/her car from External to Internal, from Internal to VIP. In this case only using polymorphism is not enough, we suggest to use `Strategy Design Pattern`



Refactoring code to strategy Design Pattern

- **Delete and modify**:
  1. those following four parameters:

```
public static final int EXTERNAL = 0;
public static final int VIP = 1;
public static final int INTERNAL = 2;
int carType;
```

  2. `getCarType` method in `Car`

```
public int getCarType() {
        return carType;
    }
```

  3. Change method signature of constructor:

```
public Car(String carNumber) {
        this.carNumber = carNumber;
    }
```

- Add three Calculate classes: `ExternalCalculate` , `VIPCalculate`, `InternalCalculate`.
- Design a method in each Calculate classes, and then split the `parkingPrice` method, put the appropriate statements into the corresponding methods.

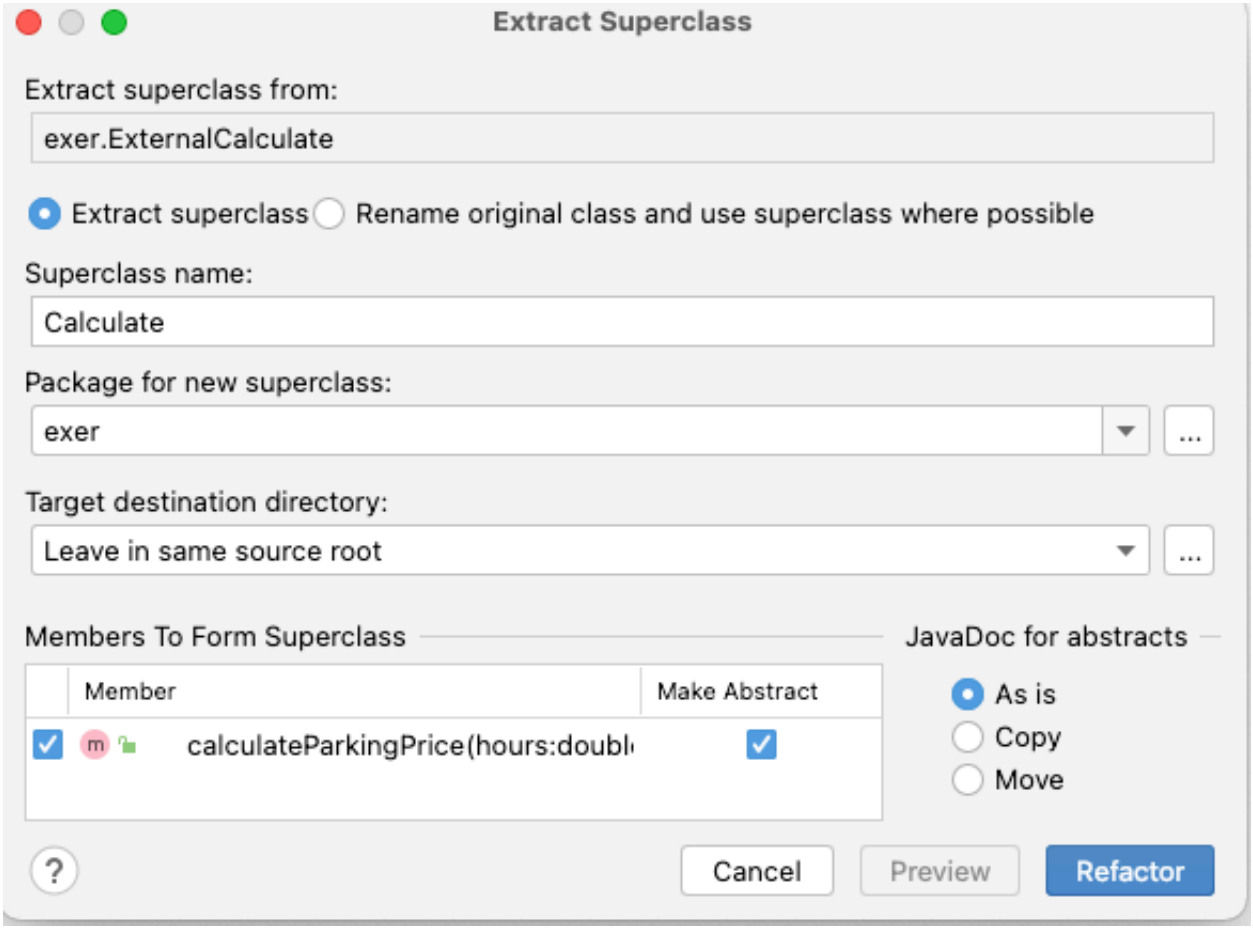
```
public int calculateParkingPrice(double hours){

}
```

- Remove all code in `parkingPrice` method in `Car`

Extract Superclass

After finishing all work above, you can try `Extract Superclass` manipulation. Right click the name of `ExternalCalculate` classe, and select "Refactor" -> "Extract Superclass".



After that, a new class will be created:

```
public abstract class Calculate {
    public abstract int calculateParkingPrice(double hours);
}
```

Complete other code:

1. Let `VIPCalculate` and `InternalCalculate` classes to extands the `Calculate` class.
2. **Exercise 4: Refactor `Car` class to be strategy pattern.**
    - hint 1: Add attribute `Calculate calculate` in `Car`
    - hint 2: Add `setter` method of calcualte field in `Car`
    - Complete other codes.
3. Modify the junit file `MainTest` and change the create Car instance code to be:

```
c1 = new Car("粤B11111");
c2 = new Car("粤B11112");
c3 = new Car("粤B11113");
c1.setCalculate(new ExternalCalculate());
c2.setCalculate(new VIPCalculate());
c3.setCalculate(new InternalCalculate());
```

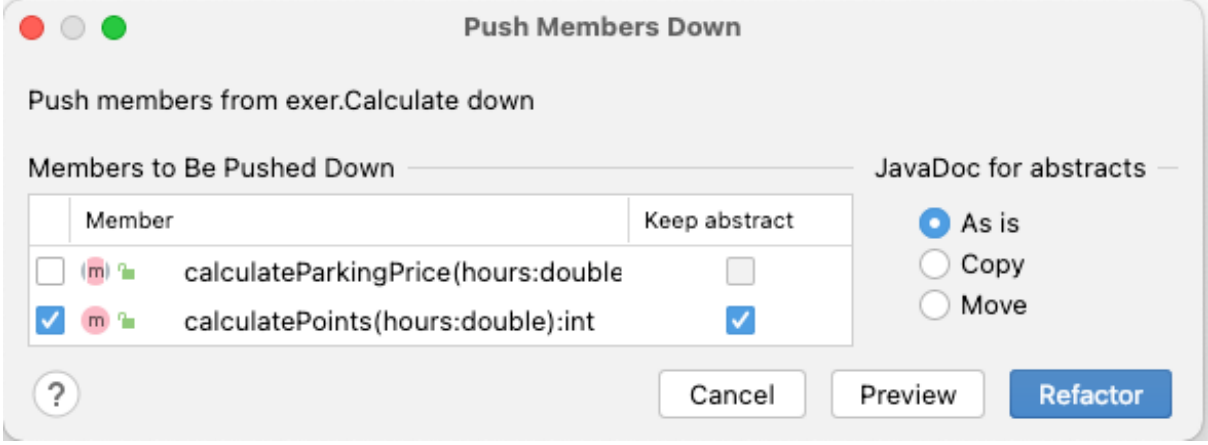
Push members down

1. Create a new method named `calculatePoints` in `Calculate` class

```
public int calculatePoints(double hours) {
        return (int) hours / 24 * 2 + 1;
    }
```

2. In this case, the method of calculatePoints in `VIPCalculate` is different from other sub Calculate classes, so that we can push down the method into subclasses of Calculate classes, and to implement the `calculatePoints` method by it own way.

   Right click `Calculate` class, and select "refactor->Push members down"



   After that, the `Calculate` class would be:

```
public abstract class Calculate {
    public abstract int calculateParkingPrice(double hours);
    public abstract int calculatePoints(double hours);
}
```

3. Modify the `calculatePoints` method in `VIPCalculate` class to be:

```
@Override
    public int calculatePoints(double hours) {
        return ((int) hours / 24 * 2 + 1) * 2;
    }
```

Exercise 5: Then we have added the calculatePoints method in Calculate class, and Refactor `Car` class to be strategy pattern.

Hint: Similar to the method `parkingPrice(double hours)`, modifed the method `increasePoints(double hours)` in `Car` class, and using startegy pattern to calcuate the result.