**Final Exam** (CSC341)

1. The following instance method has an undesirable if-else structure, which is not desirable.

*class MailSystem {*

*private List<Mail> mails;*

*// unimportant code omitted*

*public void handleMails(){*

*for (int i = 0; i < mails.size(); i++){*

*Mail mail = mails.get(i);*

*if( mail instanceof Regular ) processRegularMail( mail );*

*else if( mail instanceof Priority ) processPriorityMail ( mail );*

*else if( mail instanceof Express) processExpressMail( mail );*

*else processFirstClassMail( mail );*

*}*

*}*

*}*

* 1. Modify the design behind the code to correct the problem and express your modified design with a class diagram.
  2. Re-implement the method to show the improvement.

1. Write a use case based on your experiences of a self-checkout process at a grocery store; the use case should include
   1. A main scenario, one that occurs most of time
   2. One extension scenario of your choosing
2. For the use case in Question 2, identify software actions/methods, representing software operations of the self-checkout system and its collaborating entities such as a display device, a scanner, a credit card reader, and banking services. Then,
   * 1. Construct a sequence diagram to show this process of interactions among different entities.
     2. Construct a state diagram to visualize the same process.
3. Your posts at Facebook end up on your friends’ home screen right after you push button “Post”. Use a sequence diagram to describe how this software process could be implemented using the *Observer* design pattern.
4. Given interface:

*interface TeamMember{ void perform(); }*

Write (in a simplest way you think possible) two classes to demonstrate use of Composite design pattern, one for modeling a team member, the other for modeling a team leader (who is a team member also).

1. Write a method, *sum (IFunction fun, int n)*, to compute sum: *f(1) + f(2) + … + f(n)*, where *f* is any given function. Parameter type *IFunction* is an appropriate interface type.
   * 1. Define the interface *IFunction*
     2. Implement the method *sum*
     3. Test the method with function *Math.sin(x)*, and then with *Math.cos(x)*.
2. Suppose a light controller has a collection of “commands”, each would turn a light off or on by invoking method *execute()*. Use Command pattern to implement this controller with two Command instances, one represents “turn light on” command and the other “turn light off” command. (Hint: command receiver is an object of Light.)
3. There was a question in the midterm exam about modeling operations between a switch and a lamp. This scenario can also be implemented using the Bridge pattern. Write some code to demonstrate that. (It is possible that you might provide the same solution if you already used the pattern (in your solution to that midterm question) without knowing the pattern.)