Documentation

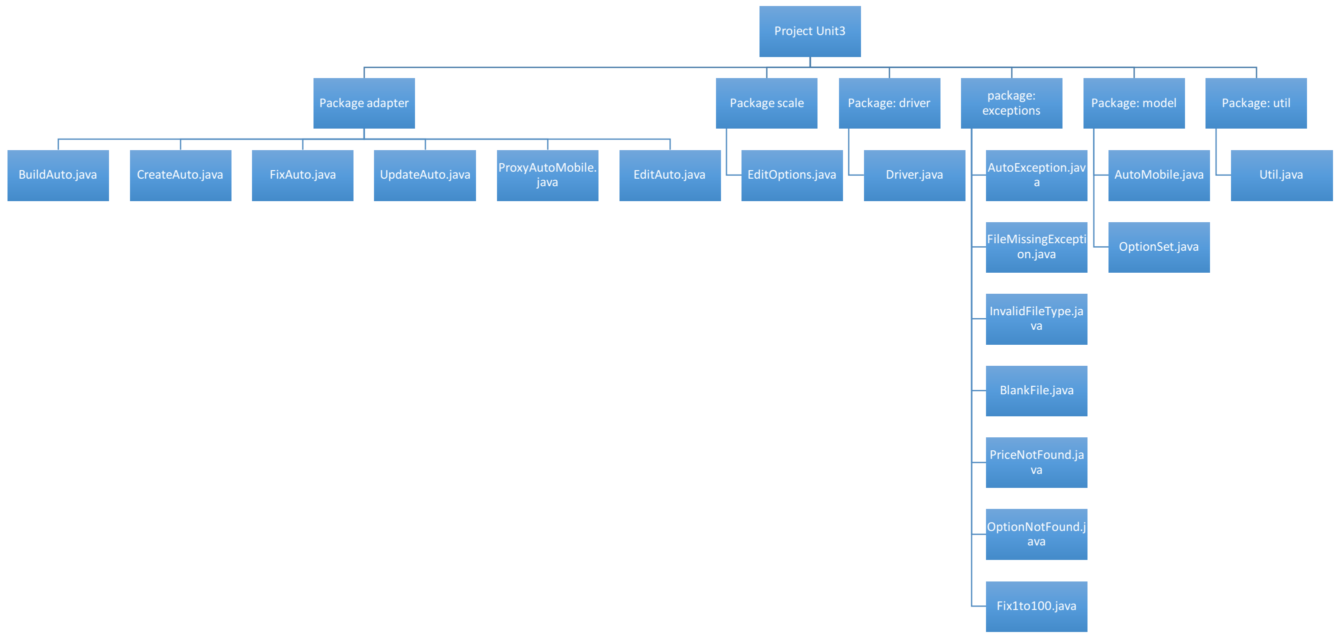
Andrew id: yiq

Attention:

This documentation is for the project 1 unit 3 of the course 18641 in CMU. Design details and test details are included.

1: Design details

In this project, I add scale package and an interfaces based on unit 2-part B. There is a EditOptions.java file in the scale package and this file is for multithread tasking. In package adapter I added an interface called EditAuto.java and there are two methods signature, which are given specific implementation in ProxyAutoMobile.java. So in this way, BuildAuto class and EditOptions class are associated by interface EditAuto. The following diagram is the package and class relationship.



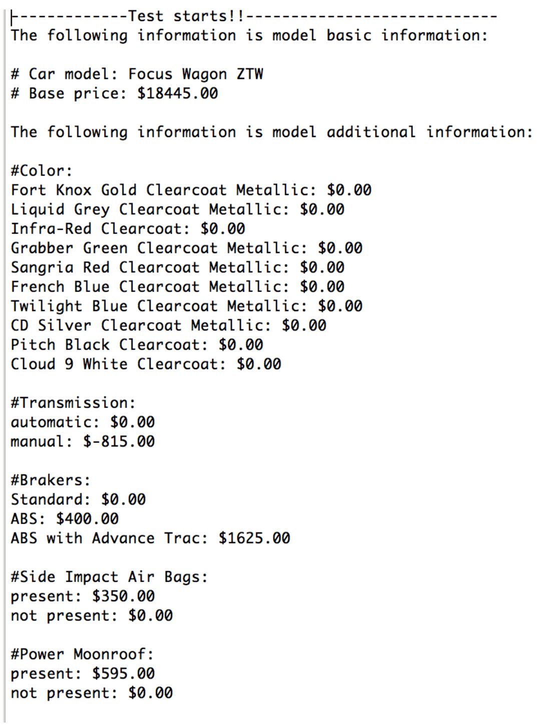
As for the synchronization part, unlike synchronizing each method is AutoMobile class, which is recommended in project handout, I synchronized the whole automobile object and in this way the code can be cleaner and more readable and understandable. Here is the snapshot of the design details:

```
public void updateOptionSetName() {
    synchronized (automobile) {
        waiting();
        automobile.updateOptionSetName(args[0], args[1]);
        printUpdateSetName();
    }
}

public void updateOptionPrice() {
    synchronized (automobile) {
        if(automobile.getOptionPrice(args[0], args[1]) > 300){
            waiting();
            automobile.updateOptionPrice(args[0], args[1], price);
            printUpdateOptionPrice();
        }else {
            System.out.println("The current price is below 300 and can not update.");
        }
    }
}
```

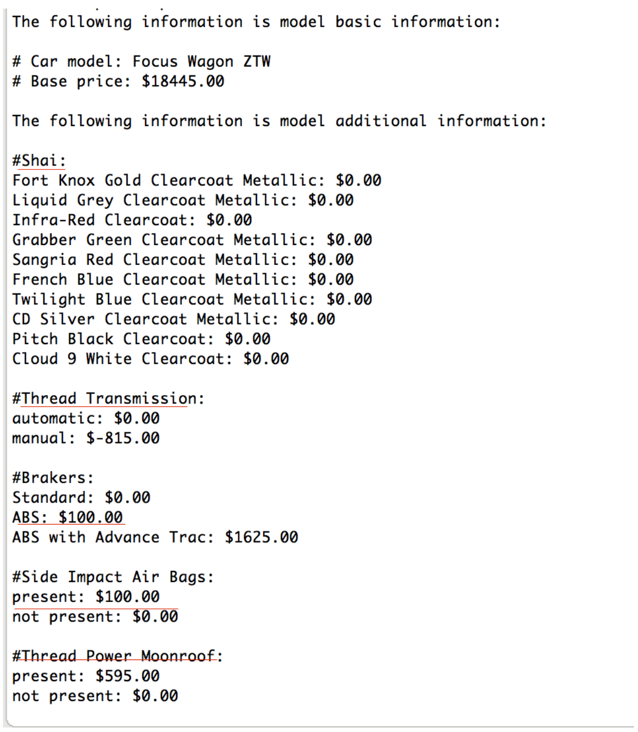
2: Test details:

In this test part, I created six threads to update option set name and option price. In this test, synchronization is used, so the six thread proceed thread by thread and always come in the same order.

First I gave the comparison between origin car information and updated car information:

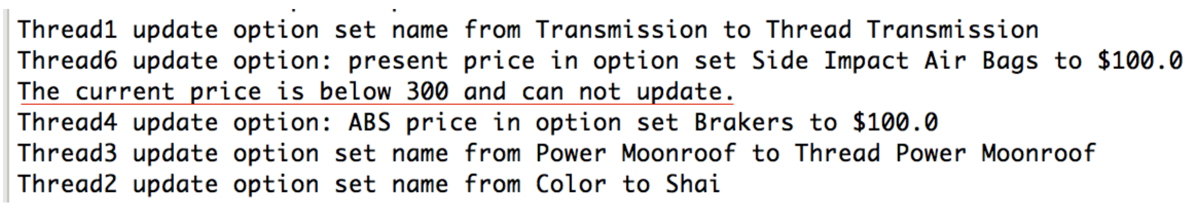


```
-------------Test starts!!----------------------------
The following information is model basic information:

# Car model: Focus Wagon ZTW
# Base price: $18445.00

The following information is model additional information:

#Color:
Fort Knox Gold Clearcoat Metallic: $0.00
Liquid Grey Clearcoat Metallic: $0.00
Infra-Red Clearcoat: $0.00
Grabber Green Clearcoat Metallic: $0.00
Sangria Red Clearcoat Metallic: $0.00
French Blue Clearcoat Metallic: $0.00
Twilight Blue Clearcoat Metallic: $0.00
CD Silver Clearcoat Metallic: $0.00
Pitch Black Clearcoat: $0.00
Cloud 9 White Clearcoat: $0.00

#Transmission:
automatic: $0.00
manual: $-815.00

#Brakers:
Standard: $0.00
ABS: $400.00
ABS with Advance Trac: $1625.00

#Side Impact Air Bags:
present: $350.00
not present: $0.00

#Power Moonroof:
present: $595.00
not present: $0.00
```



```
The following information is model basic information:

# Car model: Focus Wagon ZTW
# Base price: $18445.00

The following information is model additional information:

#Shai:
Fort Knox Gold Clearcoat Metallic: $0.00
Liquid Grey Clearcoat Metallic: $0.00
Infra-Red Clearcoat: $0.00
Grabber Green Clearcoat Metallic: $0.00
Sangria Red Clearcoat Metallic: $0.00
French Blue Clearcoat Metallic: $0.00
Twilight Blue Clearcoat Metallic: $0.00
CD Silver Clearcoat Metallic: $0.00
Pitch Black Clearcoat: $0.00
Cloud 9 White Clearcoat: $0.00

#Thread Transmission:
automatic: $0.00
manual: $-815.00

#Brakers:
Standard: $0.00
ABS: $100.00
ABS with Advance Trac: $1625.00

#Side Impact Air Bags:
present: $100.00
not present: $0.00

#Thread Power Moonroof:
present: $595.00
not present: $0.00
```

Here is the thread update information:



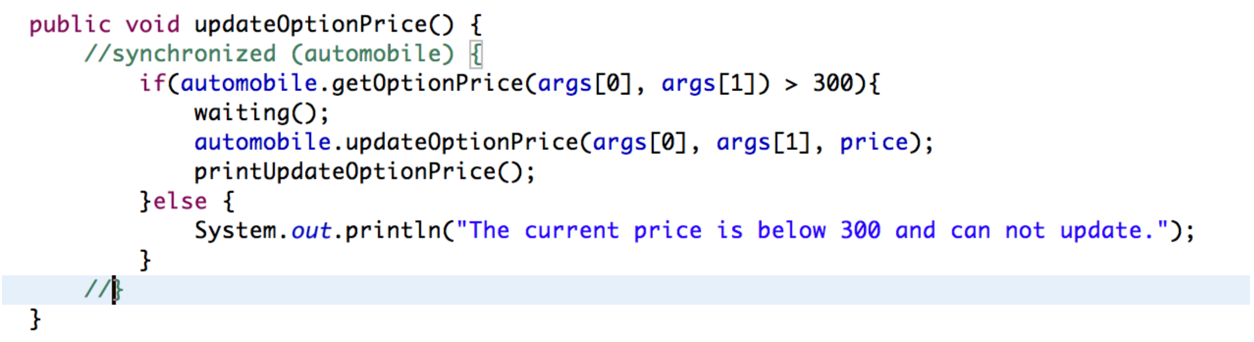
```
Thread1 update option set name from Transmission to Thread Transmission
Thread6 update option: present price in option set Side Impact Air Bags to $100.0
The current price is below 300 and can not update.
Thread4 update option: ABS price in option set Brakers to $100.0
Thread3 update option set name from Power Moonroof to Thread Power Moonroof
Thread2 update option set name from Color to Shai
```

You can find something different in the the update information snapshot, which I have tagged with underline. This is the thread5 update information and with this thread I will explain why should we use synchronization when these threads share the same object.

Data corruption:

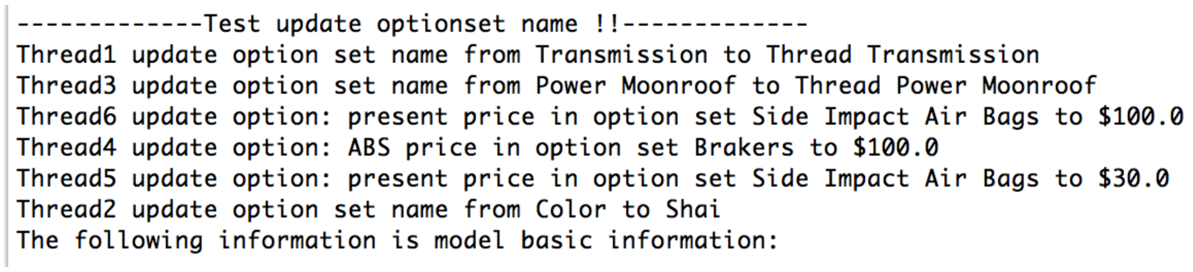
Data corruption happens when different threads share the same data and there is no lock mechanism in the code. In this part, I will demonstrate data corruption occurs without synchronization and prevent data corruption with synchronization. I especially pay attention to the updateOptionPrice() method.

First no synchronization in the updateOptionPrice()



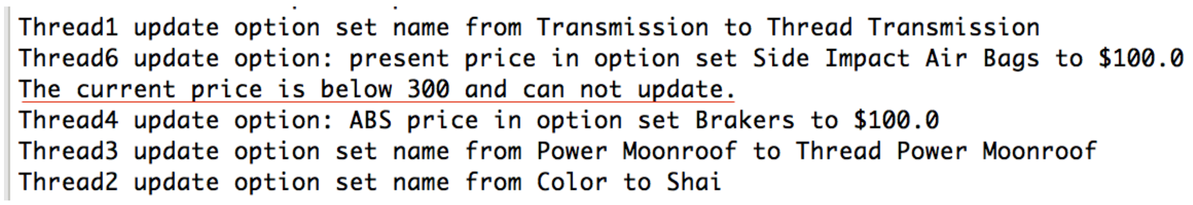
```
public void updateOptionPrice() {
    //synchronized (automobile) {
        if(automobile.getOptionPrice(args[0], args[1]) > 300){
            waiting();
            automobile.updateOptionPrice(args[0], args[1], price);
            printUpdateOptionPrice();
        }else {
            System.out.println("The current price is below 300 and can not update.");
        }
    //}
}
```

Something important should be declared here: Thread 1, thread 2 and thread 3 is to update option set name and thread 4, thread 5 and thread 6 is to update option price. Since only update option price method is tested with no synchronization, so you may get different thread order among thread 4, 5 and 6. But for thread 1, 2 and 3, their relative order is determined since synchronization is used.
And the thread update information: (Without synchronization in updateOptionPrice())



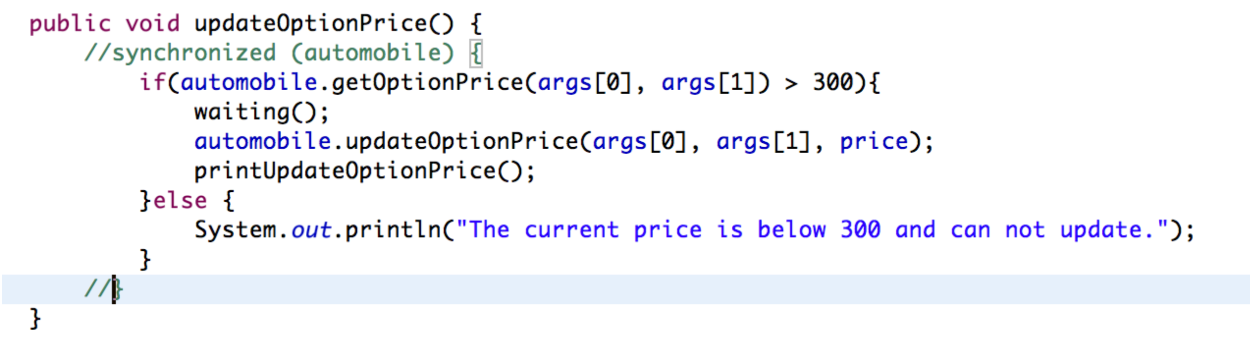
```
-------------Test update optionset name !!-------------
Thread1 update option set name from Transmission to Thread Transmission
Thread3 update option set name from Power Moonroof to Thread Power Moonroof
Thread6 update option: present price in option set Side Impact Air Bags to $100.0
Thread4 update option: ABS price in option set Brakers to $100.0
Thread5 update option: present price in option set Side Impact Air Bags to $30.0
Thread2 update option set name from Color to Shai
The following information is model basic information:
```

Something different from the thread update information with synchronization:



```
Thread1 update option set name from Transmission to Thread Transmission
Thread6 update option: present price in option set Side Impact Air Bags to $100.0
The current price is below 300 and can not update.
Thread4 update option: ABS price in option set Brakers to $100.0
Thread3 update option set name from Power Moonroof to Thread Power Moonroof
Thread2 update option set name from Color to Shai
```

You may find that in the update information without synchronization, thread 5 update the present price in option set Side Impact Air Bags, but in the update information with synchronization, thread 5 can't update the price. Why? Since thread 6 has update the option price to $100. And from the updateOptionPrice() code, we can get that if the current price is lower than $300, no update operation can be done and should print out can-not-update information.



```
public void updateOptionPrice() {
    //synchronized (automobile) {
        if(automobile.getOptionPrice(args[0], args[1]) > 300){
            waiting();
            automobile.updateOptionPrice(args[0], args[1], price);
            printUpdateOptionPrice();
        }else {
            System.out.println("The current price is below 300 and can not update.");
        }
    //}
}
```

Explanation: In the update option price without synchronization, let's assume that thread 5 first acquire the updateOptionPrice() and then do waiting(). Now the thread 6 also acquire the updateOptionPrice() and since thread 5 is waiting and have not updated the option price, the option price is still higher than $300, so thread 6 also get in the method. And this cause data corruption. Vice versa. Whoever can firstly acquire the updateOptionPrice() depends on the JVM.

While in the update option price with synchronization, thread 6 firstly acquire the updateOptionPrice() and then this whole code block is locked. At this time, if thread 5 wants to get in the method, it has to wait until the thread 6 unlock the code block. Once the thread 6 unlock the whole code block, thread 5 can get in but now the price has been updated and is lower than 300, so thread 5 can not update price and just print out can-not-update information.

This example dose not show much usefulness in actual implementation and just aims to demonstrate the synchronization can prevent data corruption.

PS: Since in the submitted code, the method updateOptionPrice() is synchronized and if you want to test data corruption, remember comment synchronized code in method updateOptionPrice() in EditOptions class.