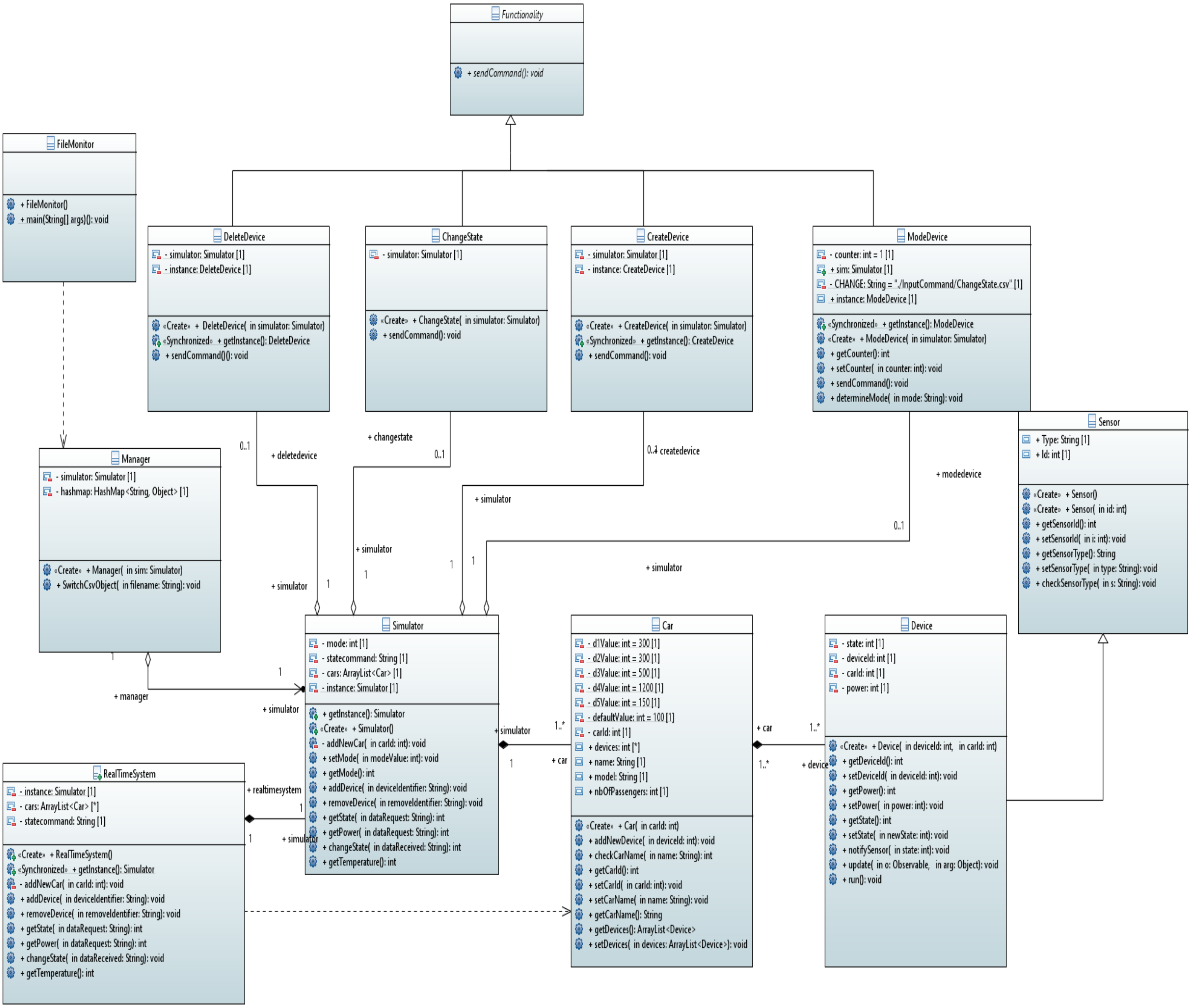

Functionality
+ sendCommand(): void
FileMonitor
+ FileMonitor()
+ main(String[] args)(): void
DeleteDevice
- simulator: Simulator [1]
- instance: DeleteDevice [1]
«Create» + DeleteDevice(in simulator: Simulator)
«Synchronized» + getInstance(): DeleteDevice
+ sendCommand()(): void
ChangeState
- simulator: Simulator [1]
«Create» + ChangeState(in simulator: Simulator)
+ sendCommand(): void
CreateDevice
- simulator: Simulator [1]
- instance: CreateDevice [1]
«Create» + CreateDevice(in simulator: Simulator)
«Synchronized» + getInstance(): CreateDevice
+ sendCommand(): void
ModeDevice
- counter: int = 1 [1]
+ sim: Simulator [1]
- CHANGE: String = "./InputCommand/ChangeState.csv" [1]
+ instance: ModeDevice [1]
«Synchronized» + getInstance(): ModeDevice
«Create» + ModeDevice(in simulator: Simulator)
+ getCounter(): int
+ setCounter(in counter: int): void
+ sendCommand(): void
+ determineMode(in mode: String): void
Sensor
+ Type: String [1]
+ Id: int [1]
«Create» + Sensor()
«Create» + Sensor(in id: int)
+ getSensorId(): int
+ setSensorId(in i: int): void
+ getSensorType(): String
+ setSensorType(in type: String): void
+ checkSensorType(in s: String): void
Manager
- simulator: Simulator [1]
- hashmap: HashMap<String, Object> [1]
«Create» + Manager(in sim: Simulator)
+ SwitchCsvObject(in filename: String): void
0..1
+ deletedevice
+ changestate
0..1
0..1 createdevice
+ modedevice
0..1
+ simulator
+ simulator
+ simulator
+ simulator
1
1
1
1
Simulator
- mode: int [1]
- statecommand: String [1]
- cars: ArrayList<Car> [1]
- instance: Simulator [1]
+ getInstance(): Simulator
«Create» + Simulator()
- addNewCar(in carId: int): void
+ setMode(in modeValue: int): void
+ getMode(): int
+ addDevice(in deviceIdentifier: String): void
+ removeDevice(in removeIdentifier: String): void
+ getState(in dataRequest: String): int
+ getPower(in dataRequest: String): int
+ changeState(in dataReceived: String): int
+ getTemperature(): int
1
1
+ manager
+ simulator
+ simulator
1..*
1
+ car
Car
- d1Value: int = 300 [1]
- d2Value: int = 300 [1]
- d3Value: int = 500 [1]
- d4Value: int = 1200 [1]
- d5Value: int = 150 [1]
- defaultValue: int = 100 [1]
- carId: int [1]
+ devices: int [*]
+ name: String [1]
+ model: String [1]
+ nbOfPassengers: int [1]
«Create» + Car(in carId: int)
+ addNewDevice(in deviceId: int): void
+ checkCarName(in name: String): int
+ getCarId(): int
+ setCarId(in carId: int): void
+ setCarName(in name: String): void
+ getCarName(): String
+ getDevices(): ArrayList<Device>
+ setDevices(in devices: ArrayList<Device>): void
+ car
1..*
1..*
+ device
Device
- state: int [1]
- deviceId: int [1]
- carId: int [1]
- power: int [1]
«Create» + Device(in deviceId: int, in carId: int)
+ getDeviceId(): int
+ setDeviceId(in deviceId: int): void
+ getPower(): int
+ setPower(in power: int): void
+ getState(): int
+ setState(in newState: int): void
+ notifySensor(in state: int): void
+ update(in o: Observable, in arg: Object): void
+ run(): void
RealTimeSystem
- instance: Simulator [1]
- cars: ArrayList<Car> [*]
- statecommand: String [1]
«Create» + RealTimeSystem()
«Synchronized» + getInstance(): Simulator
- addNewCar(in carId: int): void
+ addDevice(in deviceIdentifier: String): void
+ removeDevice(in removeIdentifier: String): void
+ getState(in dataRequest: String): int
+ getPower(in dataRequest: String): int
+ changeState(in dataReceived: String): void
+ getTemperature(): int
+ realtimesystem
1
1
+ simulator

2.

The purpose of this project is to simulate a system that allows users to manage some cars and some devices in each car by changing data in different files. It is about how users can manage different cars or different devices on the same car while others are not be affected. When the data in the file changes, the project will make the corresponding changes automatically such as create devices or change state. In this project, users could change the mode by changing data in Mode.csv file. When data changes in the file create.csv or delete.csv, the project can add or delete the devices in different cars. When the car doesn't exist, the project could create a new car and then add devices. The state of devices can also be changed by changing the data in ChangeState.csv.

3.

1\.

Functionality class, the sendCommand() method

(1) CreateDevice class, the sendCommand() method :override

(2) DeleteDevice class, the sendCommand() method :override

(3) ChangeState class, the sendCommand() method :override

(4) ModeDevice class, the sendCommand() method :override

2\.

(1) Sensor class, the Sensor() method

(2)Sensor class, the Sensor(int id) method :overload

3\.

Device class, the update(Observable o, Object arg) method :override

4\.

Device class, the run() method :override

For the second case, it uses the overload method. When the project is running, it may need different method signatures to achieve different function. In this case, if the project call Sensor() method, it will create an instance with the Id is 0. When the project wants to create a Sensor instance with definite Id, it can achieve it by calling Sensor(int id) method.

4.

The class Observer contains a method called update(Objservable o, Object arg). This method is used for receiving notifications form the notifier to make corresponding changes. And it needs to be overridden in other classes. The Observable class is an interface which contains a Boolean variable called changed and a serious of methods. The Observable object is the object that the project wants to observe and an observable object can have many observers at the same time. When an object is changed, the value of variable changed will be changed. Then the method notifyObservers(Object org) in Observable will call the method updata(Observable o, Object arg) which defined in Observer class to notify observers that the instance has been changed and make the corresponding change. In this project, the observer is device and the status of device is observed. Once the state of device is changed, the project will call notifyObservers(Object org) defined in Observable class and then the method updata(Observable o, Object arg) will be called to notify the observer and make change. By using Observe and Observable class, project can add or delete observer at any point.

Runnable is an interface with only one method called run() and this method has no argument. By implementing Runnable, the project could create many threads to run and the code can be shared by multiple threads. In this project, by using the Runnable interface, the device class can implement Observer and Runnable interfaces at the same time. when add a new device, a new thread will be created for this device. All the threads can share the same code and resources.

5.

Once an attribute is defined as Final Static, it means the value of this attribute can not be changed and it can be accessed by calling its class name. In this project, the power of each device is fixed and is not allowed to be changed. This project uses the Final Static statement to ensure that the value of these attributes cannot be changed.

6.

(1)

ChangeState file represents changing the state of devices.

Create file represents creates devices for a special car. If there no car before, the project will create a new car and then add devices.

Delete file represents delete devices in the car.

Mode file represents changed the mode of car (Auto mode/manual mode)

(2)

In the Create file, the horizontal line represents new device. It has four vertical lines. The first vertical line means the Car ID. The second line means the Device ID. The third line means the state of the device. For the last line, I think it may represents the number of senor that the device have.

(3)

To read the value of these files, the first thing that the project needs to do is open these files. In this project, reader = new BufferedReader(new FileReader("flie path")) is used to open and read file. Then the project uses readline() method to read the value of file line by line and store in tempString variable. As the type of file is .csv, when the project reads a line of data from a file, each data is separated by a comma. So tempString.spilt(",") is used to store each data in an array. When the pointer points the end of the file, readline() will return null and the project will close the file.

7.

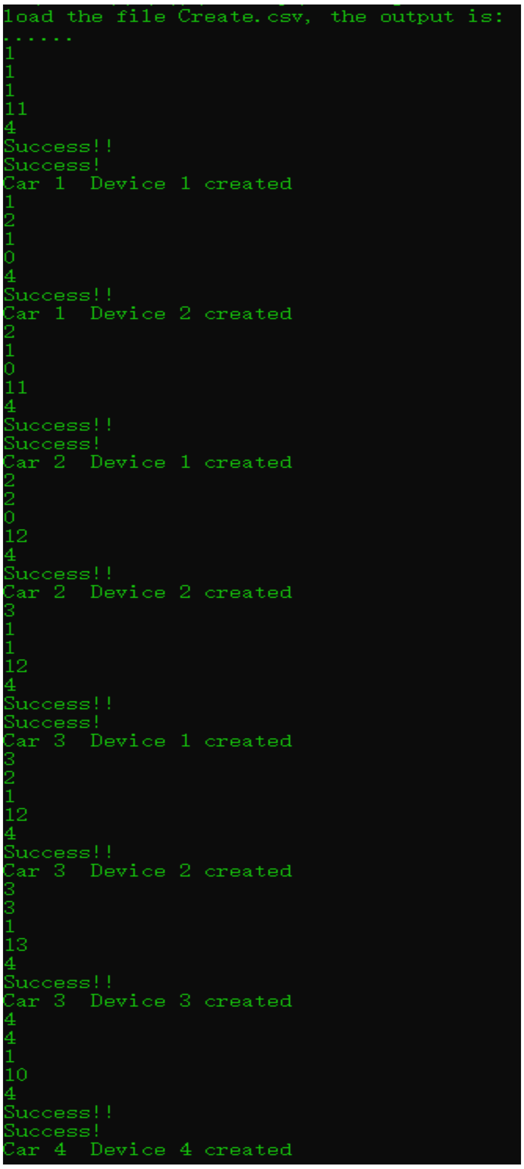
1. The ArrayList<Car> cars in Simulator class. It contains many elements of type Car. The Car type are defined in Car.java file contains many attributes and methods, it also contain another containers. It also be used in RealTimeSystem class
2. The ArrayList<Device> devices in Car class. It contains some Device type elements.

8.

First， I use statement:

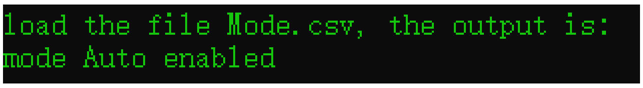
```
System.out.println("load the file Create.csv, the output is: ");
manager.SwitchCsvObject("Create.csv");
```

to load the Create.csv file and run the program and the output as follows:



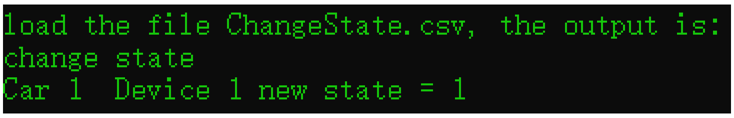
```
load the file Create.csv, the output is:
......
1
1
1
11
4
Success!!
Success!
Car 1  Device 1 created
1
2
1
0
4
Success!!
Car 1  Device 2 created
2
1
0
11
4
Success!!
Success!
Car 2  Device 1 created
2
2
0
12
4
Success!!
Car 2  Device 2 created
3
1
1
12
4
Success!!
Success!
Car 3  Device 1 created
3
2
1
12
4
Success!!
Car 3  Device 2 created
3
3
1
13
4
Success!!
Car 3  Device 3 created
4
4
1
10
4
Success!!
Success!
Car 4  Device 4 created
```

Then, I use code: manager.SwitchCsvObject("Mode.csv"); to load the Mode.csv file and the out put is



```
load the file Mode.csv, the output is:
mode Auto enabled
```

When load and read the file ChangeState.csv, the program will skip the first row and save the data from the second line. Since there is only one row of data in the file, when load it, the program will ignore it. I add some data to the second row as 1, 1, 1 and then, when using code: manager.SwitchCsvObject("ChangeState.csv"); the output as follows:



```
load the file ChangeState.csv, the output is:
change state
Car 1  Device 1 new state = 1
```

When load Delete file, there is no output.