Abstract factory Pattern(Source Code)

the AbstractFactory, GasPump1_Factory and GasPump2_Factory classes(inside the AbstractFactory package) are responsible for the abstract factory pattern.

Source code in the AbstractFactory class:

```
public abstract class AbstractFactory {

    public abstract Data getDataObj();

    public abstract InputProcessor getInputProcessor();

    public abstract StateMachine getStateMachine();

    public abstract OutputProcessor getOutputProcessor();

}
```

Source code in the GasPump1_Factory class:

```
public class GasPump1_Factory extends AbstractFactory {
       private StateMachine model;
       private OutputProcessor op;
    private DataForGasPump1 data;
    private IPGasPump1 ip;

    //constructor
    public GasPump1_Factory() {
        // create all the objects that the GasPump application needs
        data  = new DataForGasPump1();
        ip    = new IPGasPump1();
        model = new StateMachine();
        op    = new OutputProcessor();
        // set the InputProcessor's model object reference
        ip.setModel(model);
        // Both the InputProcessor and OutputProcessor need the data storage class for
gasPump-1
        ip.setData(data);
        op.setData(data);

        // For each actions needed in the gasPump-1, set the OutputProcessor's action
strategies
        // and also pass in the shared Data, if needed.
        // Since we use strategy design patterns, the specified algorithm can be
independently from clients that use it.
        AbstractCancelMsg cal_msg = new CancelMsg1();
        op.setCancelMsg(cal_msg);

        AbstractDisplayMenu display_menu = new DisplayMenu1(data);
        op.setDisplayMenu(display_menu);

        AbstractGasPumpedMsg gasPump_msg = new GasPumpedMsg1(data);
        op.setGasPumpedMsg(gasPump_msg);

        AbstractPayMsg pay_msg = new PayMsg1();
        op.setPayMsg(pay_msg);
```

```
        AbstractPrintReceipt printReceipt = new PrintReceipt1(data);
        op.setPrintReceipt(printReceipt);

        AbstractPumpGasUnit gasUnit = new PumpGasUnit1(data);
        op.setPumpGasUnit(gasUnit);

        AbstractReadyMsg ready_msg = new ReadyMsg1(data);
        op.setReadyMsg(ready_msg);

        AbstractRejectMsg reject_msg = new RejectMsg1();
        op.setRejectMsg(reject_msg);

        AbstractReturnCash returnCash = new ReturnCash1();
        op.setReturnCash(returnCash);

        AbstractSetInitialValues setInitial = new SetInitialValues1(data);
        op.setSetInitialValues(setInitial);

        AbstractSetPrice setPrice = new SetPrice1(data);
        op.setSetPrice(setPrice);

        AbstractStopMsg stop_msg = new StopMsg1();
        op.setStopMsg(stop_msg);

        AbstractStoreCash storeCash = new StoreCash1();
        op.setStoreCash(storeCash);

        AbstractStoreData storeData = new StoreData1(data);
        op.setStoreData(storeData);

        // set the EFSM model's OutputProcessor's object reference
        model.setOP(op);
    }

    //getters and setters
    @Override
    public Data getDataObj() {
        return this.data;
    }

    @Override
    public InputProcessor getInputProcessor() {
        return this.ip;
    }

    @Override
    public StateMachine getStateMachine() {
        return this.model;
    }

    @Override
    public OutputProcessor getOutputProcessor() {
        return this.op;
    }
}
```

source code in the GasPump2_Factory class:

```
public class GasPump2_Factory extends AbstractFactory {
       private StateMachine model;
       private OutputProcessor op;
    private DataForGasPump2 data;
    private IPGasPump2 ip;
    public GasPump2_Factory() {
        // create all the objects that the GasPump application needs
        data  = new DataForGasPump2();
        ip    = new IPGasPump2();
        model = new StateMachine();
        op    = new OutputProcessor();

        // set the InputProcessor's model object reference
        ip.setModel(model);
        // Both the InputProcessor and OutputProcessor need the data storage class for
gasPump-1
        ip.setData(data);
        op.setData(data);

        // For each actions needed in the gasPump-1, set the OutputProcessor's action
strategies
        // and also pass in the shared Data, if needed.
        // Since we use strategy design patterns, the specified algorithm can be
independently from clients that use it.
        AbstractCancelMsg cal_msg = new CancelMsg2();
        op.setCancelMsg(cal_msg);

        AbstractDisplayMenu display_menu = new DisplayMenu2(data);
        op.setDisplayMenu(display_menu);

        AbstractGasPumpedMsg gasPump_msg = new GasPumpedMsg2(data);
        op.setGasPumpedMsg(gasPump_msg);

        AbstractPayMsg pay_msg = new PayMsg2();
        op.setPayMsg(pay_msg);

        AbstractPrintReceipt printReceipt = new PrintReceipt2(data);
        op.setPrintReceipt(printReceipt);

        AbstractPumpGasUnit gasUnit = new PumpGasUnit2(data);
        op.setPumpGasUnit(gasUnit);
        AbstractReadyMsg ready_msg = new ReadyMsg2(data);
        op.setReadyMsg(ready_msg);

        AbstractRejectMsg reject_msg = new RejectMsg2();
        op.setRejectMsg(reject_msg);

        AbstractReturnCash returnCash = new ReturnCash2(data);
        op.setReturnCash(returnCash);

        AbstractSetInitialValues setInitial = new SetInitialValues2(data);
        op.setSetInitialValues(setInitial);

        AbstractSetPrice setPrice = new SetPrice2(data);
        op.setSetPrice(setPrice);
```

```
        AbstractStopMsg stop_msg = new StopMsg2();
        op.setStopMsg(stop_msg);

        AbstractStoreCash storeCash = new StoreCash2(data);
        op.setStoreCash(storeCash);

        AbstractStoreData storeData = new StoreData2(data);
        op.setStoreData(storeData);

        // set the EFSM model's OutputProcessor's object reference
        model.setOP(op);
    }

    @Override
    public Data getDataObj() {
        return this.data;
    }

    @Override
    public InputProcessor getInputProcessor() {
        return this.ip;
    }

    @Override
    public StateMachine getStateMachine() {
        return this.model;
    }

    @Override
    public OutputProcessor getOutputProcessor() {
        return this.op;
    }

}
```