Project Part #2 is posted

deadline: Friday, December 5

sample test cases are posted.

Exam #3

Monday, December 8
at 5:00 - 7:00 pm.

The coverage is posted

This is a comprehensive exam

Sample exam #3 is posted

COVERAGE FOR EXAM #3

CS 586; Fall 2025

Exam #3 will be held on **Monday, December 8, 2025,** between **5:00-7:00 p.m.**
Location: 152 PS (Pritzker Science Center)
The exam is a **CLOSED books and notes** exam.

Coverage for the exam:

- OO design patterns: item description, whole-part, observer, state, proxy, adapter, strategy, and abstract factory patterns. [Textbook: Sections 3.1, 3.2; Section 3.4 (pp. 263-275); Section 3.6 (pp.339-343); Handout #1, class notes]
- Interactive systems. Model-View-Controller architectural pattern [Textbook: Section 2.4, pp. 123-143]
- Client-Server Architecture
    - Client-Dispatcher-Server [Section 3.6: pp. 323-337]
    - Client-Broker-Server Architecture [Textbook: Section 2.3; pp. 99-122]
- Layered architecture [Textbook: Section 2.2; pp. 31-51]
- Pipe and Filter Architecture [Textbook: Section 2.2; pp. 53-70]
- Adaptable Systems:
    - Micro-kernel architectural pattern [Textbook: Section 2.5, pp. 169-192]
- Fault-tolerant architecture [Handout #2]
    - N-version architecture
    - Recovery-Block architecture
    - N-Self Checking architecture

Sources:

- Textbook: F. Buschmann, et. al., Pattern-oriented software architecture, vol. I, John Wiley & Sons.
- Class notes
- Handouts

Project #2

Deadline: Friday, December 5.

Report

4. sequence diagram
   assume that all
   objects are initialized

PART #2: PROJECT DESIGN, IMPLEMENTATION, and REPORT

CS 586; Fall 2025

Final Project Deadline: **Friday, December 5, 2025**
Late submissions: ~~50% off~~
After **December 9,** the final project will not be accepted.

Submission: The project must be submitted on Canvas. The hardcopy submissions will not be accepted.

This is an **individual** project, not a team project. Identical or similar submissions will be penalized.

DESIGN and IMPLEMENTATION

The goal of the second part of the project is to design two *Gas Pump* components using the Model-Driven Architecture (MDA) and then implement these *Gas Pump* components based on this design using the OO programming language. This OO-oriented design should be based on the MDA-EFSM for both *Gas Pump* components that was identified in the first part of the project. You may use your own MDA-EFSM (assuming that it was correct), or you can use the posted sample MDA-EFSM. In your design, you **MUST** use the following OO design patterns:

- state pattern
- strategy pattern
- abstract factory pattern

In the design, you need to provide the class diagram, in which the coupling between components should be minimized and the cohesion of components should be maximized (components with high cohesion and low coupling between components). In addition, two sequence diagrams should be provided as described on the next page (Section 4 of the report).

After the design is completed, you need to implement the *Gas Pump* components based on your design using the OO programming language. In addition, the driver for the project to execute and test the correctness of the design and its implementation for the *Gas Pump* components must be implemented.

Outline of the Report & Deliverables

I: REPORT

The report **must** be submitted as one PDF file (otherwise, a **10% penalty will be applied).**

1. MDA-EFSM model for the *Gas Pump* components
   a. A list of meta events for the MDA-EFSM
   b. A list of meta actions for the MDA-EFSM with their descriptions
   c. A state diagram of the MDA-EFSM
   d. Pseudo-code of all operations of the Input Processors of Gas Pumps*: GP-1* and *GP-2*

2. Class diagram(s) of the MDA of the *Gas Pump* components. In your design, you **MUST** use the following OO design patterns:
   a. State pattern
   b. Strategy pattern
   c. Abstract factory pattern

split into several pages

3. For each class in the class diagram(s), you should:
   a. Describe the purpose of the class, i.e., responsibilities.
   b. Describe the responsibility of each operation supported by each class.

4. Dynamics. Provide sequence diagrams for two Scenarios:
   a. Scenario-I should show how one liter of gas is dispensed in *GasPump-1*, i.e., the following sequence of operations is issued: *Activate(4.1), Start(), PayCash(5.2), StartPump(), PumpLiter(), PumpLiter()*
   b. Scenario-II should show how one gallon of Regular gas is dispensed in *GasPump-2*, i.e., the following sequence of operations is issued: *Activate(4, 7), Start(), PayDebit(123), Pin(124), Pin(123), Regular(), StartPump(), PumpGallon(), FullTank()*

II: Well-documented (commented) source code

In the source code, you should clearly indicate/highlight which parts of the source code are responsible for the implementation of the three required design patterns (**if this is not clearly indicated in the source code, 20 points will be deducted**):

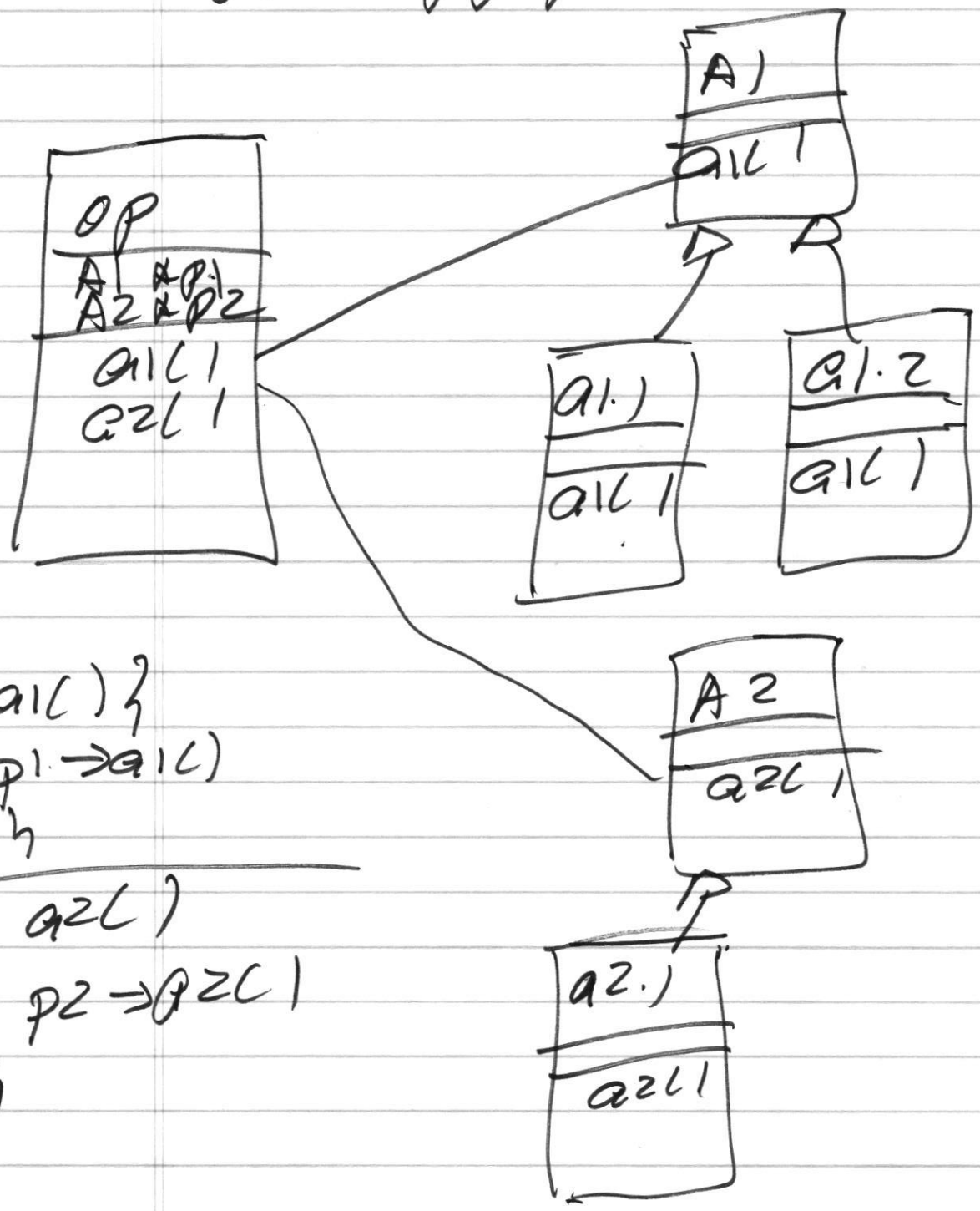
- state pattern
- strategy pattern
- abstract factory pattern.

The source code must be submitted on Canvas. Note that the source code may be compiled during the grading and then executed. If the source code is not provided, **15 POINTS** will be deducted.

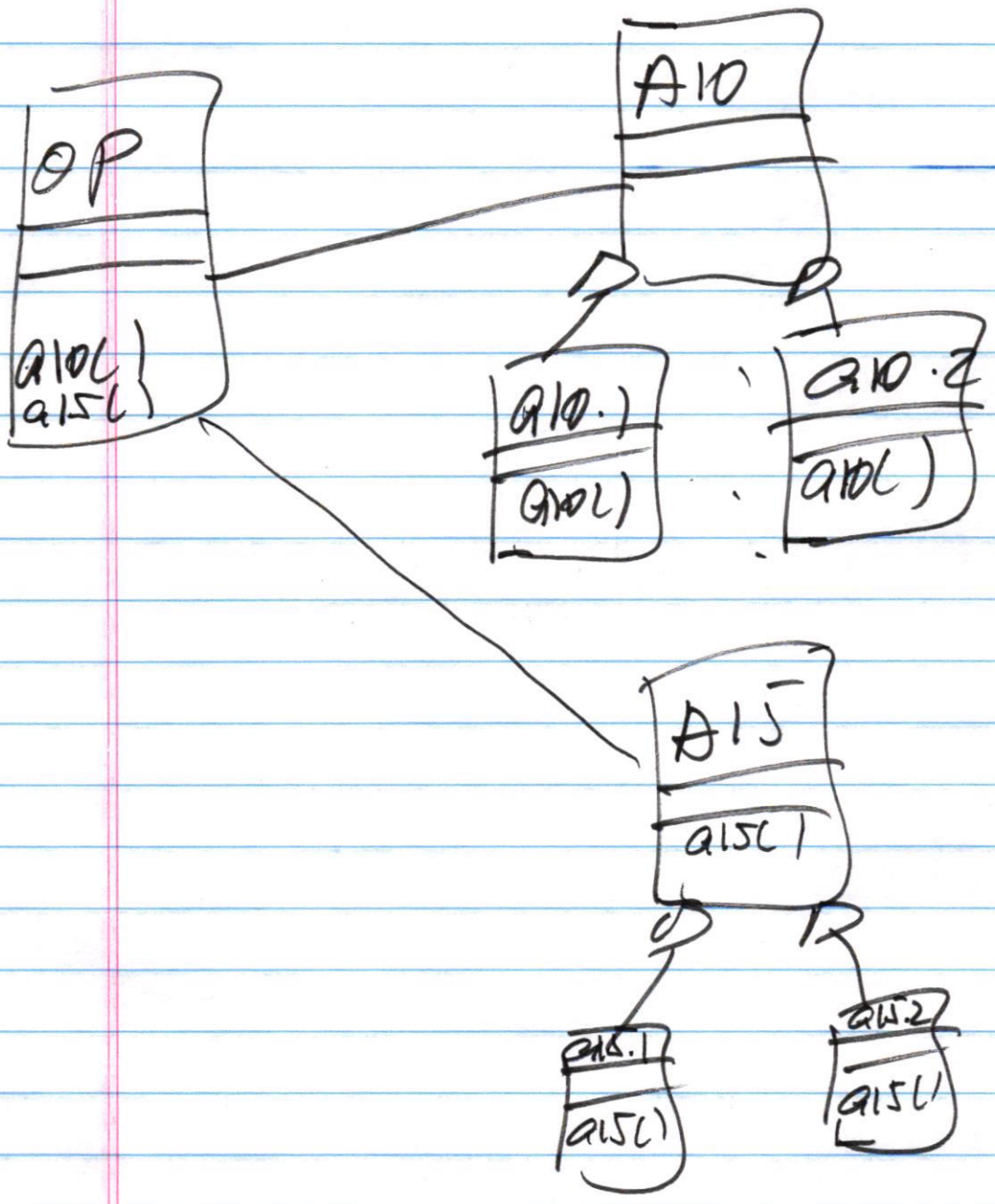
III: Project executables

The project executable(s) of the *Gas Pump* components, with detailed instructions explaining the execution of the program, must be prepared and made available for grading. The project executable should be submitted on Canvas. If the executable is not provided (or not easily available), **20 POINTS** will be automatically deducted from the project grade.

class diagram
strategy pattern




```
a1() }
p1 -> a1()
}
---------------
  a2()
} p2 -> a2()
}
```


OP
q10()
q15()
A10
q10.1
q10()
q10.2
q10()
A15
q15()
q15.1
q15()
q15.2
q15()

CS586 PROJECT – Sample Test Cases

CS 586; Fall 2025

GAS PUMP #1

Test #1: *Activate(4.1), Start(), PayCash(5.2), StartPump(), PumpLiter(), PumpLiter()*
Expected outcome: 1 liter pumped; Receipt total: $4.1

Test #2: *Activate(4.1), Start(), PayCash(5.2), Activate(7.2), StartPump(), PumpLiter(), PumpLiter()*
Expected outcome: 1 liter pumped; Receipt total: $4.1

Test #3: *Activate(4.1), Start(), PayCash(5.2), Cancel(), Start(), PayCredit(), Approved(), StartPump(), PumpLiter(), PumpLiter(), StopPump()*
Expected outcome: 2 liters pumped; Receipt total: $8.2

Test #4: *Activate(4.1), Start(), PayCredit(), Approved(), PayCash(5.2), StartPump(), PumpLiter(), PumpLiter(), StopPump()*
Expected outcome: 2 liters pumped; Receipt total: $8.2

Test #5: *Activate(4.1), Start(), PayCredit(), Reject(), Start(), PayCash(9), StartPump(), PumpLiter(), PumpLiter(), StopPump()*
Expected outcome: 2 liters pumped; Receipt total: $8.2

GAS PUMP #2

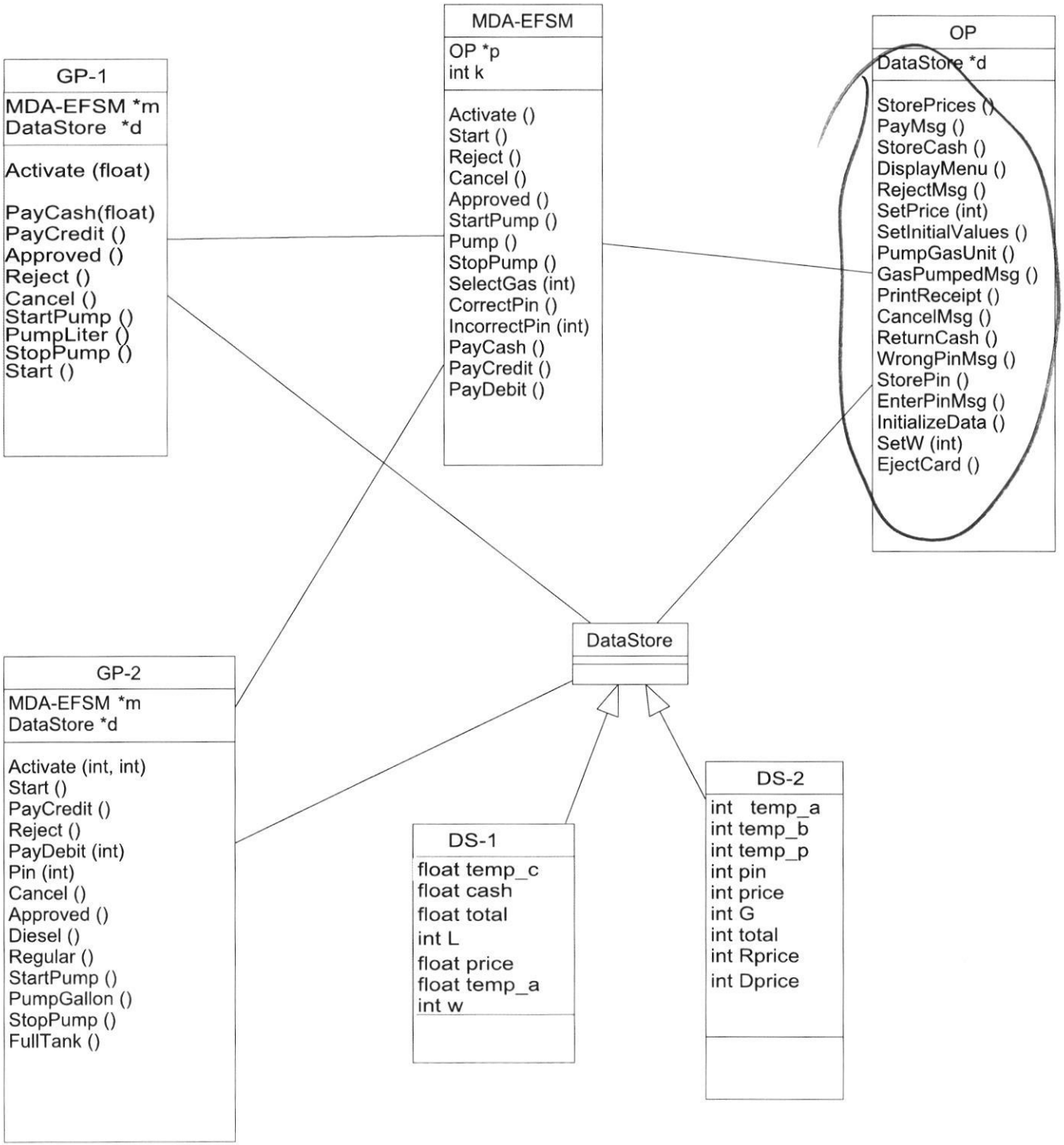
Test #6: *Activate(4, 7), Start(), PayDebit(123), Pin(124), Pin(123), Regular(), StartPump(), PumpGallon(), FullTank()*
Expected outcome: wrong pin msg, 1 gallon pumped; Receipt total: $4

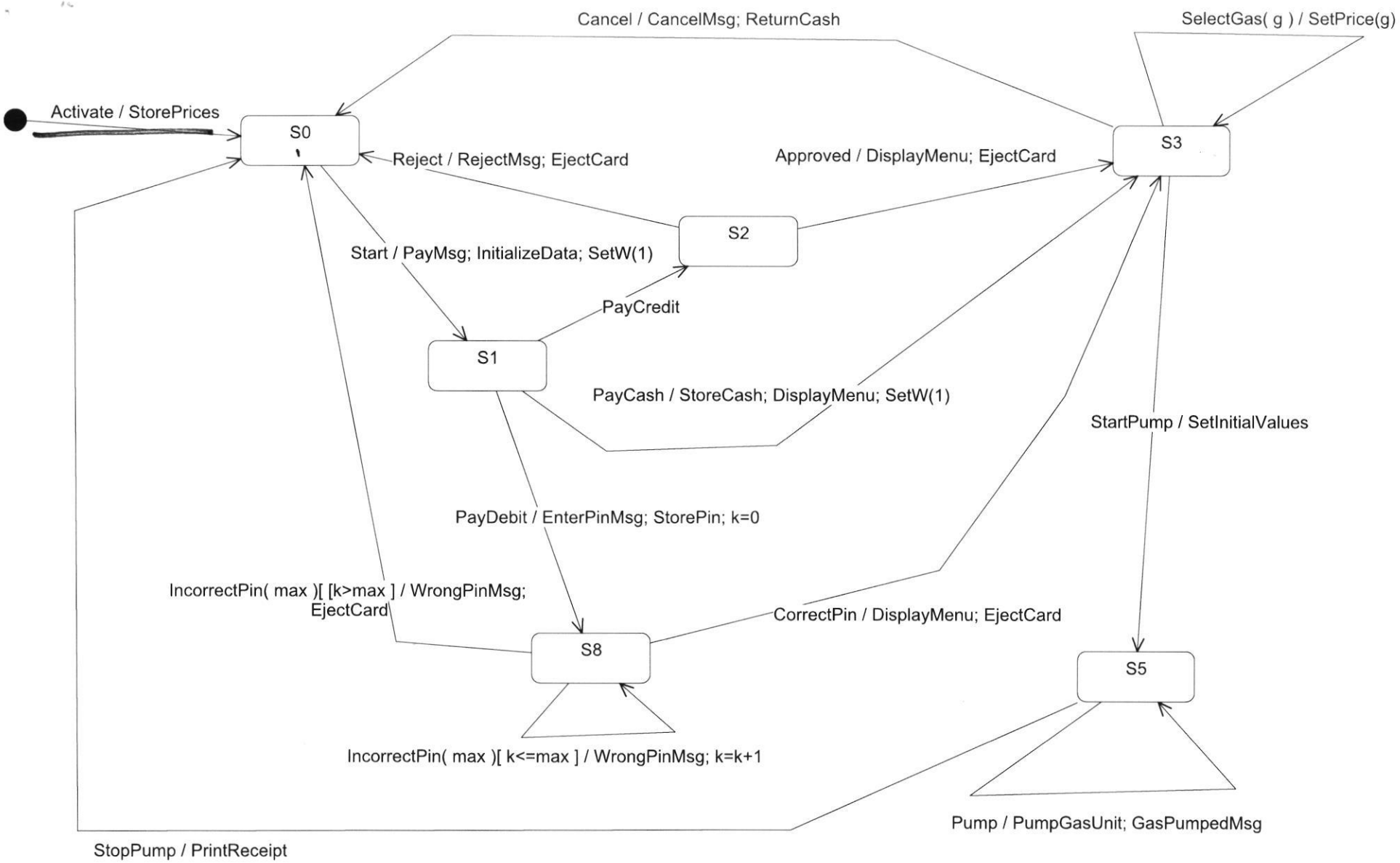
Test #7: *Activate(4, 7), Start(), PayDebit(123), Pin(123), Regular(), StartPump(), Activate(7, 3), PumpGallon(), FullTank()*
Expected outcome: 1 gallon pumped; Receipt total: $4

Test #8: *Activate(4, 7), Start(), PayDebit(123), Pin(123), Regular(), StartPump(), Diesel(), PumpGallon(), FullTank()*
Expected outcome: 1 gallon pumped; Receipt total: $4

Test #9: *Activate(4, 7), Start(), PayDebit(123), Pin(124), Pin(124), Pin(124), Start(), PayCredit(), Approved(), Diesel(), StartPump(), PumpGallon(), PumpGallon(), StopPump()*
Expected outcome: Wrong pin msg, Too many attempts msg; 2 gallons pumped; Receipt total: $14

Test #10: *Activate(4, 7), Start(), PayDebit(123), Pin(123), Regular(), Diesel(), StartPump(), PumpGallon(), FullTank()*
Expected outcome: 1 gallon pumped; Receipt total: $7


GP-1
MDA-EFSM *m
DataStore *d
Activate (float)
PayCash(float)
PayCredit ()
Approved ()
Reject ()
Cancel ()
StartPump ()
PumpLiter ()
StopPump ()
Start ()
MDA-EFSM
OP *p
int k
Activate ()
Start ()
Reject ()
Cancel ()
Approved ()
StartPump ()
Pump ()
StopPump ()
SelectGas (int)
CorrectPin ()
IncorrectPin (int)
PayCash ()
PayCredit ()
PayDebit ()
OP
DataStore *d
StorePrices ()
PayMsg ()
StoreCash ()
DisplayMenu ()
RejectMsg ()
SetPrice (int)
SetInitialValues ()
PumpGasUnit ()
GasPumpedMsg ()
PrintReceipt ()
CancelMsg ()
ReturnCash ()
WrongPinMsg ()
StorePin ()
EnterPinMsg ()
InitializeData ()
SetW (int)
EjectCard ()
DataStore
GP-2
MDA-EFSM *m
DataStore *d
Activate (int, int)
Start ()
PayCredit ()
Reject ()
PayDebit (int)
Pin (int)
Cancel ()
Approved ()
Diesel ()
Regular ()
StartPump ()
PumpGallon ()
StopPump ()
FullTank ()
DS-1
float temp_c
float cash
float total
int L
float price
float temp_a
int w
DS-2
int temp_a
int temp_b
int temp_p
int pin
int price
int G
int total
int Rprice
int Dprice



MDA-EFSM for Gas Pumps

MDA-EFSM Events:

Activate()
Start()
PayCredit()
PayCash()
PayDebit()
Reject()
Cancel()
Approved()
StartPump()
Pump()
StopPump()
SelectGas(int g)
CorrectPin()
IncorrectPin(int max)

MDA-EFSM Actions:

StorePrices	// stores price(s) for the gas from the temporary data store
PayMsg	// displays a type of payment method
StoreCash	// stores cash from the temporary data store
DisplayMenu	// display a menu with a list of selections
RejectMsg	// displays credit card not approved message
SetPrice(int g)	// set the price for the gas identified by *g* identifier as in SelectGas(int g)
SetInitialValues	// set *G* (or *L*) and *total* to 0;
PumpGasUnit	// disposes unit of gas and counts # of units disposed
GasPumpedMsg	// displays the amount of disposed gas
PrintReceipt	// print a receipt
CancelMsg	// displays a cancellation message
ReturnCash	// returns the remaining cash
WrongPinMsg	// displays incorrect pin message
StorePin	// stores the pin from the temporary data store
EnterPinMsg	// displays a message to enter pin
InitializeData	// set the value of *price* to 0 for GP-2; do nothing for GP-1
EjectCard()	// card is ejected
SetW(int w)	// set value for cash flag

Operations of the Input Processor
(GasPump-1)

```
Activate(float a) {
        if (a>0) {
            d->temp_a=a;
            m->Activate()
        }
}

Start() {
        m->Start();
}
PayCash(float c) {
        if (c>0) {
            d->temp_c=c;
            m->PayCash()
        }
}

PayCredit() {
        m->PayCredit();
}

Reject() {
        m->Reject();
}

Approved() {
        m-> Approved();
}
Cancel() {
        m->Cancel();
}

StartPump() {
        m->StartPump();
}
```

```
PumpLiter() {
if (d->w==1) m->Pump()
else if (d->cash>0)&&(d->cash < d->price*(d->L+1))
            m->StopPump();
else m->Pump()
}

StopPump() {
        m->StopPump();
}
```

Notice:
cash: contains the value of cash deposited
price: contains the price of the selected gas
L: contains the number of liters already pumped
w: cash flag (cash: w=0; otherwise: w=1)
cash, *L*, *price, w* are in the data store
m: is a pointer to the MDA-EFSM object
d: is a pointer to the Data Store object

Operations of the Input Processor (GasPump-2)

```
Activate(int a, int b) {
        if ((a>0)&&(b>0)) {
            d->temp_a=a;
            d->temp_b=b;
            m->Activate()
        }
}

Start() {
        m->Start();
}

PayCredit() {
        m->PayCredit();
}

Reject() {
        m->Reject();
}

PayDebit(int p) {
        d->temp_p=p;
        m->PayDebit();
}

Pin(int x) {
        if (d->pin==x) m->CorrectPin()
        else m->InCorrectPin(1);
}

Cancel() {
        m->Cancel();
}

Approved() {
        m->Approved();
}

Diesel() {
        m->SelectGas(2)
}

Regular() {
        m->SelectGas(1)
}

StartPump() {
        if (d->price>0) m->StartPump();
}

PumpGallon() {
        m->Pump();

StopPump() {
        m->StopPump();
}

FullTank() {
        m->StopPump();
}
```

Notice:
pin: contains the pin in the data store
m: is a pointer to the MDA-EFSM object
d: is a pointer to the Data Store object
SelectGas(g): Regular: g=1; Diesel: g=2