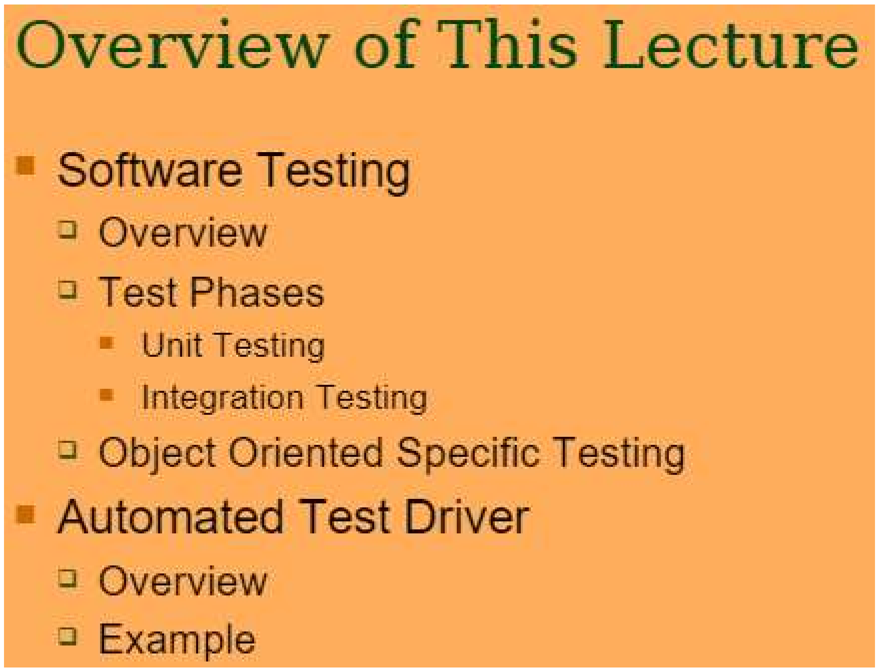
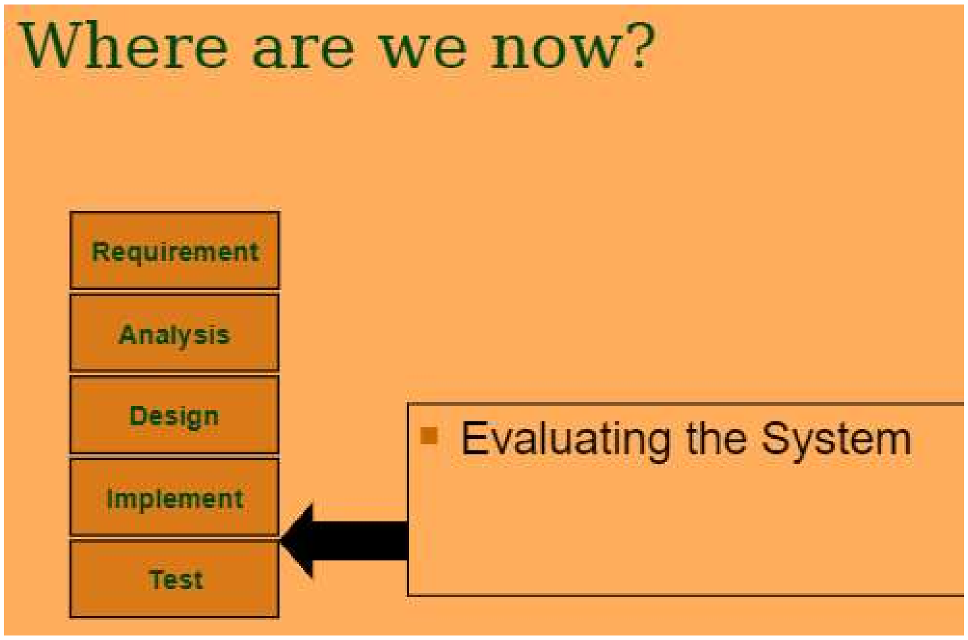
Lecture 12

Friday, April 07, 2023 7:23 PM



Overview of This Lecture

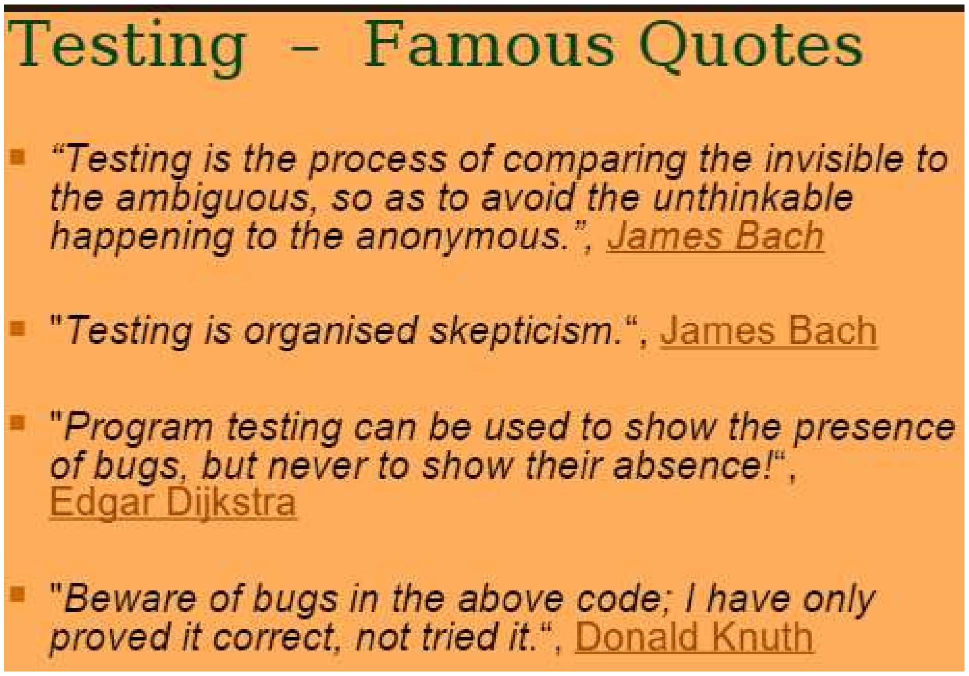
- Software Testing
  - Overview
  - Test Phases
    - Unit Testing
    - Integration Testing
  - Object Oriented Specific Testing
- Automated Test Driver
  - Overview
  - Example

-
- 

Where are we now?

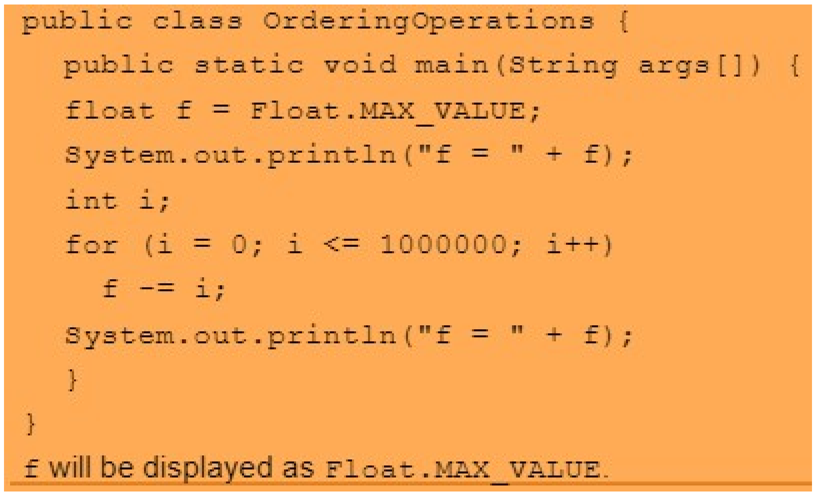
- Evaluating the System

-

- 

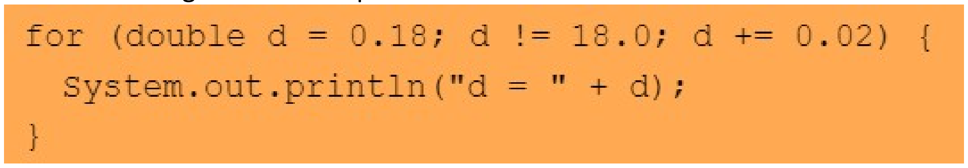
  Testing – Famous Quotes

  - *“Testing is the process of comparing the invisible to the ambiguous, so as to avoid the unthinkable happening to the anonymous.”*, James Bach
  - *"Testing is organised skepticism.“*, James Bach
  - *"Program testing can be used to show the presence of bugs, but never to show their absence!“*, Edgar Dijkstra
  - *"Beware of bugs in the above code; I have only proved it correct, not tried it.“*, Donald Knuth
- Testing: Definitions
  - "process of establishing confidence that a program or system does what it is supposed to"
  - "process of executing a pgrm/ sys w/ intent of finding errors"
  - "in any activity aimed at evaluating an attribute or capability of a pgrm / sys and determining that it meets its required results"
- Testing: Overview
  - Software test process based on well-defined software quality ctrl and testing standards, testing methods, strategy, test criteria, and tools
  - Engineers perform all types of software testing activities to perform a software test process
  - Last quality checking point for software is on its production line
- Testing: Objectives
  - Uncover as many errors (bugs) as possible in given time
  - Demonstrate that given software product matches its requirement specifications
  - Make high quality test cases, perform effective tests, and issue correct and helpful problem reports
  - Uncover errors (defects) in the software including errors in
    - Requirements from requirement analysis
    - Design documented in design specifications
    - Coding (implementation)
    - System resources and system environment
    - Hardware problems and their interfaces to software
- Ex.s of pgrm defect
  - One defect due to large numb that don’t store enough significant figures to be able to accurately represent the result

- 

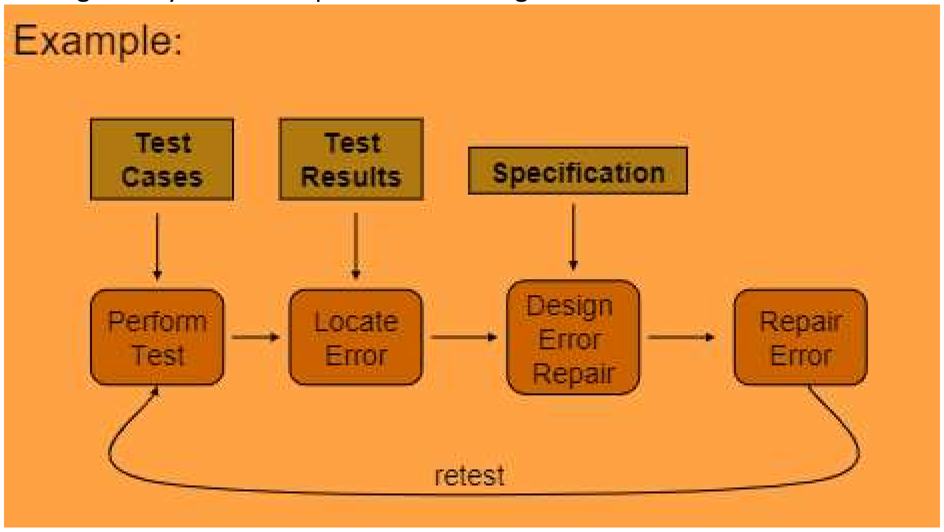
```
public class OrderingOperations {
  public static void main(String args[]) {
  float f = Float.MAX_VALUE;
  System.out.println("f = " + f);
  int i;
  for (i = 0; i <= 1000000; i++)
    f -= i;
  System.out.println("f = " + f);
  }
}
```

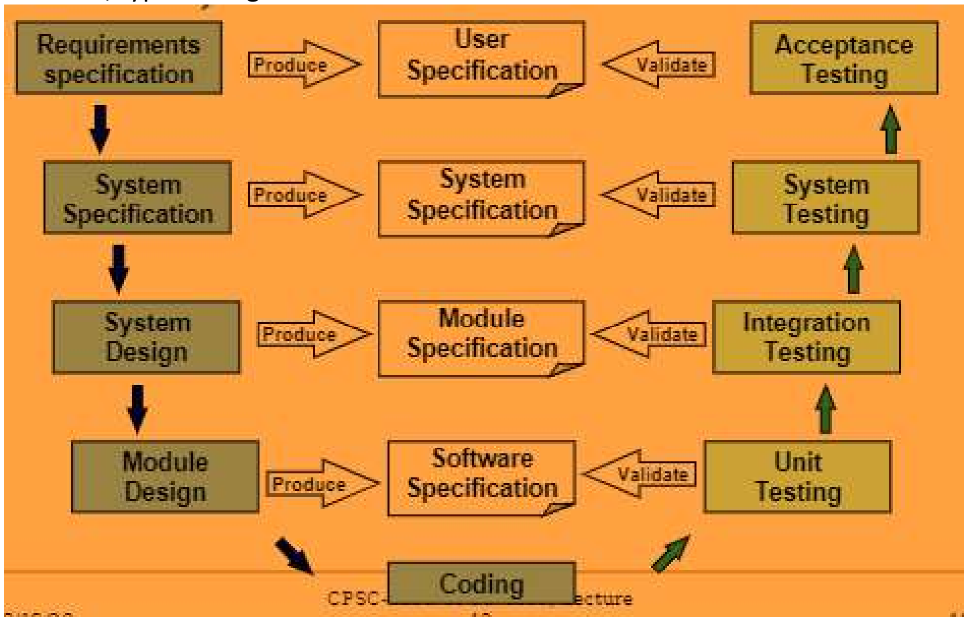
f will be displayed as `Float.MAX_VALUE`.

- Another would be numerical algorithms assuming that a floating point value will be exactly equal to some other value
- Below code frag is infinite loop bc 18.0 is missed
- 

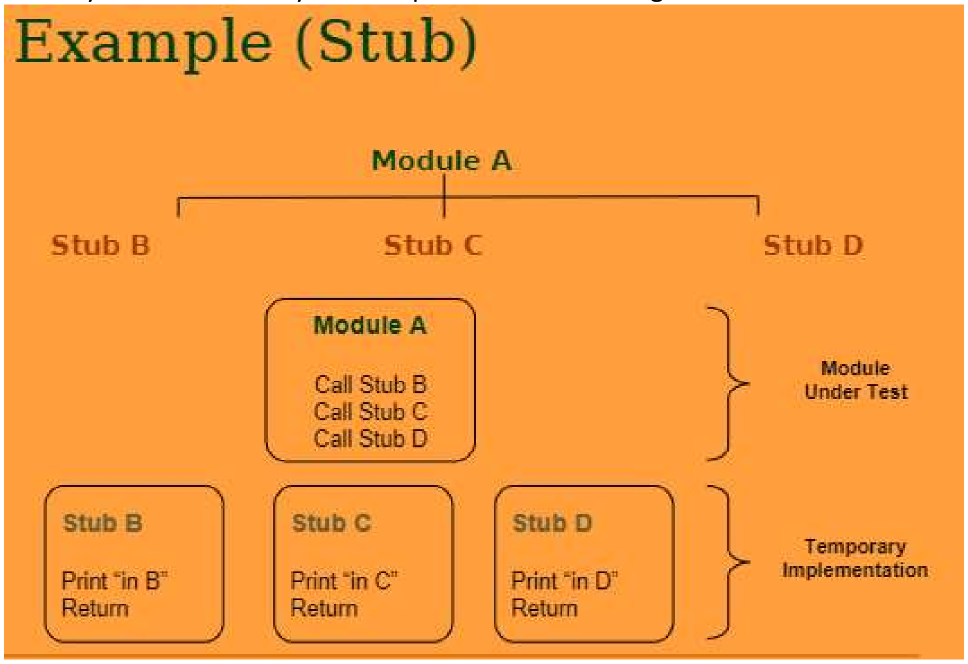
```
for (double d = 0.18; d != 18.0; d += 0.02) {
  System.out.println("d = " + d);
}
```

- Testing: Personnel
  - Test manager: manage, supervise, ctrl a software test project, define and specify a test plan
  - Software test engineers and testers: define test cases, write test specifications, run tests
  - Development engineers: only perform unit tests and integration tests
  - Quality assurance group and engineers: perform system testing, define software testing standards and quality control process
  - Independent Test group
- Testing: Activities
  - Test planning:
    - a test schedule for a test process and its activities, as well as assignments
    - Test requirements and items
    - Test strategy and supporting tools
  - Test Design and Specification:
    - Conduct the software design based on well-defined test generation methods
    - Specify test cases to achieve a targeted test coverage
  - Test set up
    - Testing tools and environment set-up
    - Test suite set up
  - Test operation and execution
    - Run test cases manually or automatically
  - Test result analysis and reporting
    - Report software testing results and conduct test result analysis
  - Problem Reporting
    - Report pgrm errors using a systematic solution
  - Test management and measurement
    - Manage software testing activities, ctrl testing schedule, measure testing complexity and cost
  - Test automation
    - Define and develop software test tools

        - Adopt and use software test tools
        - Write software test scripts
    - Test config management
        - Manage and maintain diff versions of software test suites, test environment and tools, and documents for various product versions
- Testing: Cycle of Activities
    - Testing usually involves repetition of testing activities
    - 

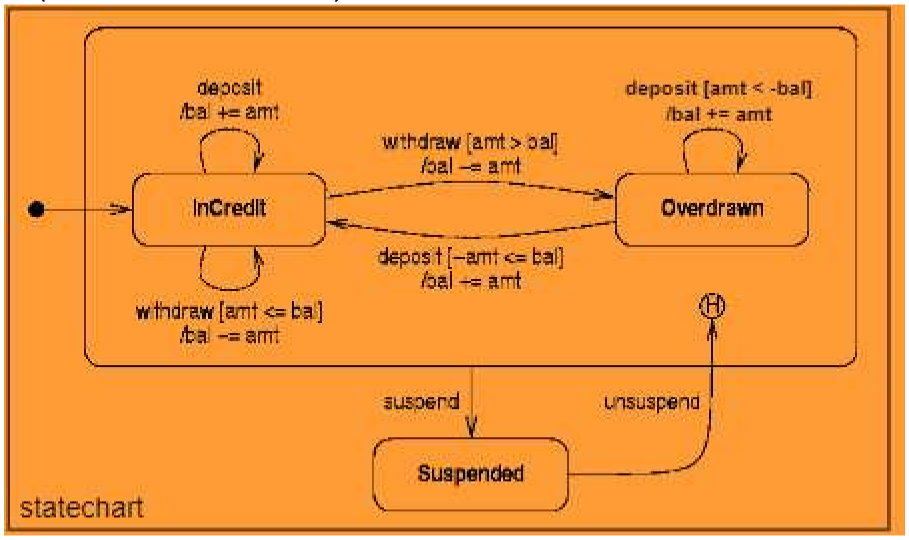
- Test Organization
    - In large software system, testing has to be carried out in multiple stages
    - Test stages linked to the various software development stages
    - Ex: acceptance test is linked to the requirement and the specification stages
    - V-model, type of diagram
    - 

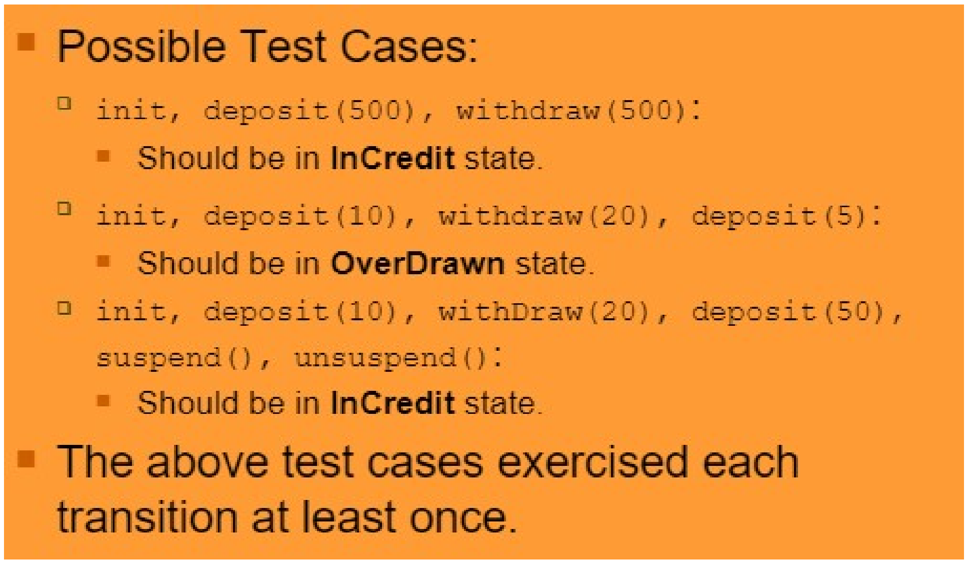
- Test Stage: Unit Testing
    - Testing individual modules
        - Methods/fn
        - Classes
        - Sometimes class clusters
    - Based on info abt the structure of a code frag
    - Carried out by dev engineer
    - Objective to test that the unit performs the fn for which designed

    - Unit tests can be designed b4 coding begins, or just after src code made
- Unit Testing: Check List
    - Interface: ensure information properly flows in and out
    - Data Structure: ensure the stored data maintains integrity
    - Boundary conditions: module operates properly at boundaries
    - Independent paths: all paths thru the ctrl structure are exercised
    - Error handling paths: paths that handle errors are exercised
- Unit Testing: Test Setup
    - Test Case:
        - Test data, should be coupled w/ expected results
        - Often lists Test identifier, test objectives, test resources, test procedure
    - Driver:
        - Module that has the test case data, passes it to component under testing, and logs the results
    - Stub:
        - Dummy module called by the component under testing
        - 

- Generating Unit Test Cases
    - Statement coverage
    - Graph based:
        - Branch coverage
        - Condition coverage
        - Path coverage
    - Also applicable to testing methods in a class
- Unit Testing: Object-Oriented Code
    - Class lvl testing:
        - Operations in the class are the smallest testable units
        - Testing a single operation in isolation is difficult
        - Unit test generally driven by the structure of methods of the class and the state behavior of that class
    - Recall state-transition model, state of an object
    - public method in a class can be tested using a black-box approach
        - Start from the specification - class interface

- Consider each param in method signature, and id its possible data range(s)
  - Equivalence classes- black box test approach
- Incorporate pre-conditions and post-conditions in the test of a method
- Test exceptions
- For complex logic, also use white-box testing/ static testing
- For private methods
  - Either mod the class (temporarily) so it can be tested externally – change access to public
  - Or incorporate a test driver in the class
  - Or use static test methods, like pgrm tracing

- Using a Statechart
  - Make test cases corresponding to each transition path that reps a full object life cycle
  - Make sure each transition is exercised at least once
  - Ex (account from lecture 10):
  - 

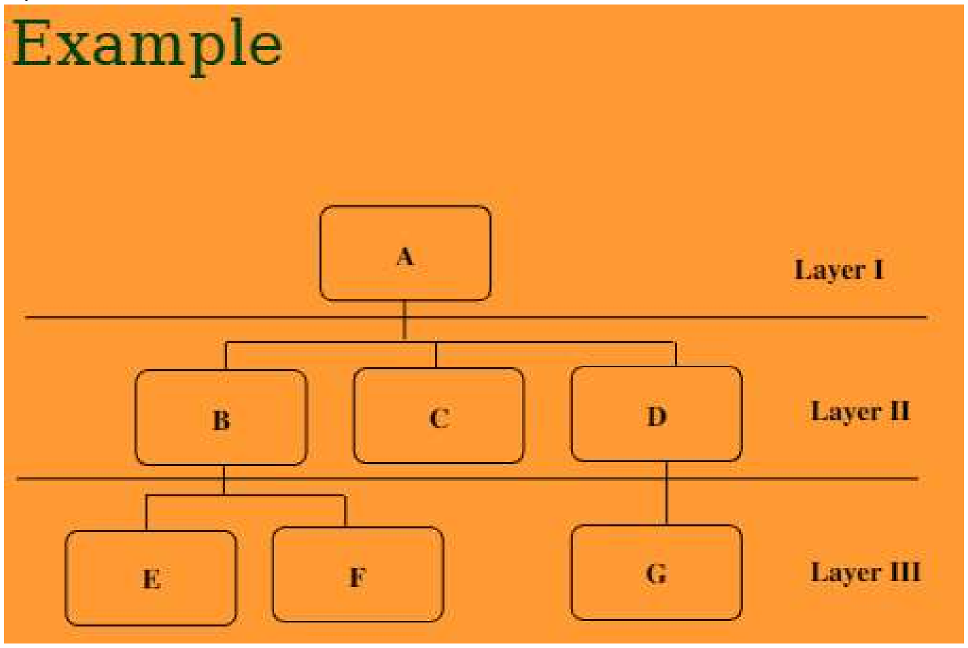
  - 

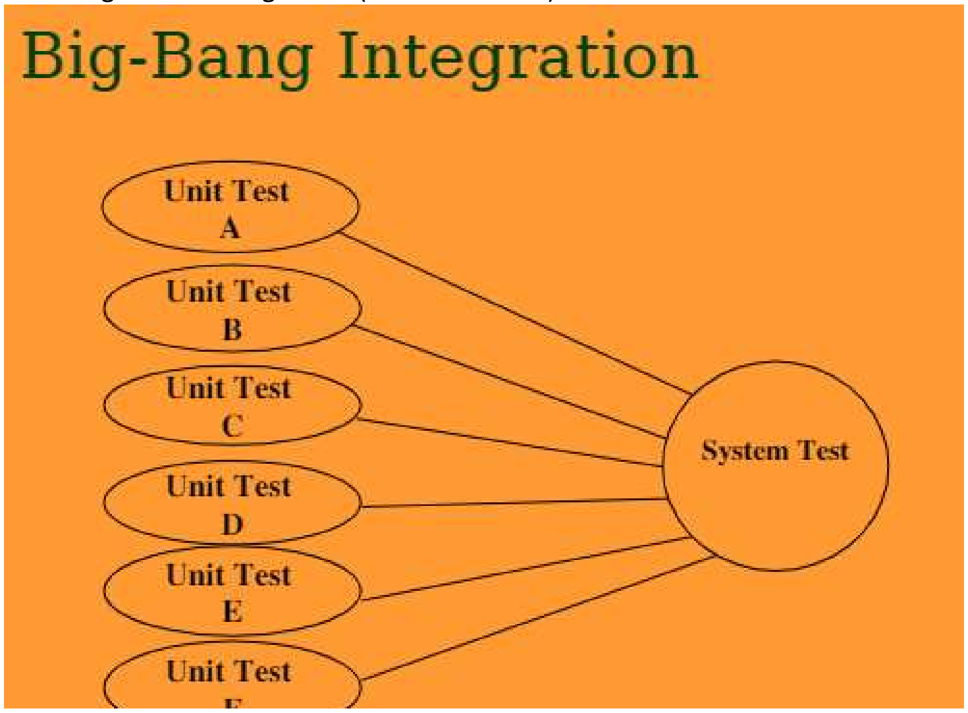
■ Possible Test Cases:

- `init, deposit(500), withdraw(500):`
  - Should be in **InCredit** state.
- `init, deposit(10), withdraw(20), deposit(5):`
  - Should be in **OverDrawn** state.
- `init, deposit(10), withDraw(20), deposit(50), suspend(), unsuspend():`
  - Should be in **InCredit** state.

■ The above test cases exercised each transition at least once.

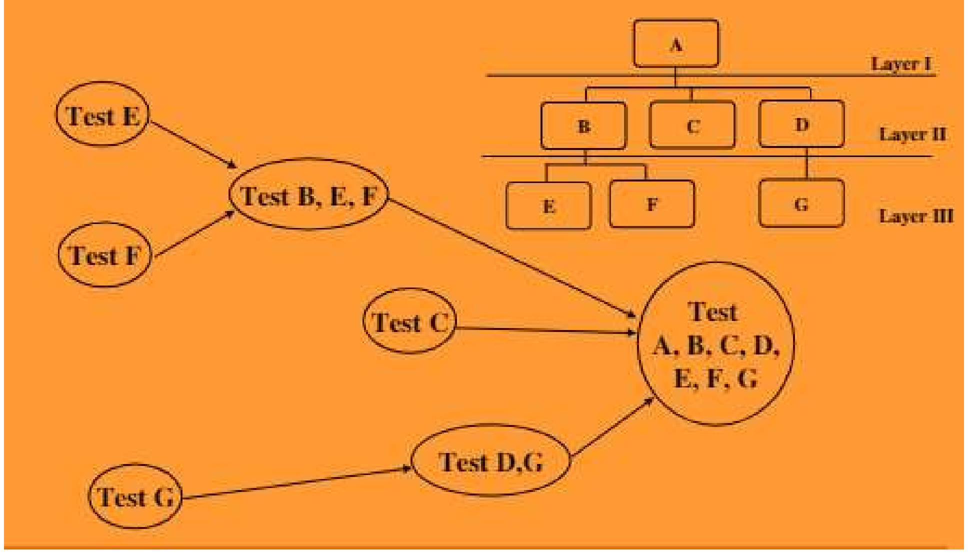
- Test State: Integration Testing
  - Entire sys viewed as collection of subsys (sets of classes) determined during the sys and object design
  - Order in which the subsys are selected for testing and integration determines the testing strat
  - Carried out by dev engineer

- Test that leads to construction of the complete software architecture
- Big bang integration: all components together
- Bottom up integration: from lower lvls, no test stubs needed
- Top down integration: from higher lvls, no test drivers needed
- Sandwich testing: combo of bottom-up and top-down, top layer w/ stubs and bottom layer w/ drivers
- 

- General Observations
  - Low lvl components usually perform more common tasks:
    - I/O
    - Repetitive calcs
    - Object oriented: usually entity classes
  - High lvl components usually direct the sys activities: controller
  - Summary on functionality:
    - Low lvl – more specific (utility modules)
    - High lvl – more general (ctrller modules)
  - 

○

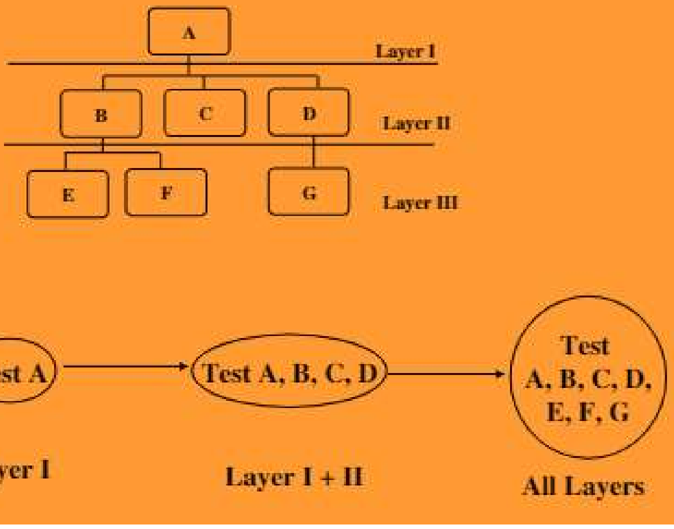
Bottom-Up Integration



○

○

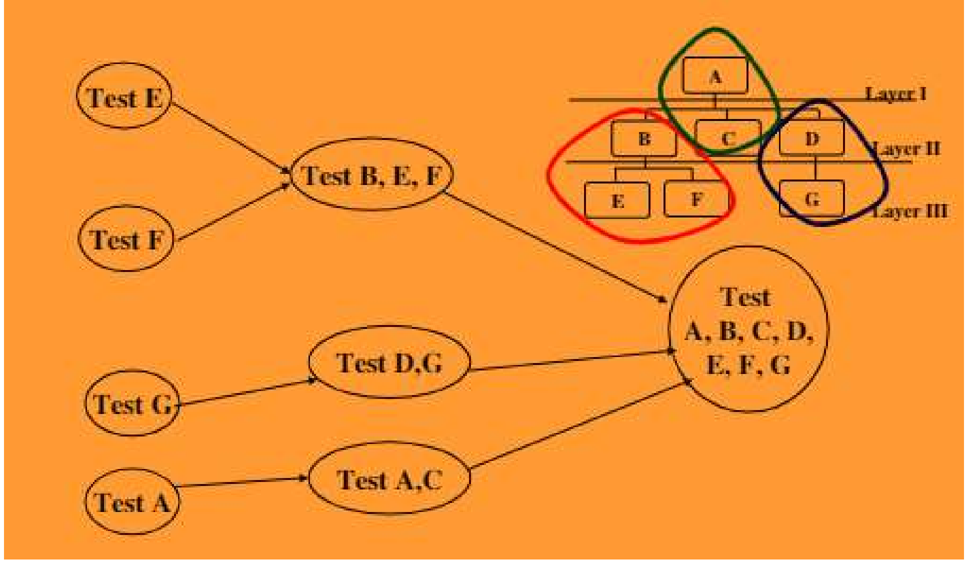
Top-Down Integration



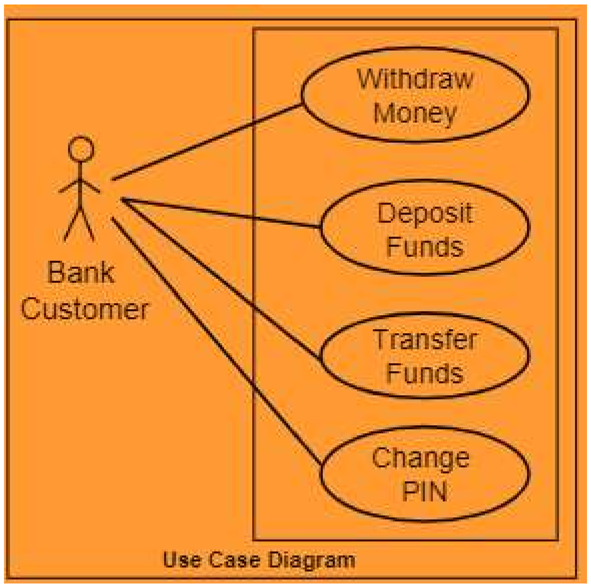
○

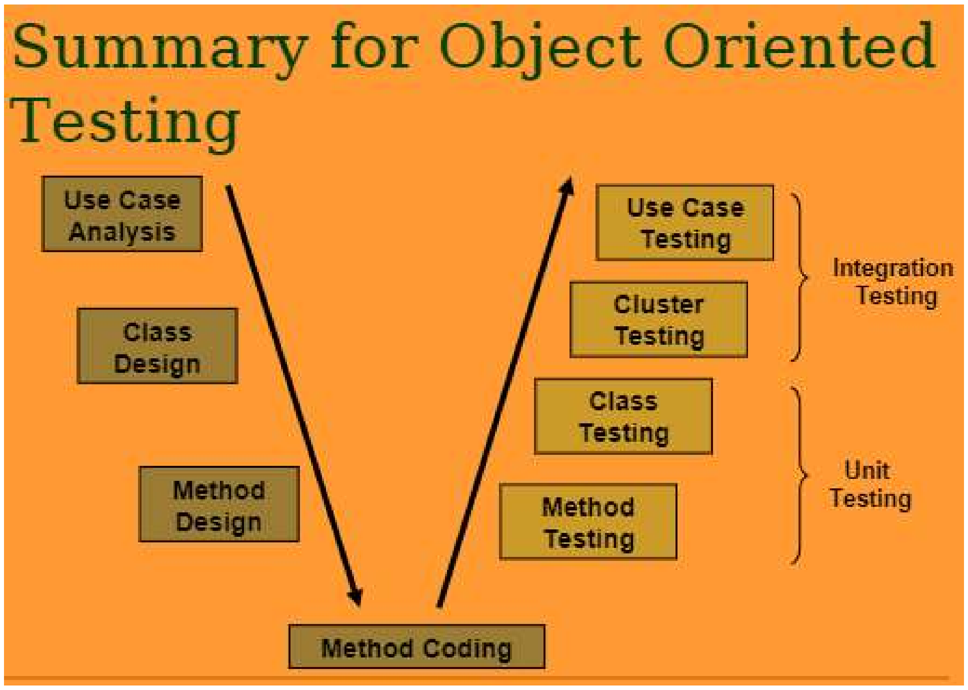
○

Sandwich Integration



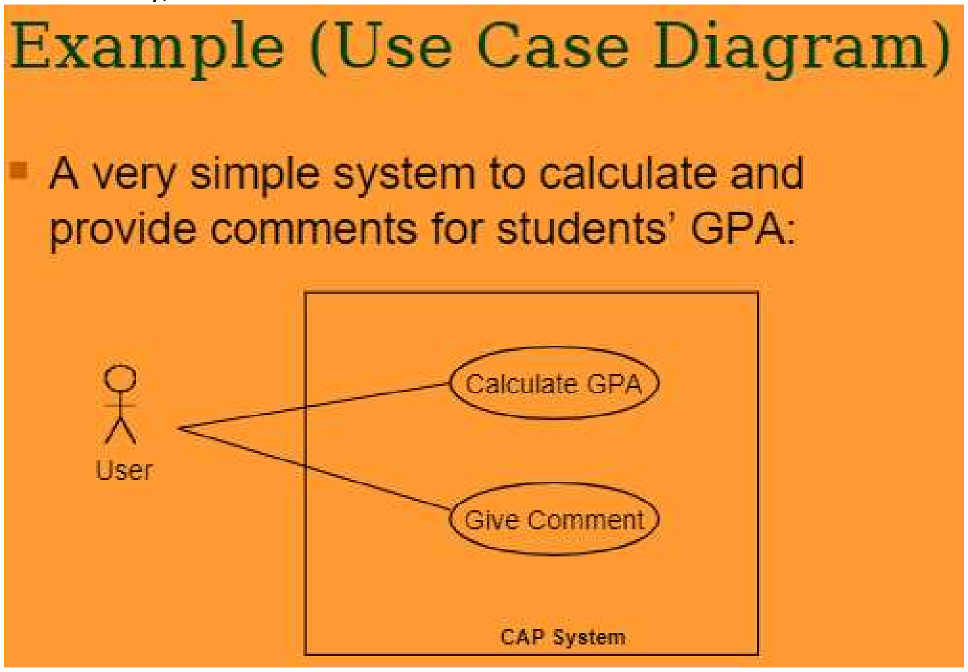
- Sandwich Integration
  - Combo of top-down and bottom-up approaches
  - Select a target lvl
    - For lvls above the target lvl, do top-down integration
    - For lvls below the target lvl, do bottom-up
  - Exercise both Controller modules and Utility modules at same time
- Integration Test: Object Oriented Code
  - Class clusters
    - Classes that are tightly coupled are good candidates for increment integration
  - Candidate class clusters
    - Classes in a package
    - Classes in a class hierarchy
    - Classes associated w/ the interaction diagram for a use case
  - Use based testing:
    - Group the classes- independent / dependent
      - Begin by testing independent classes
      - Then test dependent classes which use independent classes
- Class Cluster: Using use cases
  - Atm:
    - Use cases:
      - Withdraw money
      - Deposit funds
      - Transfer funds
      - Change PIN

            - Actor
                - Bank customer
        - 

-
- 

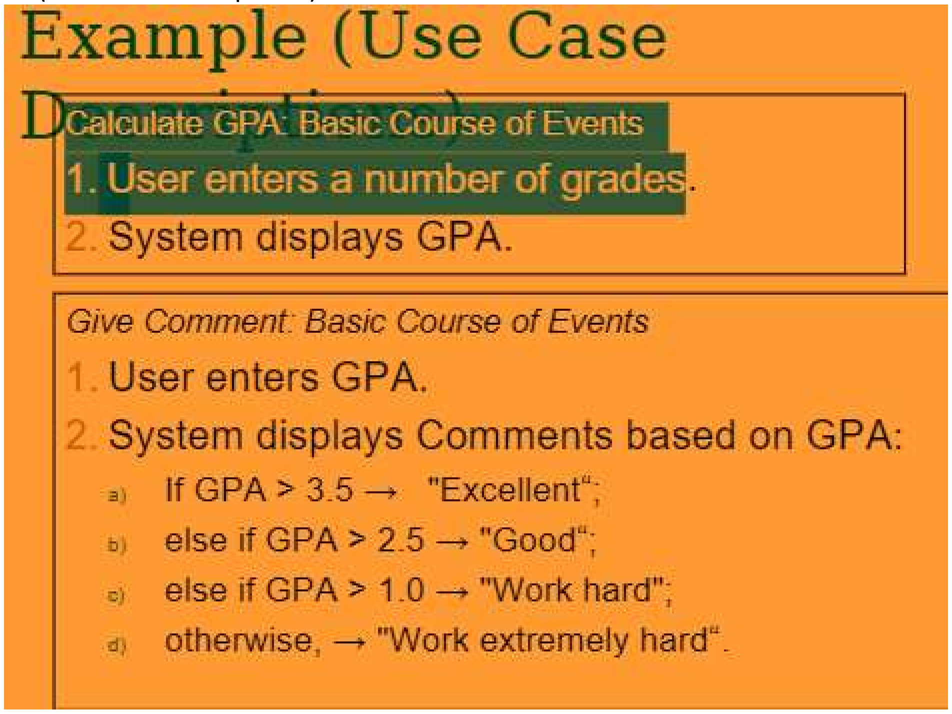
- Test Stage: System Testing
    - Performed exclusively in terms of I/O of the sys
    - Performed mostly on the target platform
    - Thread-based:
        - The behavior that results from a sys lvl input
        - An interleaved sequence of sys inputs (stimuli) and outputs (responses)
        - Depicts a possible scenario of using the sys
- Test Stage: Testing that Involves Users
    - Alpha testing:
        - In-house testing
        - By a test team/end users
    - Beta testing
        - By users/selected subset of actual customers in a norm work environment
        - Product very close to completion

        - Open beta release, let public carry out beta testing
    - Acceptance testing:
        - By users to check that the sys satisfies reqs to decide whether to accept the sys based on the test result
- Test Case Design: Overview
    - Black box vs White box approach
    - Black box testing
        - Knowing the specified fn a component has been designed for
        - Tests conducted at the interface of the component
    - White box testing:
        - Knowing the internal workings of a component
        - Test cases exercise specific sets of condition, loops, …
    - Another way to classify the test cases:
        - Test to specifications
            - Aka black-box / data driven / functional / I/O driven testing
            - Ignore the code – use the specifications to select test cases
        - Test to code:
            - Aka glass-box / logic driven / structured / path-oriented testing
    - Neither exhaustive testing to specifications nor exhaustive testing to code is feasible
- Project : Test Phase
- Reqs for Test Phase
    - Automated test driver
        - Read test cases from a file
        - Put them thru the sys
        - Capture and validate output
    - Test case Documentation
        - Based on use cases (black box)
        - 3 tests per Use Case
            - 1 valid, 2 invalid
    - Validity should be based on functionality
- Automated test driver
    - In essence, automated test driver replaces the boundary classes in ur sys
        - Read directly from a file instead of reading input from actual user
        - Get output directly from the sys and place them into a file instead of showing on screen
- Automated Test Driver: Input File
    - Input file:
        - Txt base: facilitate mods of test cases
        - Has numb of test cases
    - Each of the test cases has
        - Method to be tested
        - Params for the method
        - Expected output
        - Comments (optional)
- Automated Test Driver: Output File
    - Output file
        - Txt base: easy for human inspections
        - Test case result
    - Each test case results has

        - Output of the method (if any)
        - Common categories of the test result
            - Pass: Test result == expected result
            - Failed: Test result != Expected result
            - Human check: more complicated cases where the expected result is hard/impossible to be defined
- Test Case Design
    - Automated test driver can be used to test in diff test phases
        - Unit testing, integration testing, …
    - Usually, test cases should be derived from Use cases
- 

Example (Use Case Diagram)

- A very simple system to calculate and provide comments for students' GPA:

- Ex (Use Case Descriptions)
- 

Example (Use Case Descriptions)

Calculate GPA: Basic Course of Events

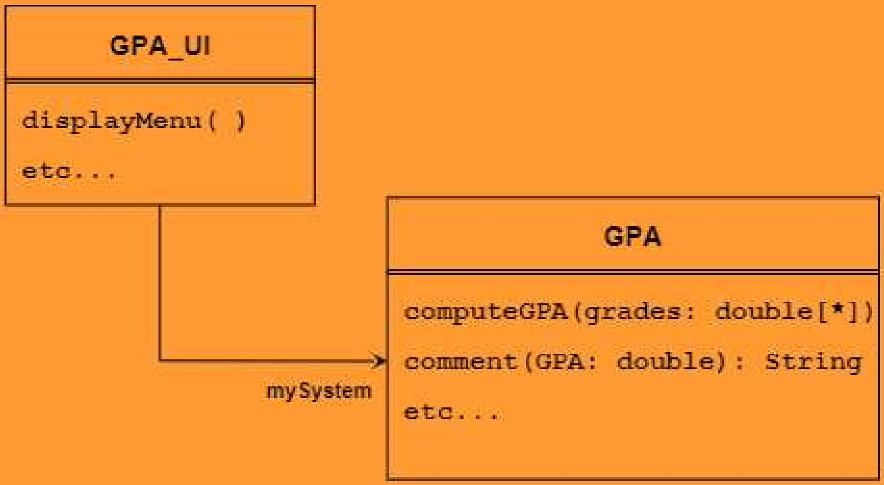
1. User enters a number of grades.
2. System displays GPA.

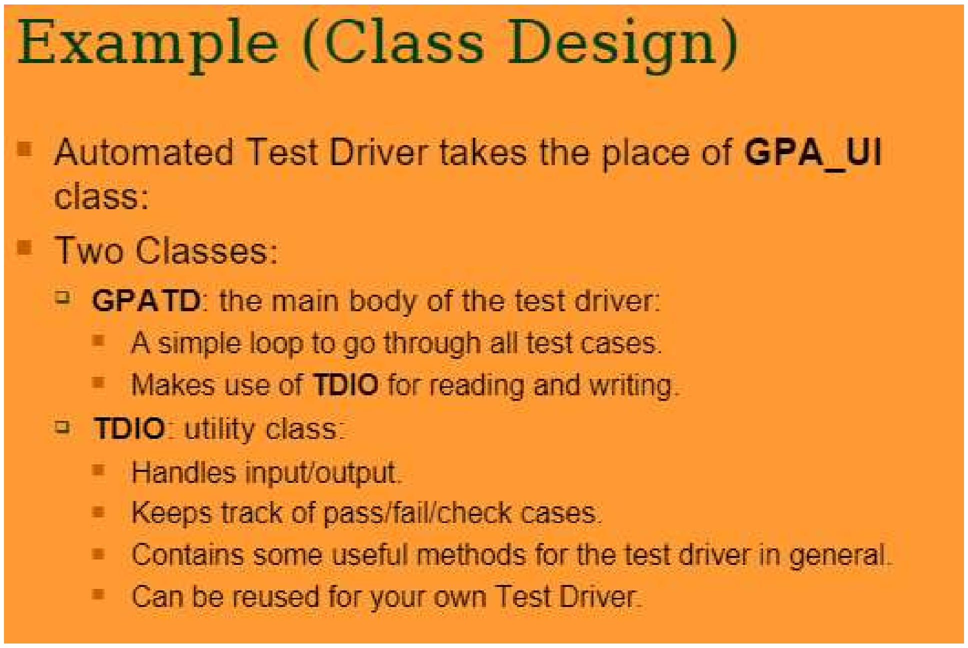
Give Comment: Basic Course of Events

1. User enters GPA.
2. System displays Comments based on GPA:
    a) If GPA > 3.5 → "Excellent";
    b) else if GPA > 2.5 → "Good";
    c) else if GPA > 1.0 → "Work hard";
    d) otherwise, → "Work extremely hard".

-

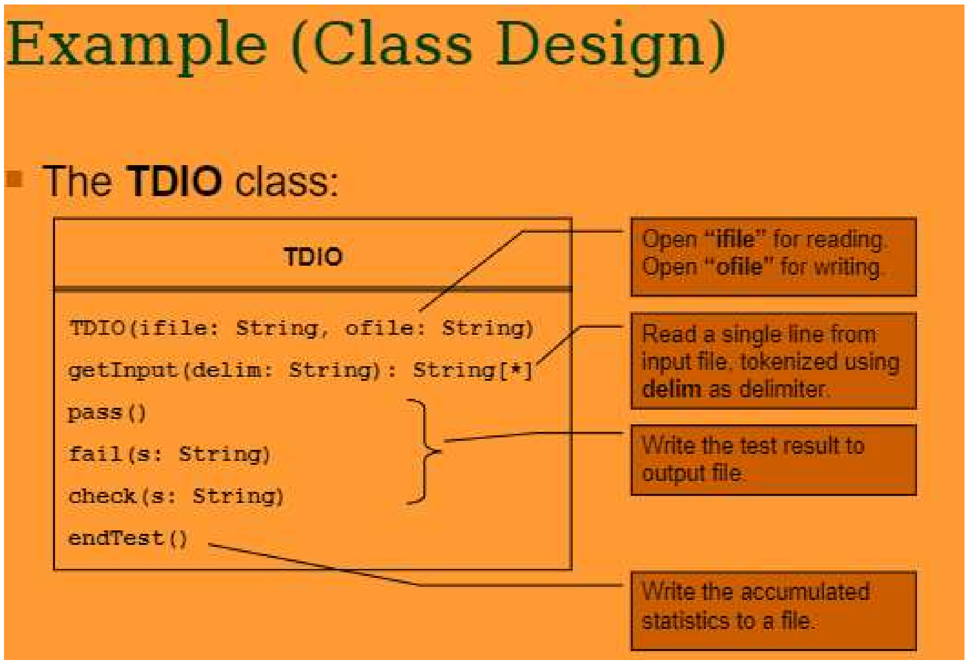
Example (Class Diagram)





Example (Class Design)

- Automated Test Driver takes the place of **GPA_UI** class:
- Two Classes:
  - **GPATD**: the main body of the test driver:
    - A simple loop to go through all test cases.
    - Makes use of **TDIO** for reading and writing.
  - **TDIO**: utility class:
    - Handles input/output.
    - Keeps track of pass/fail/check cases.
    - Contains some useful methods for the test driver in general.
    - Can be reused for your own Test Driver.

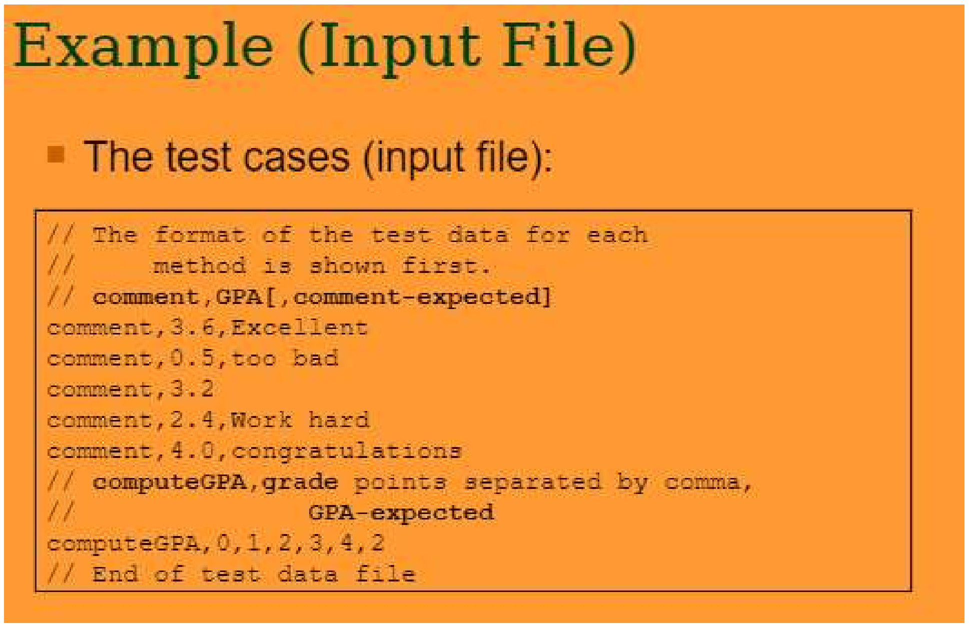


Example (Class Design)

- The **TDIO** class:

```
TDIO
TDIO(ifile: String, ofile: String)
getInput(delim: String): String[*]
pass()
fail(s: String)
check(s: String)
endTest()
```

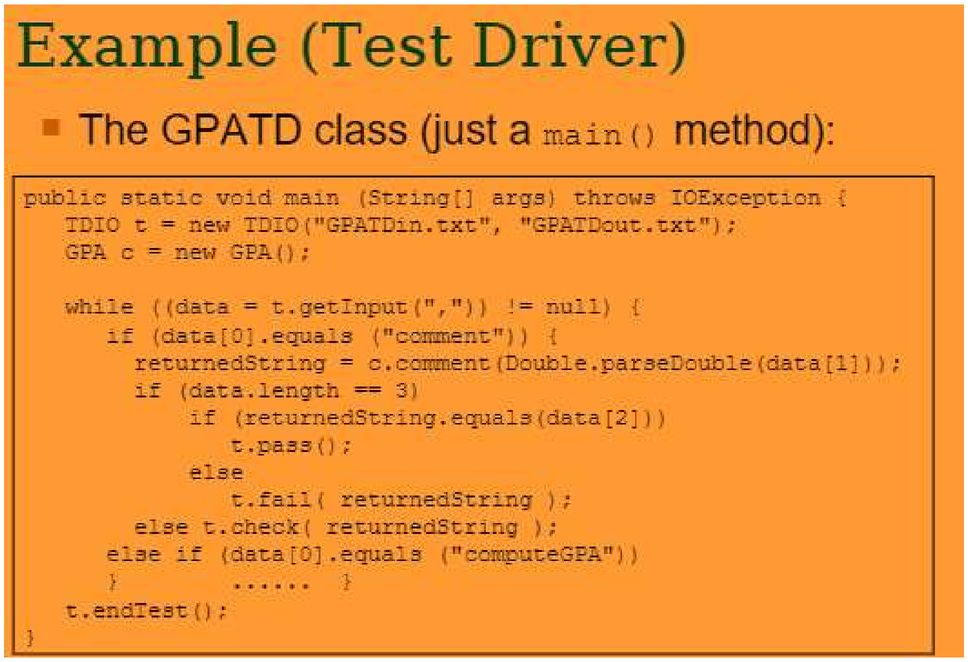
Open **“ifile”** for reading.
Open **“ofile”** for writing.
Read a single line from input file, tokenized using **delim** as delimiter.
Write the test result to output file.
Write the accumulated statistics to a file.



Example (Input File)

- The test cases (input file):

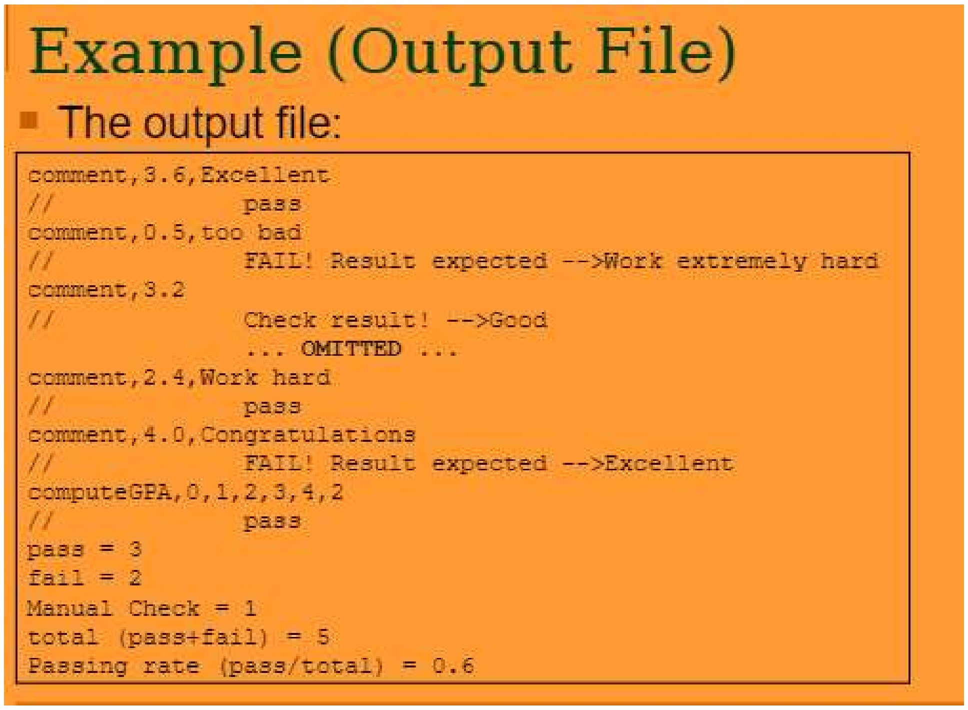
```
// The format of the test data for each
//     method is shown first.
// comment,GPA[,comment-expected]
comment,3.6,Excellent
comment,0.5,too bad
comment,3.2
comment,2.4,Work hard
comment,4.0,congratulations
// computeGPA,grade points separated by comma,
//                  GPA-expected
computeGPA,0,1,2,3,4,2
// End of test data file
```

- 

Example (Test Driver)

- The GPATD class (just a `main()` method):

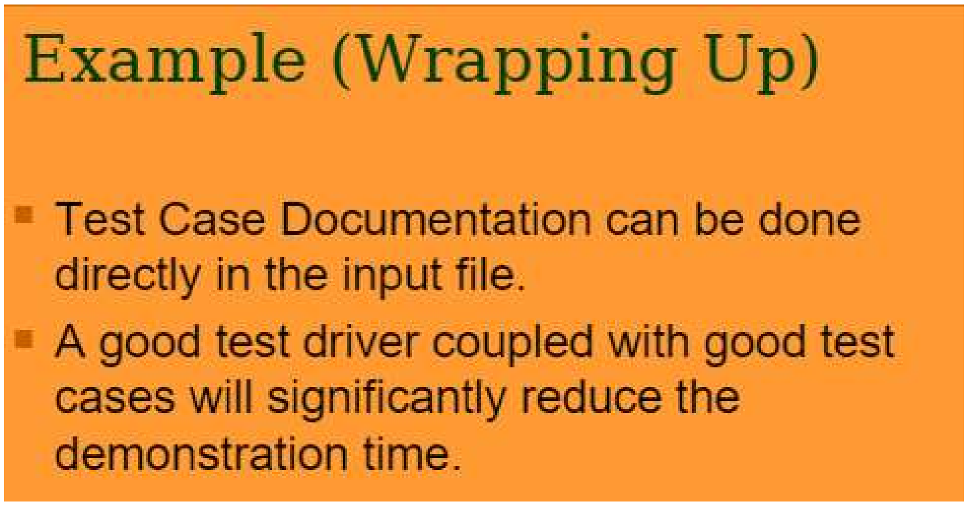
```
public static void main (String[] args) throws IOException {
   TDIO t = new TDIO("GPATDin.txt", "GPATDout.txt");
   GPA c = new GPA();

   while ((data = t.getInput(",")) != null) {
      if (data[0].equals ("comment")) {
        returnedString = c.comment(Double.parseDouble(data[1]));
        if (data.length == 3)
            if (returnedString.equals(data[2]))
               t.pass();
            else
               t.fail( returnedString );
        else t.check( returnedString );
      else if (data[0].equals ("computeGPA"))
      }        ......  }
   t.endTest();
}
```

- 

Example (Output File)

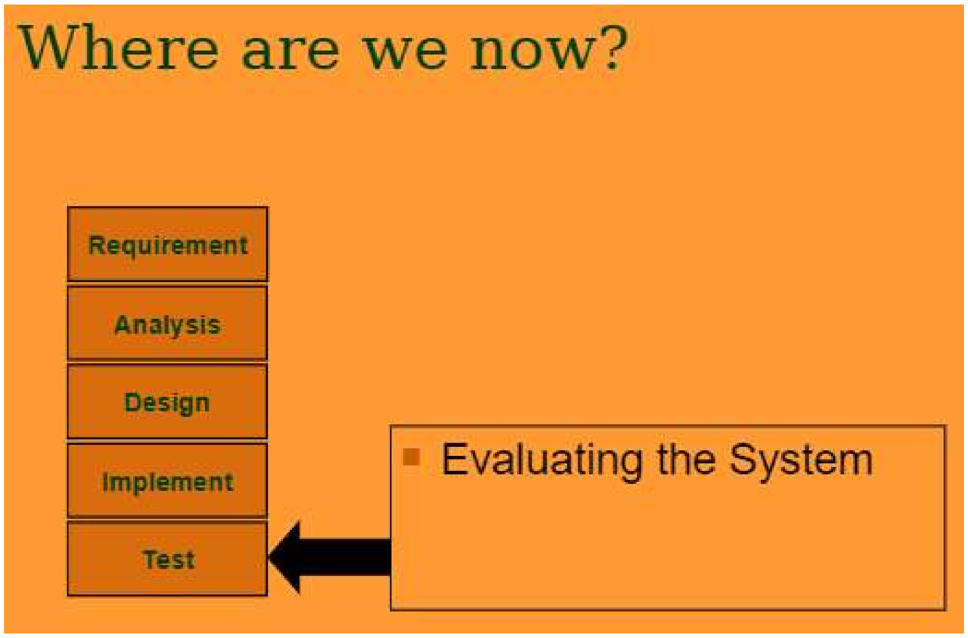
- The output file:

```
comment,3.6,Excellent
//              pass
comment,0.5,too bad
//              FAIL! Result expected -->Work extremely hard
comment,3.2
//              Check result! -->Good
                ... OMITTED ...
comment,2.4,Work hard
//              pass
comment,4.0,Congratulations
//              FAIL! Result expected -->Excellent
computeGPA,0,1,2,3,4,2
//              pass
pass = 3
fail = 2
Manual Check = 1
total (pass+fail) = 5
Passing rate (pass/total) = 0.6
```

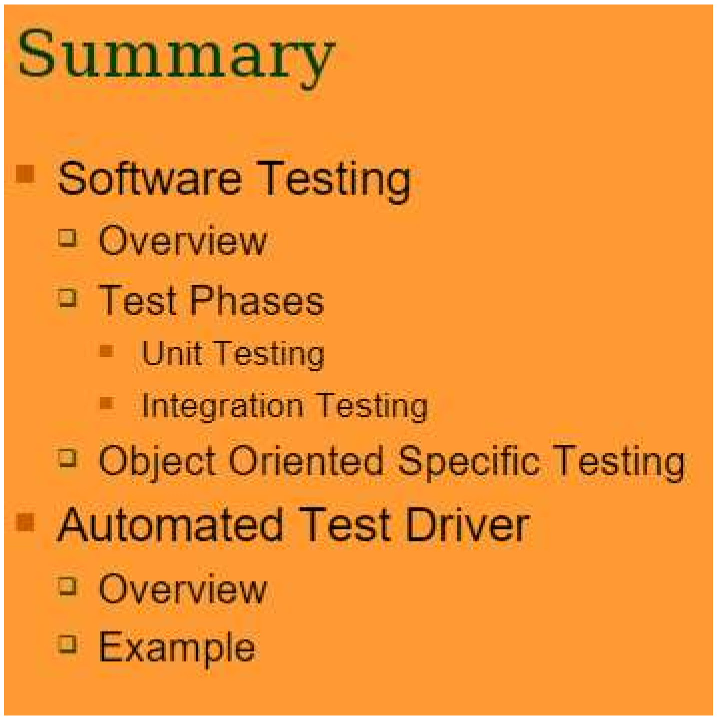
- 

Example (Wrapping Up)

- Test Case Documentation can be done directly in the input file.
- A good test driver coupled with good test cases will significantly reduce the demonstration time.

-
- 

  ## Where are we now?

  - Evaluating the System
- 

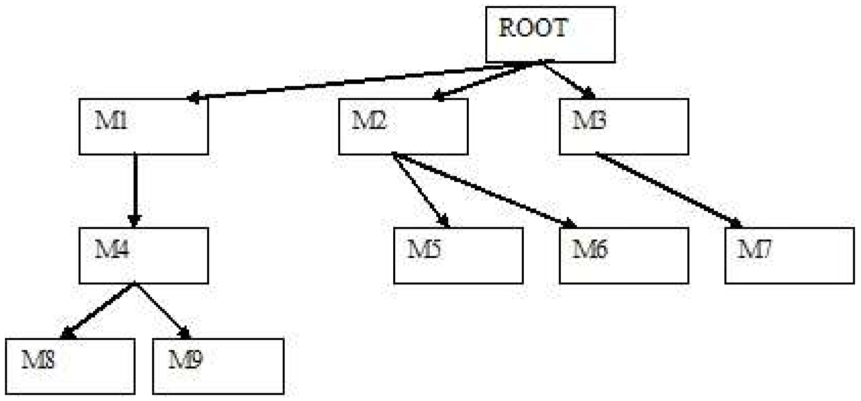
  ## Summary

  - Software Testing
    - Overview
    - Test Phases
      - Unit Testing
      - Integration Testing
    - Object Oriented Specific Testing
  - Automated Test Driver
    - Overview
    - Example
-
-
  -
-

L12 Assessment

Friday, April 07, 2023 8:43 PM

The structure chart below illustrates the hierarchy of modules in a software sub-system. Describe the sequence of module tests for integrating the modules using the depth first top-down approach.



For the top down approach, the tests would look like such:

Test ROOT -> Test ROOT, M1, M2, M3 -> Test ROOT, M1, M2, M3, M4, M5, M6, M7 -> Test ROOT, M1, M2, M3, M4, M5, M6, M7, M8, M9

Question 2

Explain why there exist defects in numerical algorithms. Give examples of few such defects in Java programs.

Algorithms that use float values as checkers almost always have defects because many assume that these values hold simple fractions or other particular values exactly.

An example of a loop that would never stop because of a float value is shown below.

```
float c = 0.1f;
While (c != 1){
    c += 0.1;
    System.out.println("YOU CAN'T STOP ME!!!!!   MWAHAHAHAHA");
    //other code
}
```

This code would never stop because the float value used to increment is never exactly going to give the expected value. In this case, the floating point value c would never reach the exact value 1, even though 0.1 added 10 times together does (in real life) equal 1.

Question 3

What is true regarding the black box and white box testing?

The black box and the white box testing are actually the same thing;
For black box testing, tests are conducted at the interface of the component;
For black box testing, tests know the internal workings of a component;
For white box testing, tests do not know the internal workings of a component, but know only the specified function a component has been designed for;
None of the above.

Question 4
10 Points

In Black Box Testing, a tester

Runs all the inputs to test the interface;
Uses specific groups to test the interface;
Uses only some values given by the client to test the interface;
Checks code to verify, which input can be applicable for a particular function;
None of the above.

Question 5
10 Points

What is the order of these activities in software testing?

1. integration testing
2. system testing
3. unit testing
4. acceptance testing

1, 4, 2, 3
3, 2, 1, 4
3, 1, 2, 4
4, 3, 2, 1