**The following are my solutions to the assignments in Lab02:**

1. **Use Case diagram**

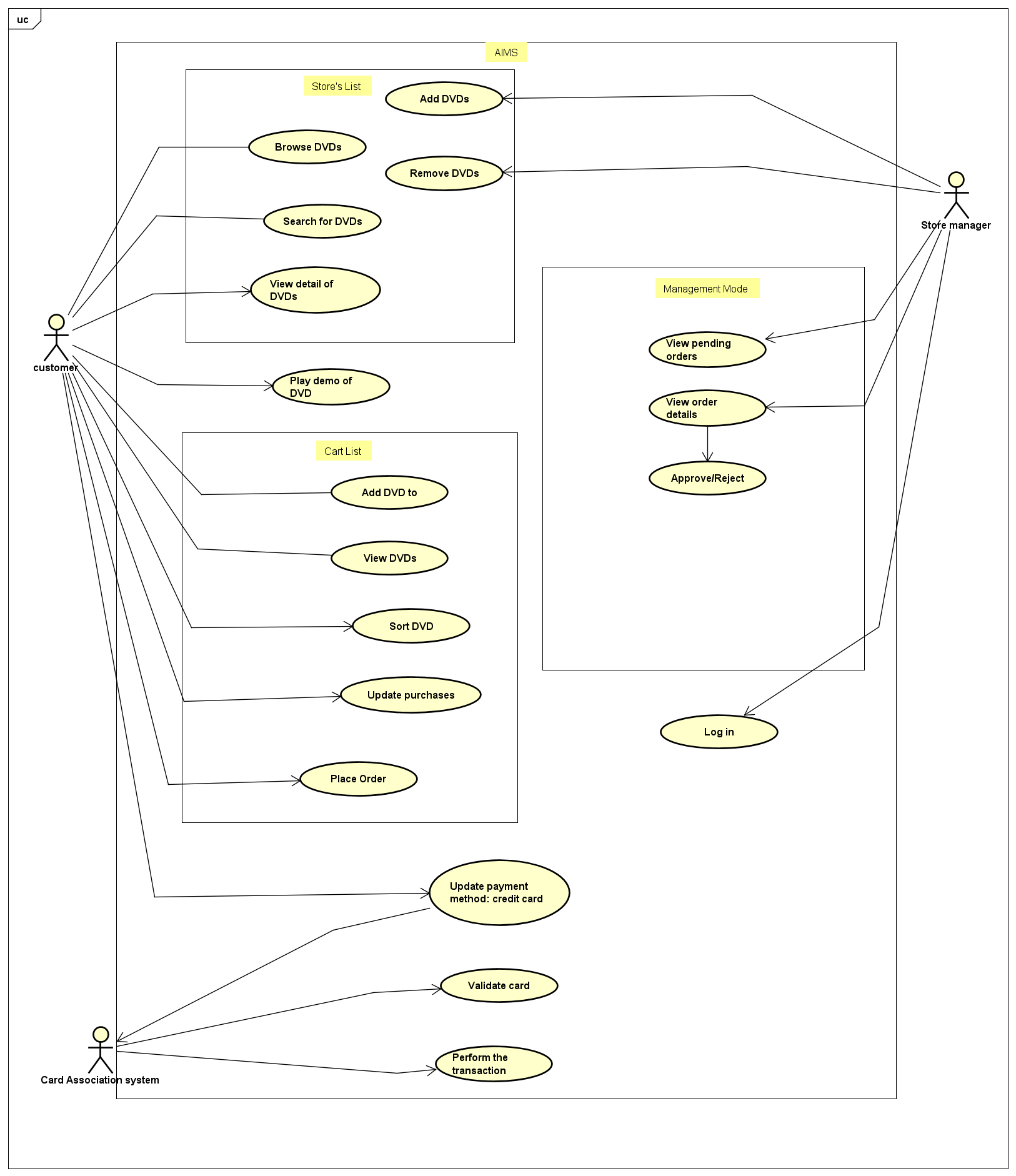


Fig 1. My Use Case diagram answer

1. **Class diagram**

A screenshot of a computer

Description automatically generated

Fig 2. My Class diagram answer

1. **Reading assignent:**

(The following are just my drafted thoughts to construct my mindmap in the ReadingAssignment folder)

1. problem with accessors

A fundamental precept of OO systems is that an object should not expose any of its implementation details.

This way, you can change the implementation without changing the code that uses the object.

=> That’s one reason why all of your instance variables (a class’s nonconstant fields) should be private.

=> Getter and setter methods (also known as accessors) are dangerous for the same reason that public fields are dangerous: They provide external access to implementation details.

example: if a particular class gets used (say, by calling its accessor methods) a lot in your program and you suddenly change the class' anatomy, you would have to edit all those method calls, and we don't have all day to be sitting here doing that

=> By designing carefully and focusing on what you must do rather than how you’ll do it (the "Object" in OO), you eliminate the vast majority of getter/setter methods in your program

2. what should we do?

- When making a class, a getIdentity() method would not be an accessor if it returns an "identity object": a complex object that does not reveal the implementation of the class itself, and

thus doesn't make rippling headaches throughout your programs when you have to edit said class.

- use Cunningham and Beck's OO design process:

+ look at at use case, take a guess on which classes would be needed to perform that use case (instead of thinking about how that use case would be done!)

+ build those classes, with:

+) the name of the classes

+) what each class could do (methods). keep these very specialized

+) which other classes can talk to it. keep the messages as small as possible

+ not enough classes to perform the use case? add more until you can do it

=> this can be captured with an UML system of diagrams!

1. **Source code:** in the folder “Aims”