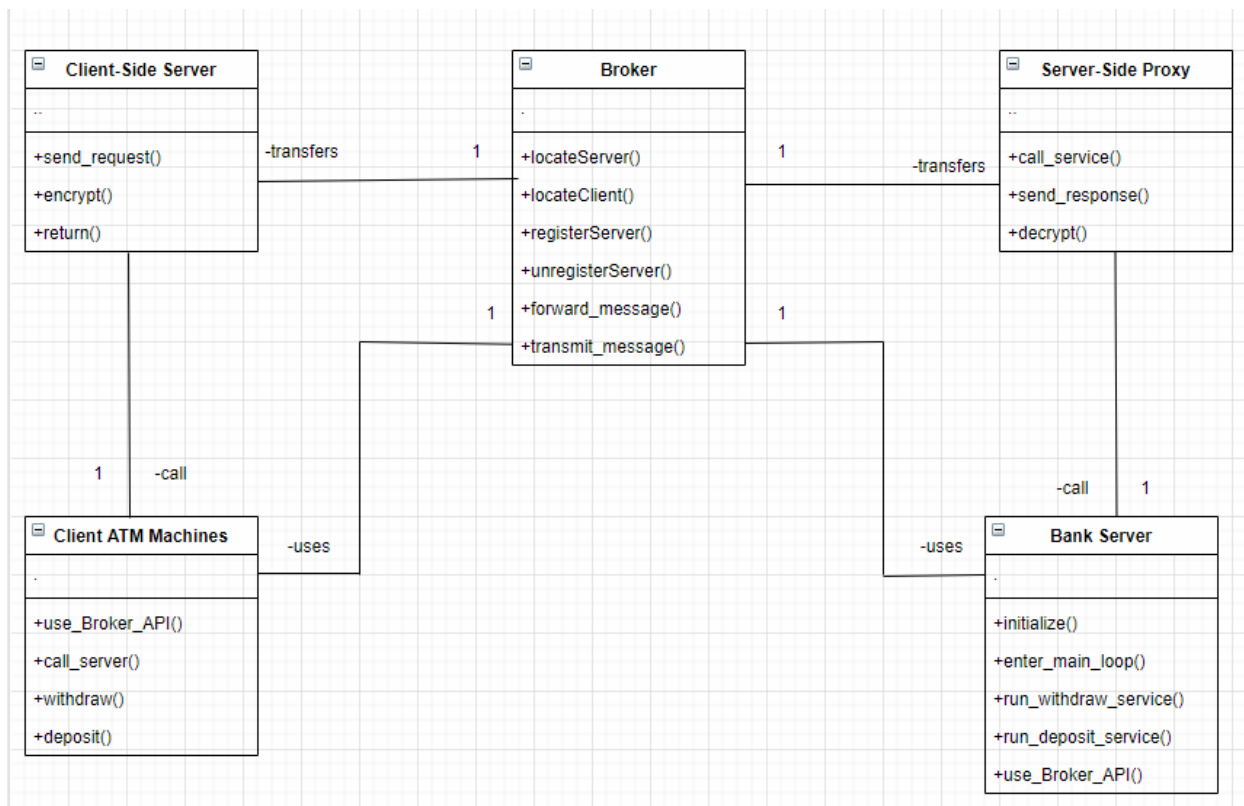


Group 10 : Assignment 3

Question 1

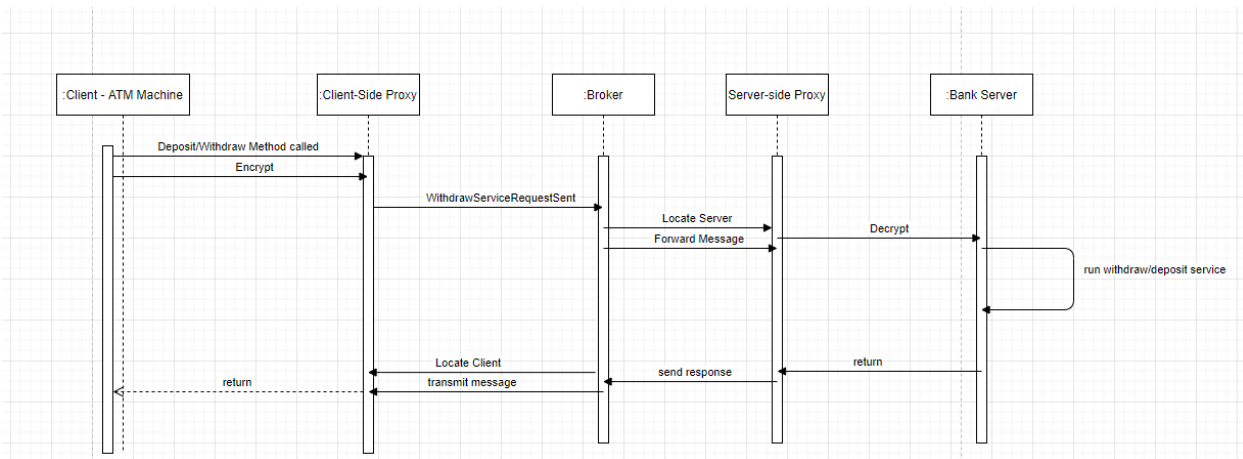
In a client-server system for a hypothetical banking application, the clients are ATM machines, and the server is the bank server. Suppose that the client has two operations: **withdraw** and **deposit**. Use the broker architectural pattern to document an initial architecture design for this system:

1. Draw a class diagram for this system. Use a client-side proxy to encrypt the data using an **encrypt** operation and use a server-side proxy to decrypt the data.



The main functions are the **withdraw** and **Deposit** function. The Client calls them, Broker API is used to connect to broker, the broker locates the server, the process happens and then everything is returned.

2. Draw a sequence diagram for this system.



3. Suppose that we want greater availability of the server, discuss what kind of tactics you should use to achieve that.

Availability means making sure that any kind of service outage period does not exceed a required value over a specific time interval. There are various availability tactics. Few of the tactics we can achieve are :

Detecting Fault : The clients are ATM machines. So an exception detection with a time stamp would make sure that we have a detection of the system condition that alters the normal flow of execution.

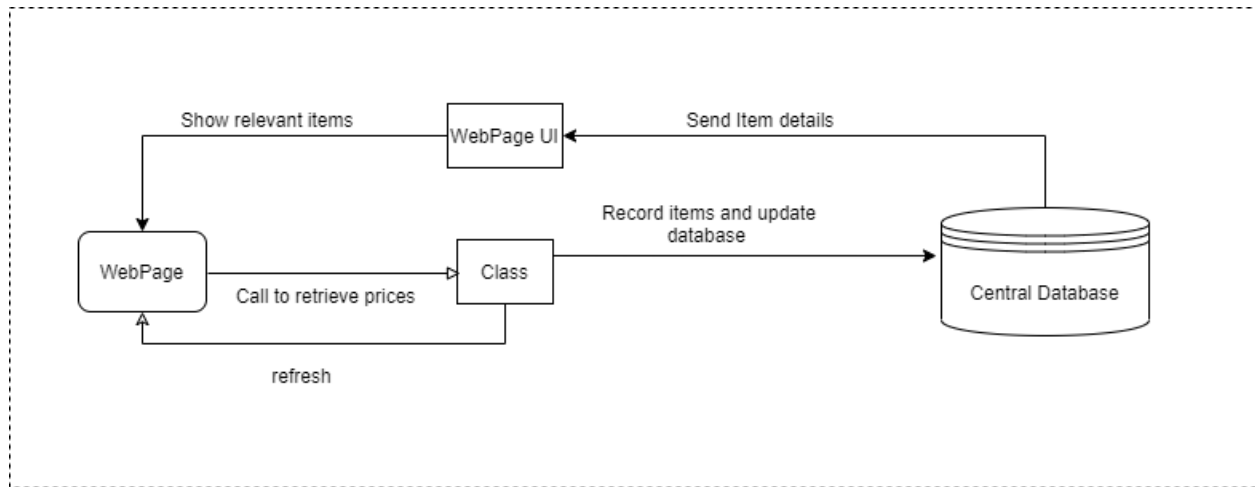
Recover from faults : Once the exception is detected, the system will handle the exception via the server to ensure that the system is still in working mode. There can also be constant software upgrades for the ATM machines as well the servers to executable code in a non-service-affecting manner.

In case there is a total failure of a part then there should be a reintroduction for the failed component and escalate the restart. For example, a particular system of the ATM machine stops working, we need a replacement for it because there is no other way.

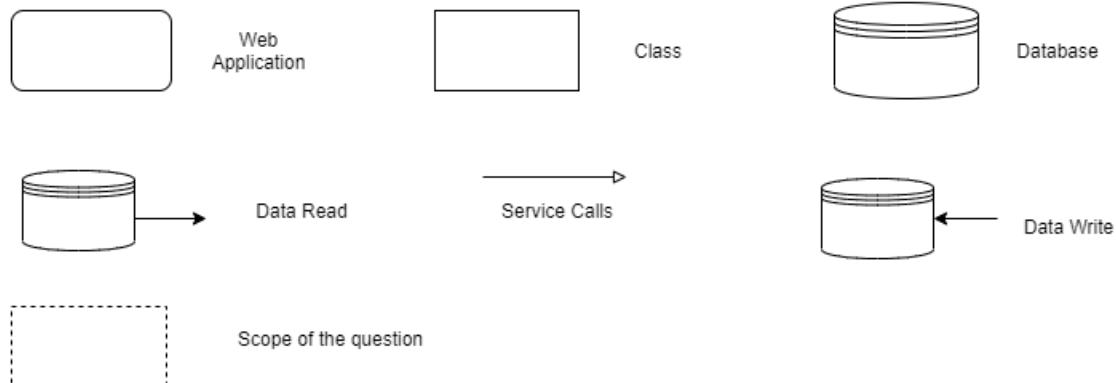
Prevent faults : Further, to ensure a greater availability, we must try to prevent faults. There should be transaction to ensure asynchronous messages exchanged between distributed components are durable by two-phase commit to prevent any race conditions caused by two processes attempting to update the same data item. Also, we will ensure an exception prevention. There can be techniques to allow system to recover from exceptions.

Question 2

The architectural pattern suitable for this type of system would be a *Service Oriented architecture*.



KEY



1. The webpage receives some sort of stimulus and calls the class.
2. The class retrieves, processes, and updates the database itself.
3. The class calls the webpage to refresh itself.
4. Every initialization of the webpage requires data from the central database. The central database does not directly update the webpage, it sends data to an intermediary class "WebPage UI" which then updates the webpage.

Comments on Modifiability: We will comment on the modifiability of the architectural pattern by answering *the three questions on modifiability*:

1. What can change?

Ans: In this scenario, except the database almost any component can change.

2. What is the likelihood of the change?

Ans: It is not very likely that any major change might occur. However, any bugs or system errors will require change

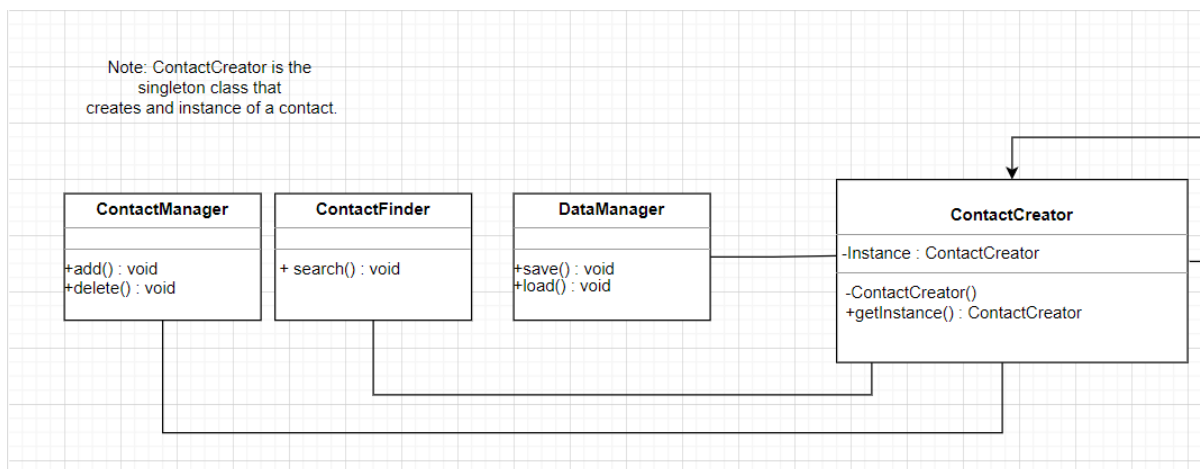
3. When is the change made?

Ans: The change is made when the web store needs to introduce new features or seasonal events.

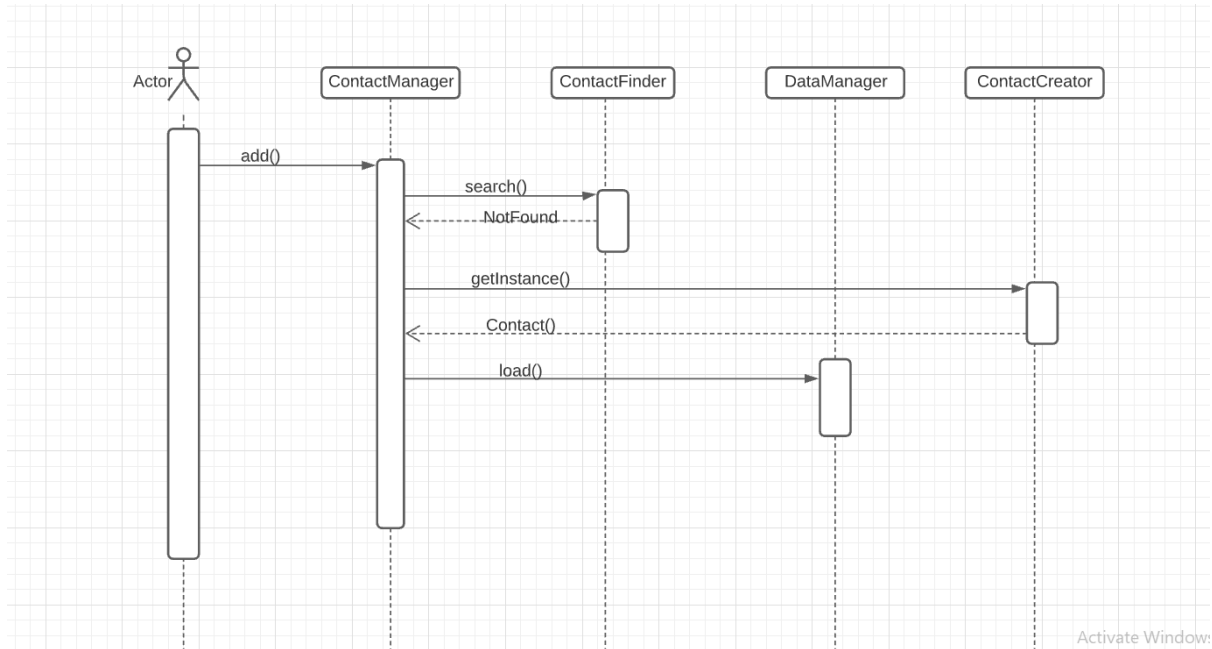
From the above scenarios we see that changes might need to happen on a seasonal basis and almost any component can change. Even in that case the pattern is still modifiable. Since each component is distinct and separate, the changes in one pattern will result in minimal changes to another. Also, due to low coupling, sometimes one component can be changed even without changing another. For example : the webpage. Therefore, the architectural pattern given here is easily modifiable.

Question 3

1) The design pattern that can be used to design the user interface would be the Singleton design pattern. Here we create and maintain an Address Book Application that allows the user to add, delete, search, save and load her contact information. Here we need only one instance of the Address Book which has to be accessible form everywhere. Hence Singleton design pattern can be used to implement the user interface.



2) UML sequence diagram showing behavioural sequence of program when a new contact is added.



Question 4

The system can be modified using various modifiability tactics with an eye on controlling the complexity of the changes. The problem that is visible in the system is strong dependencies between the modules in performing the tasks. In order to solve the problem we can increase the cohesion and lower the coupling. In the case of task #T1 being performed by both A and C, the responsibility #T1 is shared by modules A and C. Since #T1 is performed by A, calling A requires calling B and calling B requires calling A. This dependency can be broken by putting into effect an intermediary.

Another tactic that can be used is to restrict the dependencies. In the given system, calling A required calling B and vice versa. Here we can prevent both modules A and B depending on each other by preventing the case that calling A requires calling B and calling B requires calling A.

In case of #T1, #T2, #T3 not serving the same purpose the tasks can be placed in different modules for example #T1 in module A, #T2 in module B and #T3 in module C thereby increasing the semantic coherence.

Question 5

In the performance scenario of Trent Course Registration System, the servers hold information of courses available, details about course and status of the course (full or vacancies). The clients are individual PCs and systems that enable students to register for courses. The clients initiate interactions with servers with request of registrations and wait for adequate result from the server. If the desired courses are available the registration request is approved and if not the options for adding client to wait list is provided. If the client is not satisfied with the registration options, process is declined.

Elements:

- Client, is the multiple systems that allow students to invoke the services offered by the Trent student registration system. The portals in the client provide the students with the information about the services offered by the registration server.
- Server, is the system that provides the students with the registration services. The registration system provides necessary information regarding their registration process.

Major concern for this system is Latency because if not took care of, it can lead to session failures.

Latency: the delay between user request for services and the time the server takes to provide the services asked for. In the Trent course registration system, the heavy load of service requests when the registration period starts can lead to increased latency and session failure. In order to reduce latency different course departments can each be given one server. The common initial processes of registration can be sorted out and can be put in one server.

Throughput is the amount of data that can be transferred from one location to another in a given amount of time. In the current scenario the interactions between client and server can be overloaded and there by throughput can get increased. This increase in throughput can in turn increase latency and hence latency is our major concern.

Question 6

The characteristic of security which will be ensured in this scenario is *Confidentiality*. Confidentiality is the property of data to prevent access to unauthorized users.

Concrete Scenario: A pickpocket has stolen an ATM access card but does not know the pin. The thief tries multiple pins to randomly access the account.

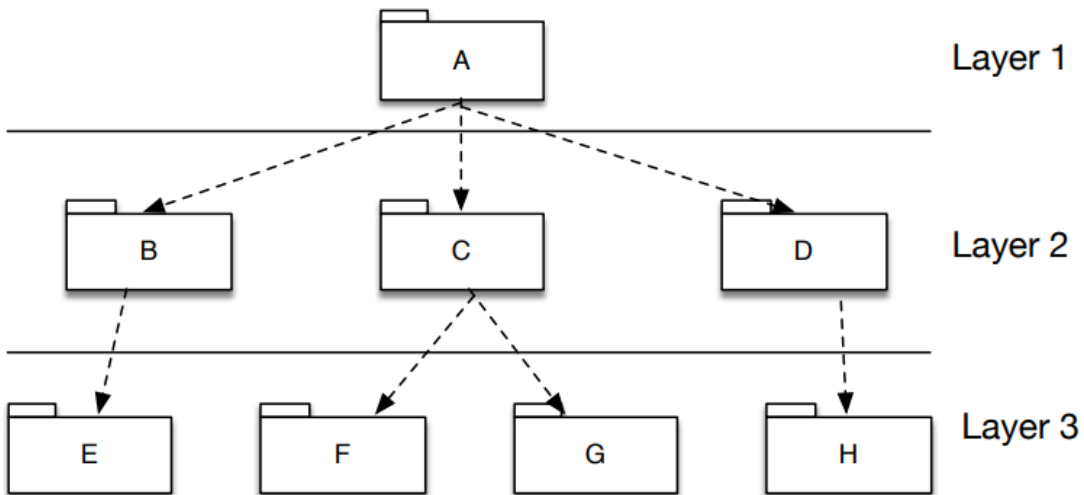
Solution/Modification: The ATM system will keep a counter for the number of times there was an attempt to access an account per day. If the counter exceeds the max value, the ATM card is locked within the machine and security is called.

Modification Steps :

1. **Detect Attacks:** The ATM machine checks for intrusions(more than max attempts).
2. **Resist Attacks:** : The ATM machine temporary blocks the stolen card
3. **React to Attacks:** : The ATM machine calls the security personnel
4. **Recover from Attacks:** The ATM machine sends an alert message to the branch manager and the user whose card was stolen.

Question 7

Show how you will do the integration testing on the above layered system using big bang, top-down, bottom up, sandwich and modified sandwich approaches.

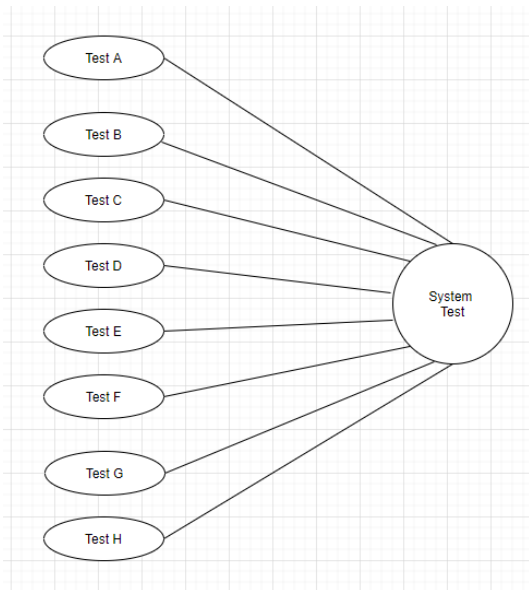


Integration testing :

1. **Big bang :** In this integration testing, all the components are integrated together at once and then tested as a whole unit. Unless all components are completed, this testing won't work.

All modules are integrated and then testing is performed.

So, A, B, C, D, E, F, G, H : all tested together.



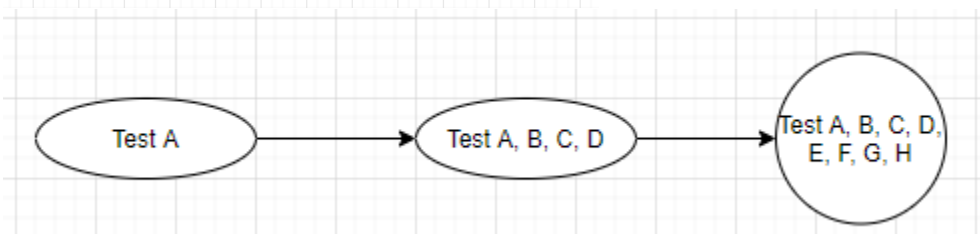
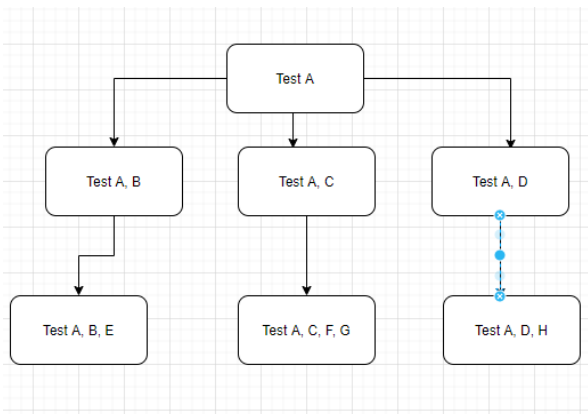
What I am trying to show is, that this is a big bang testing where they are all tested together.

2. **Top-down** : As the name suggest, we start the testing from the topmost level components(Level 1 - A) and then combine all the subsystems that are called by the tested subsystems and test the resulting collection of subsystems.

Process A is tested

Process B, C, D are integrated and tested by the Process A. (Layer 1 + 2)

All layers – A, B, C, D, E, F, G, H



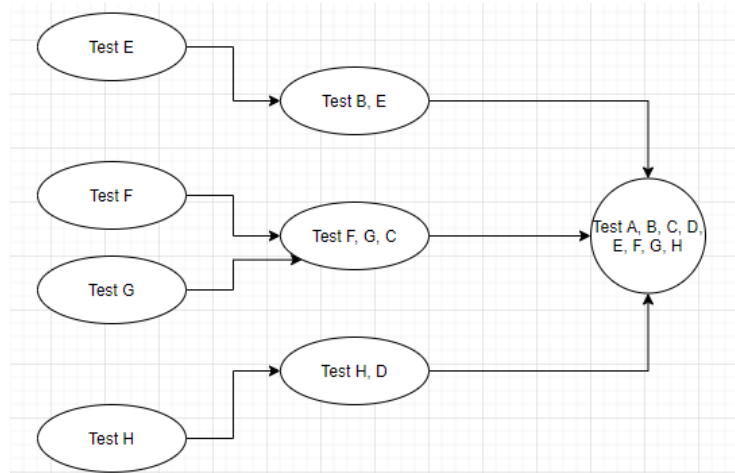
3. **Bottom up** : As the name suggests, the modules will be tested from the bottom to the up. So, the lowest level (Level 3) components (E, F, G, H) are tested first and then used to facilitate the testing of higher-level components. This goes on until we reach the top of the hierarchy.

Process F, G by Process C

Process E by Process B

Process H by Process D

Process E, B, F, G, C, H, D by Process A

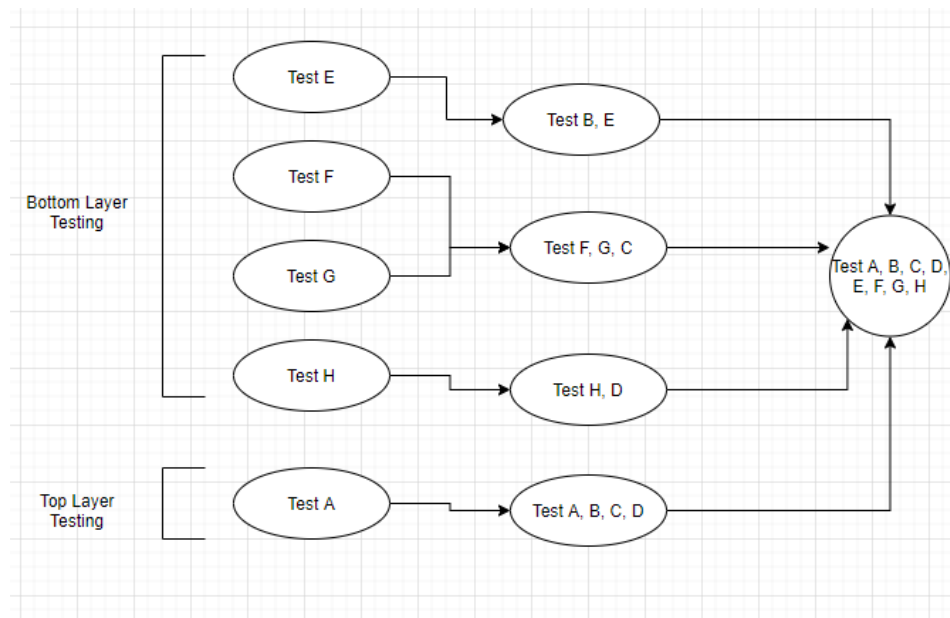


4. **Sandwich approaches** : This approach is a combination of top down and bottom-up approaches. There are three layers : A target layer in middle, one above target and other below target.

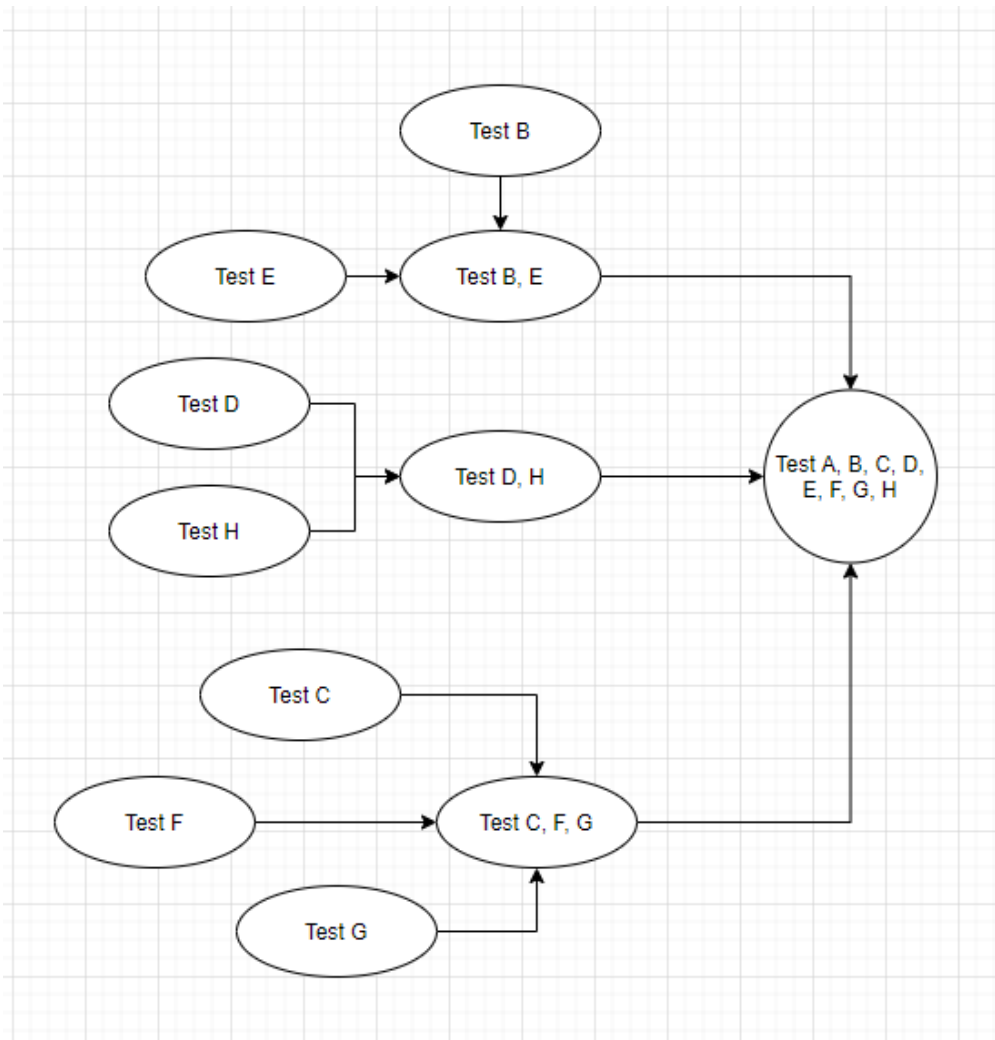
We do a bottom layer testing for layer 3 – E, F, G, H

Top layer testing for layer 1 – A

And then middle layer – layer 2 – B, C, D



5. **Modified sandwich approaches** : Modified sandwich approach is a better version of sandwich approach. It tries to take care of the drawbacks of the sandwich approach. A simultaneous testing takes place by using stubs and drivers. Tests can be done in parallel – middle layer with stubs and drivers, top layer with stubs and bottom with drivers. Then, top layer replaces the drivers when accessed by middle layer and bottom layer replaces stubs when accessed by middle layer.

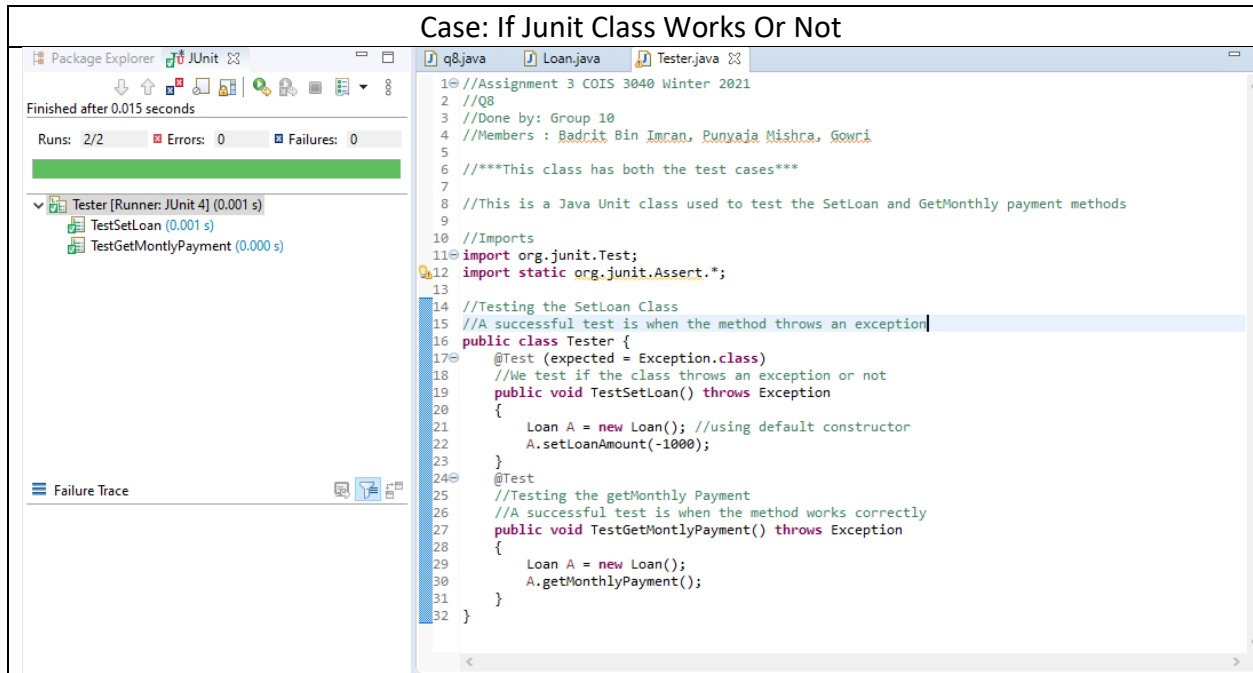


Question 8

JUnit Code: The two test classes are submitted in one java file.

Testing:

Case: If JUnit Class Works Or Not



The screenshot shows an IDE with the JUnit test runner on the left and the source code on the right. The test runner indicates that the tests passed successfully. The source code is a Java file named `Tester.java` that contains two test methods: `TestSetLoan` and `TestGetMontlyPayment`. Both methods use the `Loan` class and assert that the expected behavior is met.

```
1 //Assignment 3 COIS 3040 Winter 2021
2 //Q8
3 //Done by: Group 10
4 //Members : Badrit Bin Imran, Punyaja Mishra, Gowri
5
6 /**This class has both the test cases**
7
8 //This is a Java Unit class used to test the SetLoan and GetMonthly payment methods
9
10 //Imports
11 import org.junit.Test;
12 import static org.junit.Assert.*;
13
14 //Testing the SetLoan Class
15 //A successful test is when the method throws an exception
16 public class Tester {
17     @Test (expected = Exception.class)
18     //We test if the class throws an exception or not
19     public void TestSetLoan() throws Exception
20     {
21         Loan A = new Loan(); //using default constructor
22         A.setLoanAmount(-1000);
23     }
24     @Test
25     //Testing the getMonthly Payment
26     //A successful test is when the method works correctly
27     public void TestGetMontlyPayment() throws Exception
28     {
29         Loan A = new Loan();
30         A.getMonthlyPayment();
31     }
32 }
```