Software Maintenance coursework 1

Report

Yiming Li 20031525

1.

```
System.out.println("Load Create.csv");
manager.SwitchCsvObject("Create.csv");
```

1) The above code is added to the main method in FileMonitor to load Create.csv.

The output is

```
Load Create.csv
......
1
1
1
11
4
Success!!
Success!
Car 1  Device 1 created
1
2
1
00
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
4
4
1
10
4
Success!!
Success!
Car 4  Device 4 created
```

2) Similarly, load Mode.csv.

```
System.out.println("Load Mode.csv");
manager.SwitchCsvObject("Mode.csv");
```

The output is

```
Load Mode.csv
mode Auto enabled
```

3) Load ChangeState.csv. As first line of ChangeState.csv will not have output, I added 2,1,1 as the second line.

```
System.out.println("Load ChangeState.csv");
manager.SwitchCsvObject("Create.csv");
```

The output is

```
....
change state
Car 2  Device 1 new state = 1
!!!!
```

4) Load Delete.csv

```
System.out.println("Load Delete.csv");
manager.SwitchCsvObject("Delete.csv");
```

There's no output.

The remaining code in try and catch statement is used to watching whether any file is changed or not. If a file is changed, it will invoke the *sendCommand()* method in typical file class.

However, there is still some problems about remaining codes. First, if anything is changed before any object is created, the exception will be thrown and the program will be terminated. Secondly, it has a lot of redundant code which is not being used in the whole program, such as the class realTimeSystem, it is not used. Also, the refactoring is not applied, such as Create class.

(a). Create.csv is used to create devices for wanted cars. If the car does not exist, it will also create the car.

Mode.csv is used to determine the mode of car.

ChangeState.csv is used to change the state of devices.

Delete.csv is used to delete object when user want to delete some devices.

(b). In the Create file, every line represents a new device. It is in the form of x,x,x,x. The first number represents the Car ID. The second number represents the Device ID. The third number represents the state of the device. The last number represents the number of senor that the device has.

(c). 1). Use *reader = new BufferedReader(new FileReader("flie path"))* to open the file.

2). Use *reader.readline()* in while statement to read the file line by line until there is no more lines.

3). Use *tempString.spilt(",")* to separate every value by comma and store them in an array.

(319 words including codes)

2.

This project is aimed to simulate a real time system for user to manage devices of cars. User can create devices, delete devices and modify the state or the mode of cars by changing the content in corresponding files.

In the main function, a watching method is used to watch the changes in the files. Users can add or delete the devices in different cars by changing create.csv or delete.csv. Users can change the mode by changing mode.csv. User can also change the state of cars by changing changeState.csv.

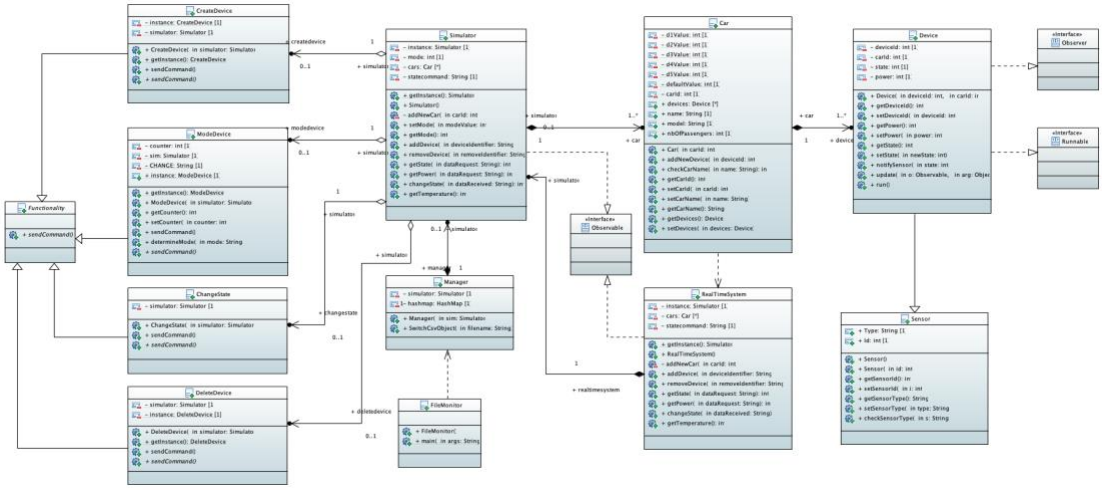
a) Observable and runnable are both interfaces. Interface usually uses an action name (ending in "able") which have no state and are only about method actions. A class that implements an interface must provide implementations for all the methods in the interface. So, in the device class, *update()* in observer and *run()* in runnable are overridden.

b) A class can implement the Observer interface when it wants to be informed of changes in observable objects. The method *update()* is called whenever the observed object is changed. An application calls an Observable object's *notifyObservers* method to have all the object's observers notified of the change. The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called *run()*. When an object implementing interface Runnable is used to create a thread, starting the thread causes the object's *run* method to be called in that separately executing thread.

(251 words)

3.

Class diagram



3(a) All the methods are in the third block of the class.

3(b) All the attributes are in the second block of the class, where the first one is the class name

3(c) All the following relationships are inheritance:

1. Observer, Observable and Runnable are three interfaces, which use the **realization** as the relationship between it and its child class.
2. Functionality is an abstract class which is implemented by CreateDevice, ModeDevice, ChangeState and DeleteDevice classes. The relationship between them is **generalization**.
3. Sensor is the parent class of Device, so the relationship between them is **generalization**.

3(d) All the following relationships are Aggregation:

1. CreateDevice, ModeDevice, ChangeState and DeleteDevice classes all have a constructor of Simulator, so the relationship between them is **Aggregation**.

3(e) In the class diagram, all the classes, attributes and methods, with the sign of + before it, are defined as public, while all the classes, attributes and methods, with the sign of – before it, are defined as private.

4.

4.1 (a). Override:

1. Abstract class: *sendCommand()* in Functionality class

   CreateDevice class, override the sendCommand() method

   DeleteDevice class, override the sendCommand() method

   ChangeState class, override the sendCommand() method

   ModeDevice class, override the sendCommand() method

2. Interface: *update()* in Observer, *run()* in Runnable

   Device class, override the update() method

   Device class, override the run() method

Overload:

1. Sensor class:

   Sensor()

   Sensor(int)

(b). There are two types of polymorphism in Java: compile-time polymorphism (overload) and runtime polymorphism (override).

For overload polymorphism, the method names are the same and the numbers of parameters are different. They also should be in the same class. Like Sensor() and Sensor(int) in Sensor class.

For override polymorphism, the method name should be the same with the method in the parent class and the current class should be a child class.

4.2

The ArrayList<Car> cars in Simulator class. It contains elements of type Car.

The ArrayList<Car> cars in realTimeSystem class. It contains elements of type Car.

The ArrayList<Device> devices in Car class. It contains elements of type Device.

The HashMap<" file name", Object> in Manager class. It contains the key, which is filename and value, which is mode.

4.3

Final Static means the value of this attribute cannot be changed and it can be accessed by calling the 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.

(225 words)

(998 words in total)