2015

Bottle Recycling Machine

IMPLEMENTATION GUIDE

VEIJO VÄISÄNEN

Implementation Guide: Bottle Recycling Machine

1. General

This document is guide to implement Bottle Recycling Machine simulation application based on object oriented design with UML language. See also example implementation of Drink Wending Machine application explained in separate document.

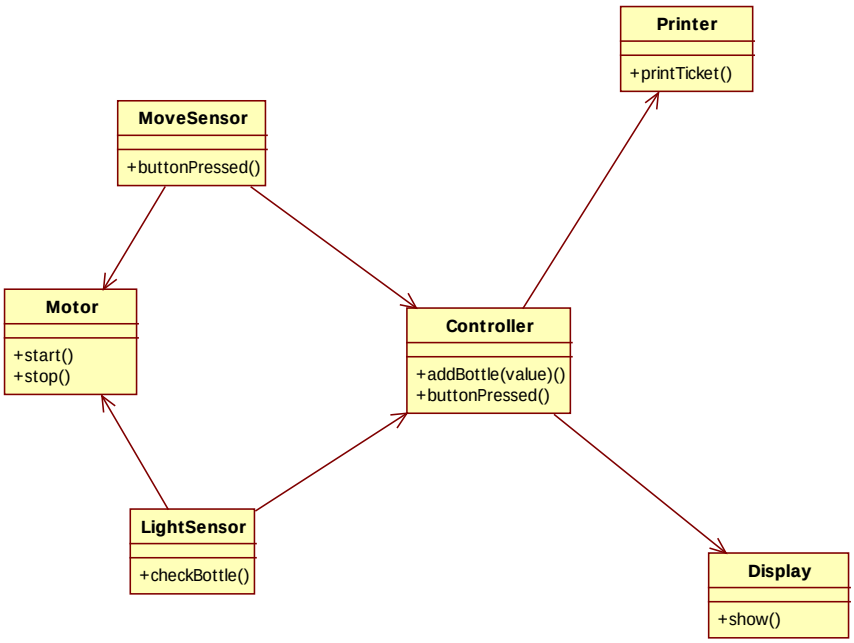
You should use Version Control System (GITHUB). In Optima there is also Java template project for the application ('BotRecMachineTemplate'). Download zipped file and unzip it to one folder. Copy all source files to your just created project in NetBeans. Then commit and push whole project to server. When working with the project remember often commit + push your changes to server.

Let's open the design with starUML tool (file 'UML model ready.uml' in optima). Basically it has 2 use cases. Make implementation one use case at a time. When you have first use case done, you can test it before starting the next one.

There are 2 different test UIs: Test.java (Command line application) and TestGUI (Java swing graphical UI application). The application code works together with both test UIs. So UI is separated from application which is one of the design principles. ***Observation*** design patterns is used for application communication with UI. In that way application don't have to know anything about UI. In addition to the business classes some utility classes are created: ***Event*** for communication between application and UI (Observation pattern). There is also ***Constants.java*** class where are all the application constants needed are defined.

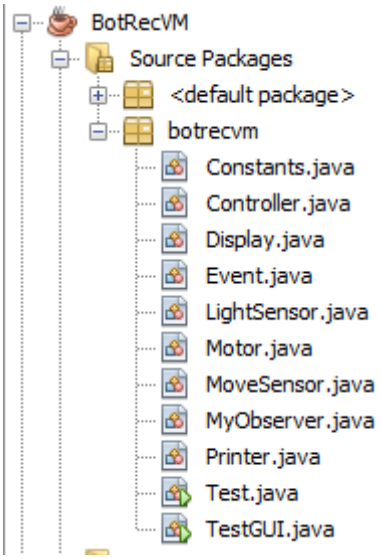
2. Initialization

First let's look class diagram of the design (This is also in uml file).



Picture 1: Class diagram

Each class has own file in implementation (All files exists in template):



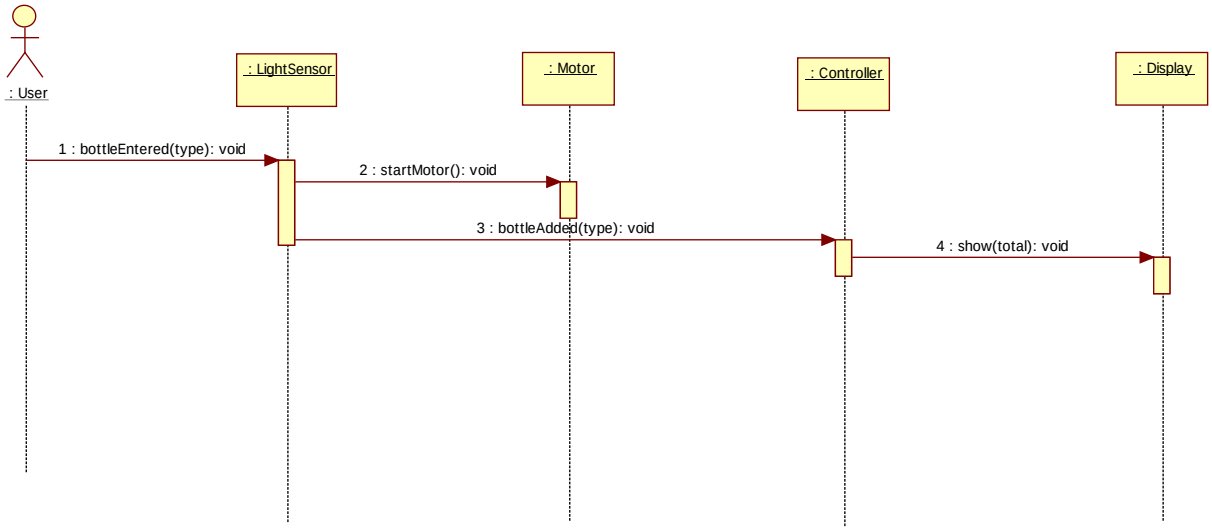
In test UI we have to create following objects:

```
10  public static void main(String[] args) {
11      MyObserver myobs = new MyObserver();
12      Motor motor = new Motor(myobs);
13      Controller controller = new Controller(myobs);
14      LightSensor lightSensor = new LightSensor(motor, controller);
15      MoveSensor moveSensor = new MoveSensor(motor, controller);
16
```

Your task is to implement all these constructors and create all associations as they are in class diagram. Follow comments in source files (Motor, Controller, LightSensor and MoveSensor). Now all objects and associations are created.

3. Use Case 1:

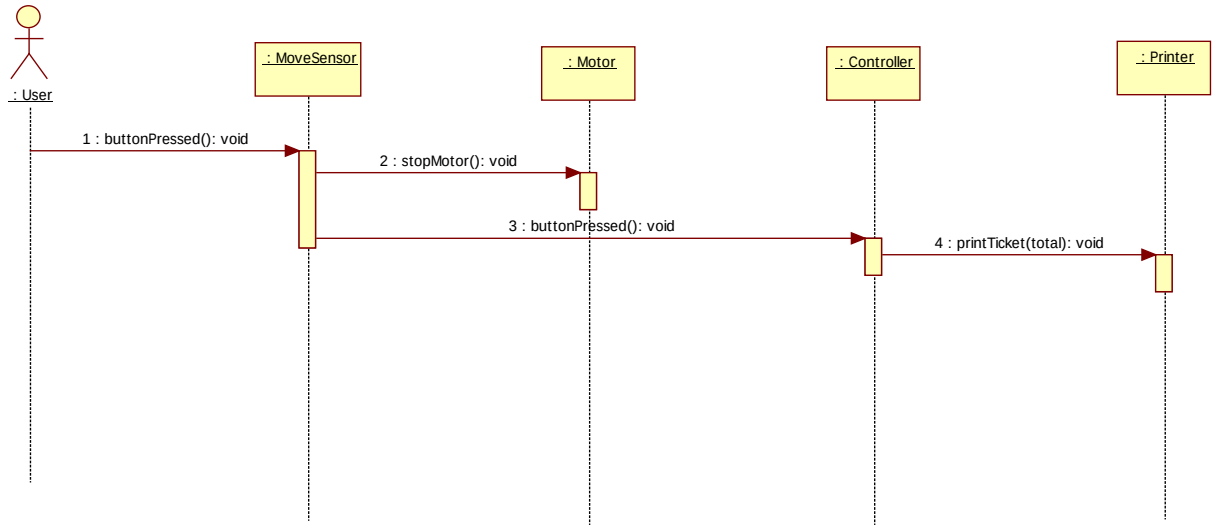
We start implementing first use case ‘Enter Bottles’. Let’s open sequence diagram in model:



Test UI acts as User. In line 20 in test UI we first ask the bottle from user. Then according to above diagram bottleEntered() method of LightSensor object is called. Then open LightSensor file and create bottleEntered() method. Follow comments in file. The flag member variable ***first*** is used to know that first bottle was entered and motor of the conveyor should be started. Let’s open file Motor.cpp and add the method. Follow comments in file. Next we write Controller method bottleAdded(). Follow comments in file. Display and Observer are already implemented in files. Now we have implemented first use case fully. Test it.

4. Use Case 2: Print Ticket

We continue implementing next use case 'Print Ticket. Let's open sequence diagram in the model:



Test UI acts as User. In line 25 in test UI we inform moveSensor object that button has been pressed. Implement method buttonPressed in MoveSensor file. Follow comments.

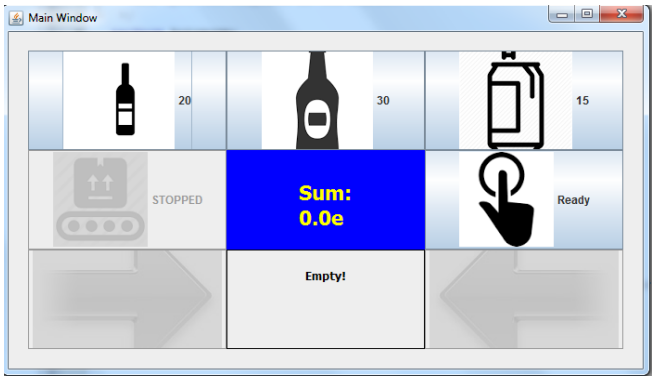
Then according to above diagram stopMotor () method of Motor object is called. Then we open Motor file and create stopMotor () method. UI is informed through Observer that motor has been stopped. Next Controller object is informed that button has been pressed. Let's open Controller file and add the method buttonPressed().

Printer class is already done in file.

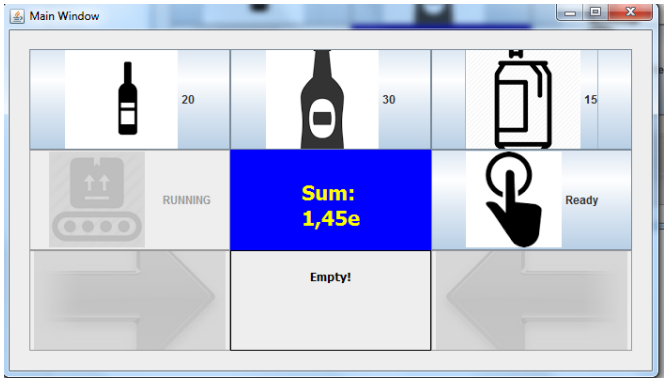
Finally test with both UIs.

APPENDIX A: Example test using GUI.

1. **Application starts**



2. **Enter bottles**



3. **Print Ticket**