Week 1: OOP

https://nus-cs2103-ay2122s1.github.io/website/schedule/week1/topics.html

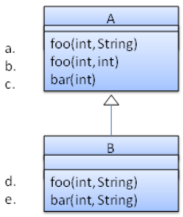
Choose the correct statement.

- [x] a. An object is an encapsulation because it packages data and behavior into one bundle.
- [ ] b. An object is an encapsulation because it lets us think in terms of higher level concepts such as Students rather than student-related functions and data separately.

(a)

Explanation: The second statement should be: An object is an **abstraction** ~~encapsulation~~ because it lets ...

Which of these methods override another method? `A` is the parent class. `B` inherits `A`.



- [ ] a
- [ ] b
- [ ] c
- [x] d
- [ ] e

d

Explanation: Method overriding requires a method in a *child class* to use the same method name and same parameter sequence used by one of its ancestors.

Which of these are suitable as class-level variables?

- [ ] a. system: multi-player Pac Man game, Class: `Player`, variable: `totalScore`
- [ ] b. system: eLearning system, class: `Course`, variable: `totalStudents`
- [x] c. system: ToDo manager, class: `Task`, variable: `totalPendingTasks`
- [ ] d. system: any, class: `ArrayList`, variable: `total` (i.e., total items in a given `ArrayList` object)

(c)

Explanation: `totalPendingTasks` should not be managed by individual `Task` objects and is therefore suitable to be maintained as a class-level variable. The other variables should be managed at instance level as their value varies from instance to instance. e.g., `totalStudents` for one `Course` object will differ from `totalStudents` of another.

Which are the benefits of exceptions?

- [x] a. Exceptions allow us to separate normal code from error handling code.
- [ ] b. Exceptions can prevent problems that happen in the environment.
- [x] c. Exceptions allow us to handle in one location an error raised in another location.

(a) (c)

Explanation: Exceptions cannot *prevent* problems in the environment. They can only be used to handle and recover from such problems.

Overridden methods are resolved using dynamic binding, and therefore resolves to the implementation in the actual type of the object.

Consider the code below. The declared type of `s` is `Staff` and it appears as if the `adjustSalary(int)` operation of the `Staff` class is invoked.

```
1 void adjustSalary(int byPercent) {
2     for (Staff s: staff) {
3         s.adjustSalary(byPercent);
4     }
5 }
```

However, at runtime `s` can receive an object of any subclass of `Staff`. That means the `adjustSalary(int)` operation of the actual subclass object will be called. If the subclass does not override that operation, the operation defined in the superclass (in this case, `Staff` class) will be called.

6. Objects interact by sending messages.
8. Packaging is one aspect of encapsulation i.e., An object packages data and related behavior together into one self-contained unit.
10. A class contains instructions for creating a specific kind of objects.
12. **A class-level member can be accessed using the class name, but also using an instance of the class.**

20. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass. (super)
21. In Java, method **overloading** can happen within the same class or among classes in an inheritance hierarchy.
24. A class that has an abstract method becomes an abstract class.
25. In Java, even a class that does not have any abstract methods can be declared as an abstract class.
27. A class implementing an interface results in an **is-a relationship**, just like in class inheritance.
28. Interfaces cannot be instantiated - they can only be implemented by classes.
33. Every instance of a subclass is an instance of the superclass, but not vice-versa.
35. These three things contribute greatly to **polymorphism: substitutability, overriding, dynamic binding**.
42. **Errors and runtime exceptions are collectively known as unchecked exceptions.**
45. In Java, checked exceptions are subject to the Catch or Specify Requirement.

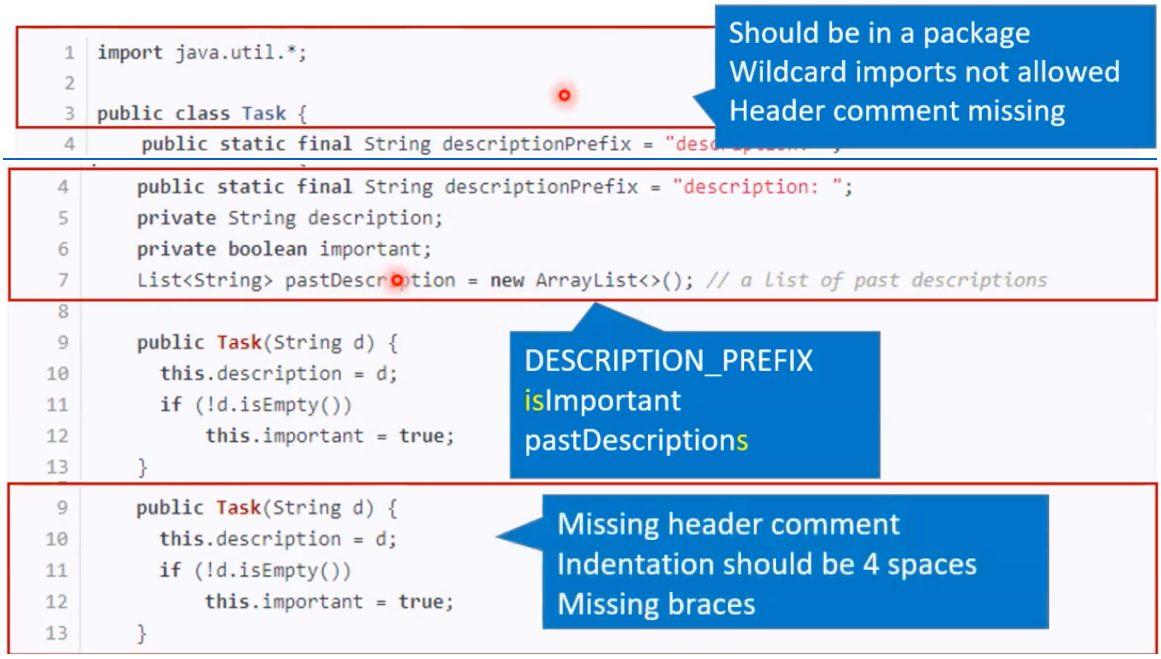
Week 2: OOP: SDLC, Revision Control Software(RCS)

https://nus-cs2103-ay2122s1.github.io/website/schedule/week2/topics.html
Software Development Life Cycle (SDLC)
Integrated Development Environments (IDEs)
Software Under Test (SUT)

11. The iterative model can be either breadth-first or depth-first.
17. When you modify a system, the modification may result in some unintended and undesirable effects on the system. Such an effect is called a regression.
18. Testing cannot fix bugs. It can detect bugs only.

Week 3: Revision Control Software(RCS), Code Quality, Developer Testing, Unit Testing

https://nus-cs2103-ay2122s1.github.io/website/schedule/week3/topics.html



this.不是必要时不要用

```
8
9     public Task(String d) {
10      this.description = d;
11      if (!d.isEmpty())
12          this.important = true;
13    }
14
15    public String getAsXML() { return "<task>"+description+"</task>"; }
8
9     public Task(String d) {
10      this.description = d;
11      if (!d.isEmpty())
12          this.important = true;
13    }
14
15    public String getAsXML() { return "<t
16
17    /**
18     * Print the description as a string.
19     */
20    public void printingDescription(){ System.out.println(this); }
21
22    @Override
23    public String toString() { return descriptionPrefix + description; }
24  }
```

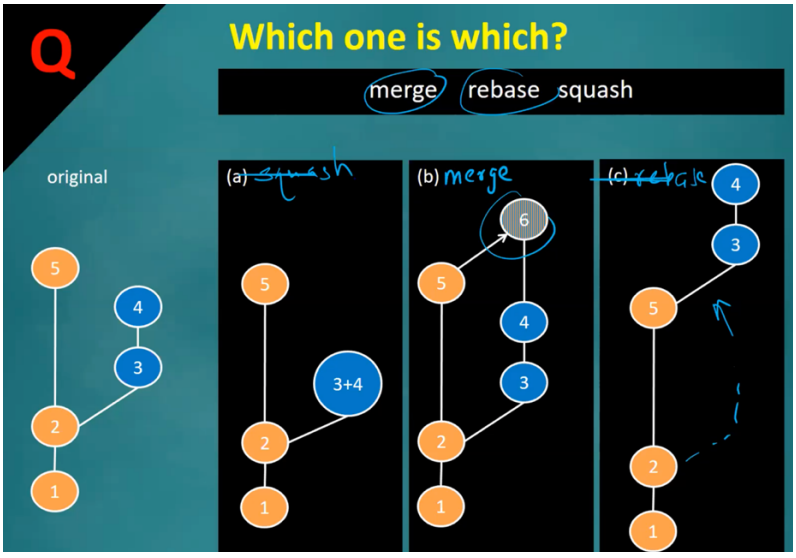
Header comment missing
getAsXml
Should use Egyptian braces
Add spaces around '+'

Prints ...
printDescription
Space before '{'
Missing header comment
Egyptian braces

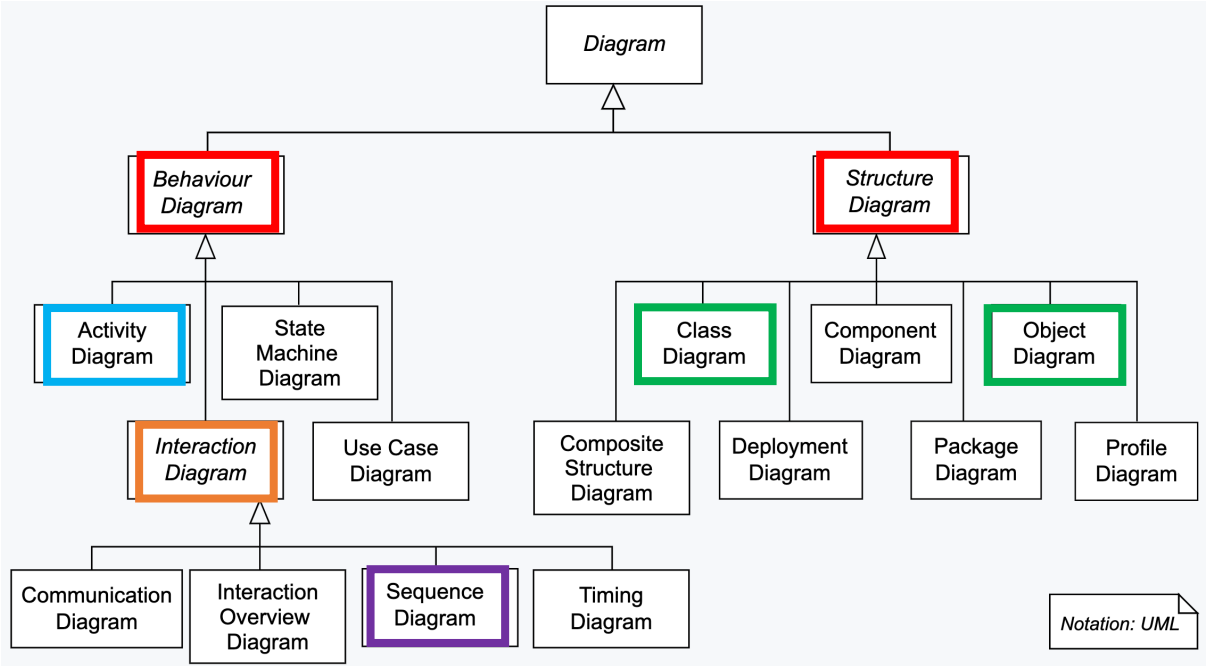
21. Developer testing is the testing done by the developers themselves.
24. **A test driver is the code** that 'drives' the SUT for the purpose of testing.
29. That is not the purpose of a stub: A stub is an alternative implementation of an SUT; by having two alternative implementations, we reduce the risk of bugs.

Week 4: Models, Class/Object Diagrams: Basics, Naming, Static Analysis, Code reviews, Revision Control Software(RCS), Automating the Build Process

https://nus-cs2103-ay2122s1.github.io/website/schedule/week4/topics.html



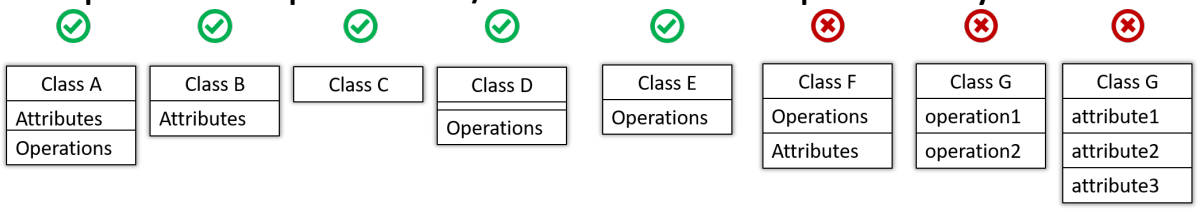
注意行为 Behaviour, 结构 Structure, 交互 Interaction



UML Object Diagrams are used to model **object structures** and UML Class Diagrams are used to model **class structures of an OO solution**.

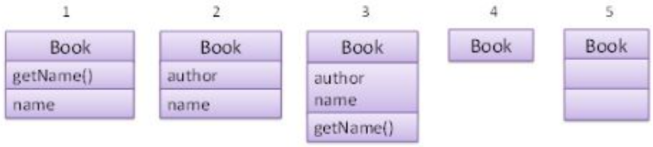
Class Diagram

The "Operations" compartment and/or the "Attributes" compartment may be omitted.



+	public
-	private
#	protected
~	package private

Which of these follow the correct UML notation?

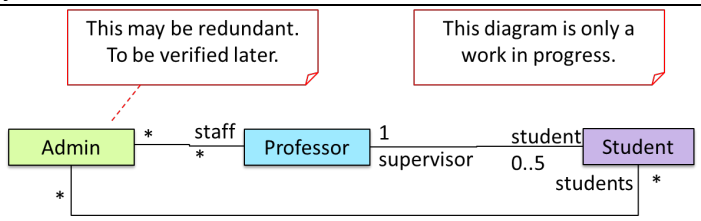
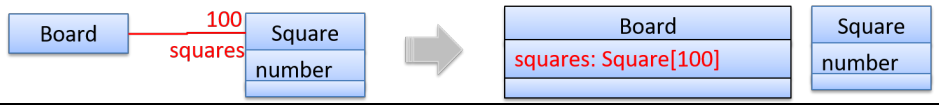
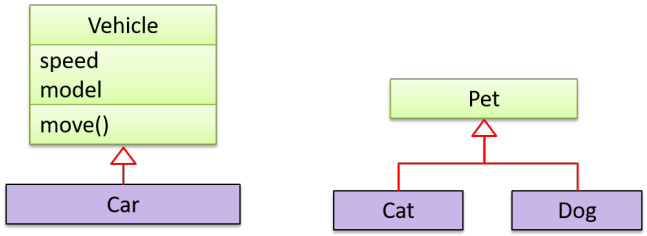
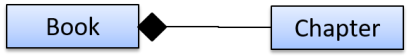
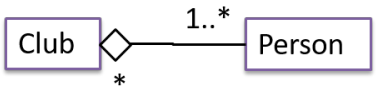


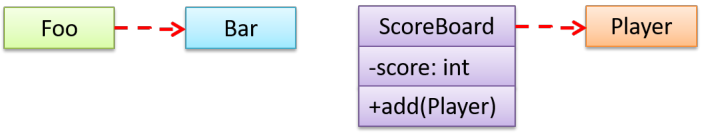
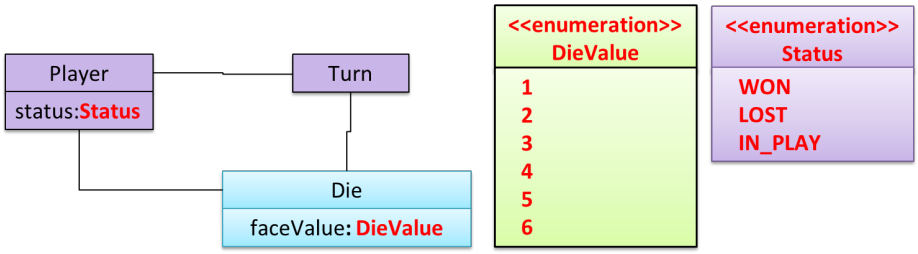
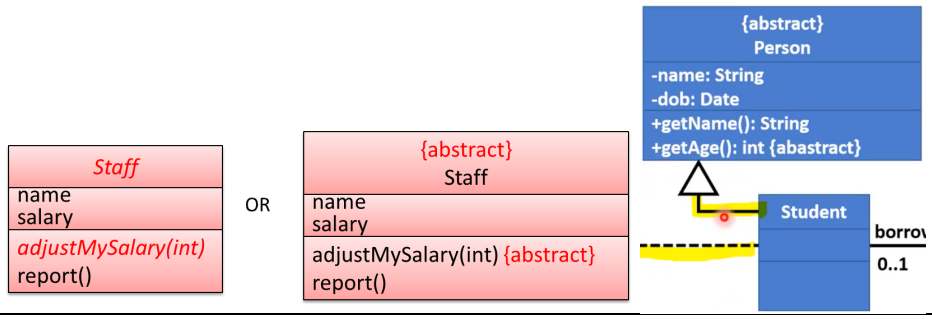
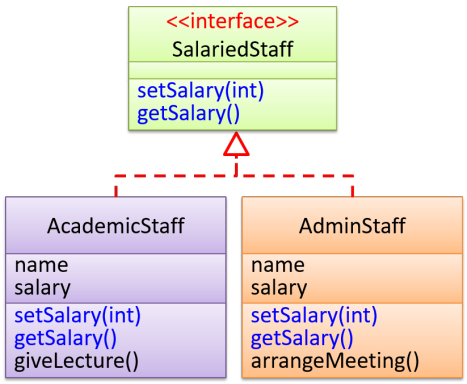
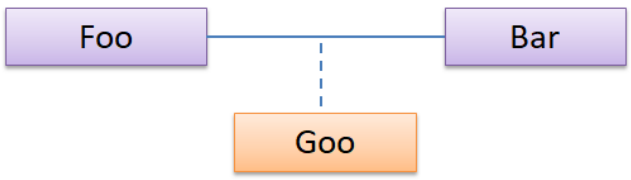
- [ ] 1
- [ ] 2
- [x] 3
- [x] 4
- [x] 5

1. Incorrect: *Attributes* compartment should be above the *Methods* compartment.
2. Incorrect: All attributes should be inside the same compartment.
3. Correct
4. Correct: Both *Attributes* and *Methods* compartments can be omitted.
5. Correct: The *Attributes* and *Methods* compartments can be empty.

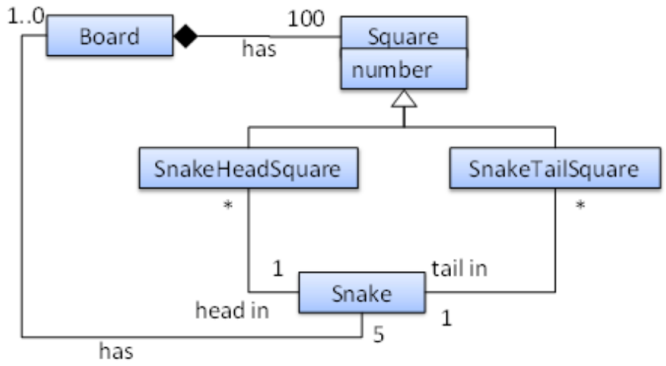
Associations	连线
Labels	按照箭头方向阅读
Roles	代表定义或实例化某个类时的名称
Multiplicity	0..1(optional associations) 1(compulsory associations) * n..m
Navigability	箭头的方向代表可以依赖方向，例如 A -> B 代表 A 中有 B

Bidirectional	**UML 图中的双向箭头不代表就是 Bidirectional**
	class Foo {
	Bar bar;
	}
	class Bar {
	Foo foo;
	}

Notes	
Associations as attributes	name: type [multiplicity] = default value **attribute 和 line 只能二选一** 
Inheritance	
Composition (Consist)	A Book **consists** of Chapter objects. **When the Book object is destroyed, its Chapter objects are destroyed too**. 
Aggregation	不建议使用 
Dependencies	**Dependencies vs associations**

	仅当依赖关系尚未以另一种方式（例如，作为关联或继承）被图表捕获时，才使用依赖关系箭头来指示依赖关系，例如 **Foo 类访问 Bar 中的常量，**但从 Foo 到 Bar 没有关联/继承。 如下图的 add(Player) 
Enumerations	
Abstract / static Classes	*斜体*或{}，可用于方法或类 
Interfaces	**注意与 Inheritance 的区别,虚线, Implement 的方法也要照抄** **注意 interface 中的 method 默认是 public +** 
Association classes	Goo 是 Association class, Foo 和 Bar 是普通 class 

Which of these statements match the class diagram?



- [ ] a. A `Snake` must belong to at least one `Board` .
- [x] b. A `SnakeHeadSquare` can contain only one `Snake` head.
- [x] c. A `Snake` head can be in a `Square`
- [x] d. A `Snake` head can be in more than one `SnakeHeadSquare` .
- [x] e. The `Board` has exactly 5 `Snake` s.

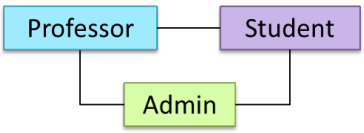
~~(a)~~(b)(c)(d)(e)

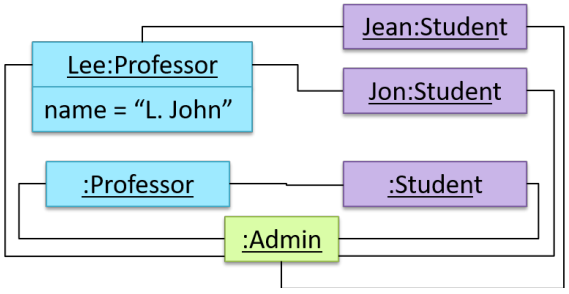
Explanation:

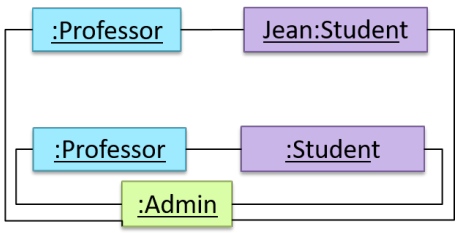
(a) does not match because a `Snake` may or may not belong to a `Board` (multiplicity is `0..1`)
(b) matches the diagram because the multiplicity given is `1`
(c) matches the diagram because `SnakeHeadSquare` is a `Square` (due to inheritance)
(d) matches the diagram because the multiplicity given is `*`
(e) matches the diagram because the multiplicity given is `5`

Object Diagram: 注意下划线

Multiple object diagrams can correspond to a single class diagram.



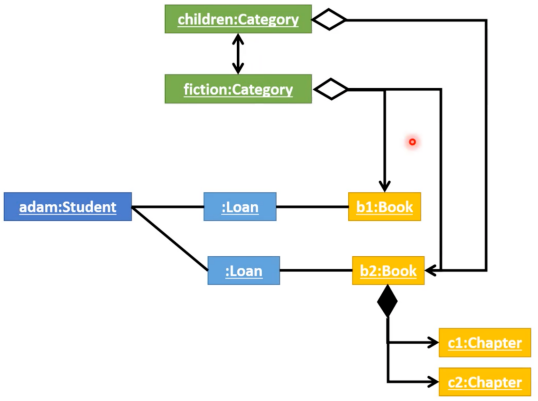




Static analysis: 执行代码前

Dynamic analysis: 执行代码后

Object Diagram represents the object structure at a specific point of time. 一个 Class Diagram 可以有多个 Object Diagram。



The student adam has borrowed two books b1 and b2.
b1 is a fiction. b2 is a children fiction, and has two chapters.
The two categories are related.

11. Multiple models of the same entity may be needed to capture it fully. In addition to a class diagram (or even multiple class diagrams), a number of other diagrams may be needed to capture various interesting aspects of the software.
15. In the following class diagram, the UML term used for the '+' and '-' signs to is **visibility** ~~accessibility~~.
36. **Composition is a whole-part relationship, aggregation is a container-containee relationship.**
43. Class names should be nouns and method names should be verbs.
46. As per the textbook, PR reviews is **one of the three ways (Pull Request reviews, In pair programming, Formal inspections)** of reviewing code.
47. As per the article (i.e., 10 tips for reviewing code you don't like) given in the resources section of the textbook, it is recommended to phrase your comments as questions.
49. Linters are a subset of static analyzers.

Week 5: Requirements, Code Quality, Refactoring, Assertions, Revision Control Software(RCS)

https://nus-cs2103-ay2122s1.github.io/website/schedule/week5/topics.html

Given below are some requirements of TEAMMATES (an online peer evaluation system for education). Which one of these are non-functional requirements?

- [x] a. The response to any use action should become visible within 5 seconds.
- [ ] b. The application admin should be able to view a log of user activities.
- [x] c. The source code should be open source.
- [x] d. A course should be able to have up to 2000 students.
- [ ] e. As a student user, I can view details of my team members so that I can know who they are.
- [x] f. The user interface should be intuitive enough for users who are not IT-savvy.
- [x] g. The product is offered as a free online service.

(a)(c)(d)(f)(g)

Explanation: (b) are (e) are functions available for a specific user types. Therefore, they are functional requirements. (a), (c), (d), (f) and (g) are either constraints on functionality or constraints on how the project is done, both of which are considered non-functional requirements.

Choose the correct statements

- [ ] a. User stories are short and written in a formal notation.
- [ ] b. User stories is another name for use cases.
- [ ] c. User stories describes past experiences users had with similar systems. These are helpful in developing the new system.
- [x] d. User stories are not detailed enough to tell us exact details of the product.

d

Explanation: User stories are short and written in natural language, NOT in a formal language. They are used for estimation and scheduling purposes but do not contain enough details to form a complete system specification.

Avoid long methods
Avoid deep nesting
Avoid complicated expressions
Avoid magic numbers
Make the code obvious
Structure code logically
Do not 'Trip Up' reader
Practice KISSing (keep it simple, stupid)
Avoid premature optimizations

Single Level of Abstraction Principle (SLAP) hard
Make the happy path prominent
Use the default branch
Don't recycle variables or parameters
Avoid empty catch blocks
Delete dead code
Minimize scope of variables
Minimize code duplication: DRY (Don't Repeat Yourself) Principle

'Extract method' and 'Inline method' refactorings

- (a) a. are opposites of each other.
- ( ) b. sound like opposites but are not.

a

Which statements are correct?

- [x] a. Use assertions to indicate that the programmer messed up; use exceptions to indicate that the user or the environment messed up.
- [ ] b. Use exceptions to indicate that the programmer messed up; use assertions to indicate that the user or the environment messed up.

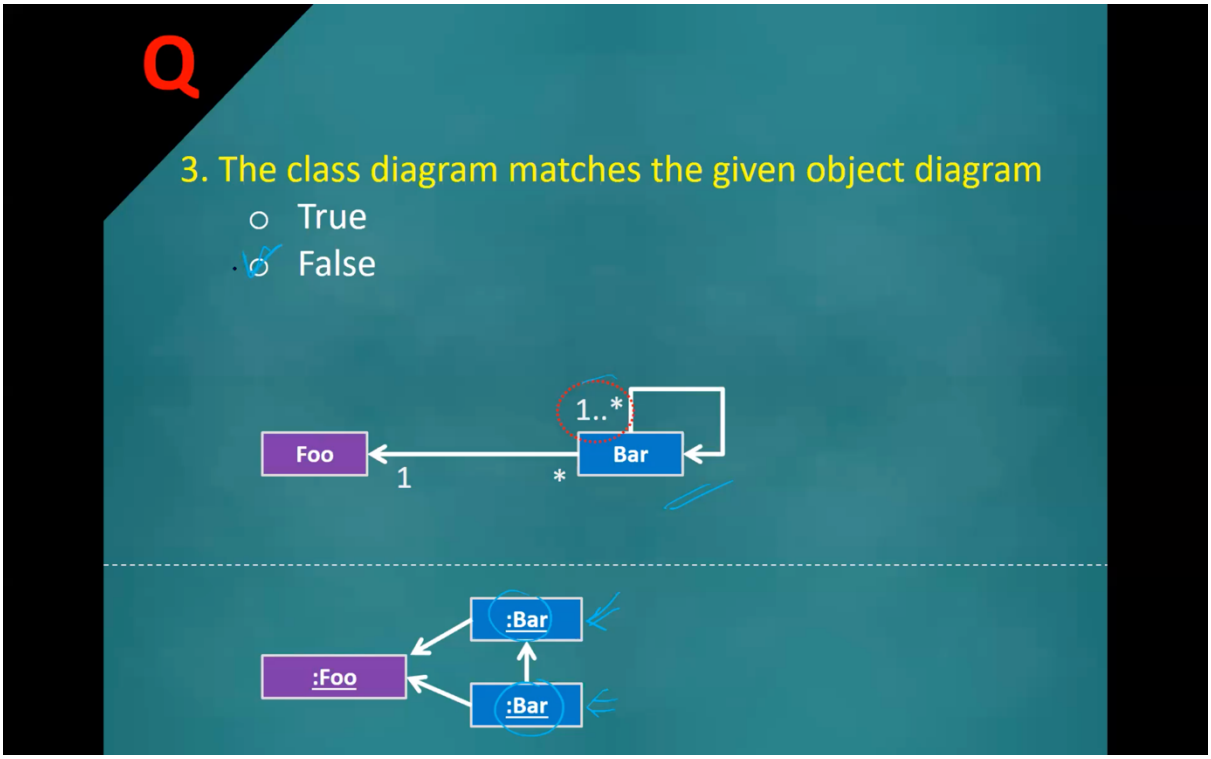
(a)

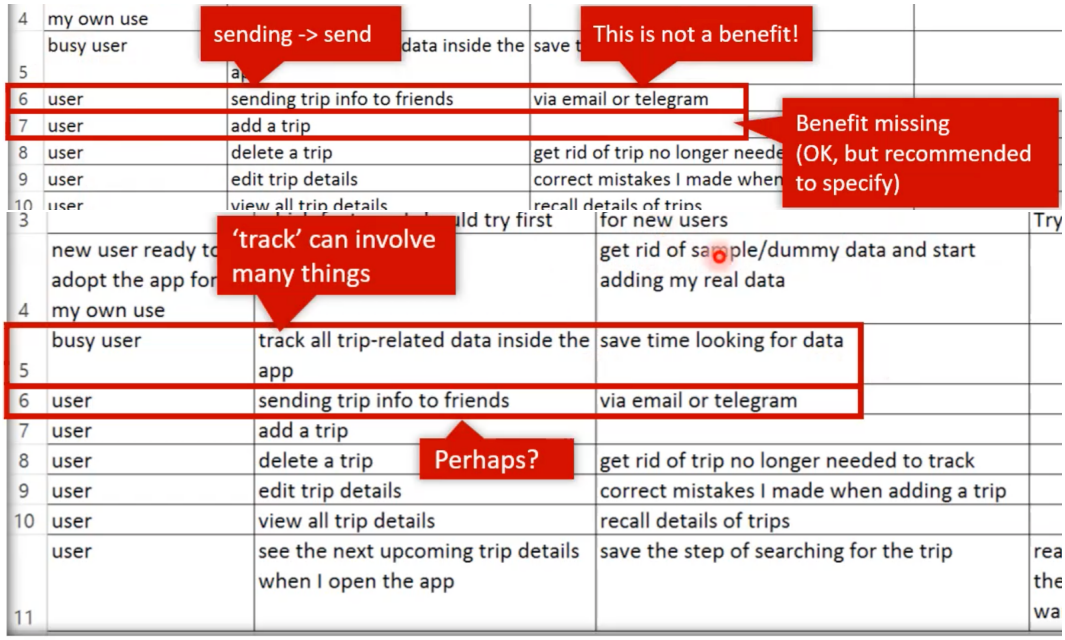
A Calculator program crashes with an 'assertion failure' message when you try to find the square root of a negative number.

- ( ) a. This is a correct use of assertions.
- ( ) b. The application should have terminated with an exception instead.
- (c) c. The program has a bug.
- ( ) d. All statements above are incorrect.

(c)

Explanation: An assertion failure indicates a bug in the code. (b) is not acceptable because of the word "terminated". The application should not fail at all for this input. But it could have used an exception to handle the situation internally.



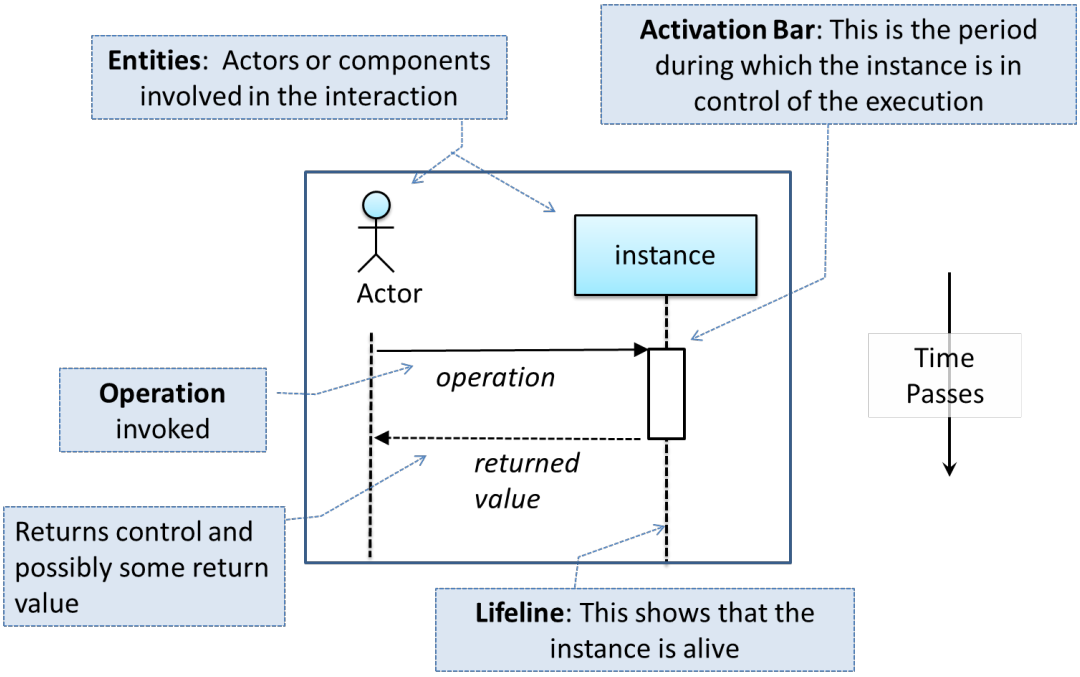


Any of them **too big** for the tP planning?
i.e., cannot be implemented by one person within 1-2 days.

14. **Requirements should be implementation-free**.
24. High-level user stories are sometimes called **themes or epics**.
25. You can add other useful info (Conditions, Priority, Size, Urgency) to a user story.
27. **User stories can capture non-functional requirements** too because even NFRs must benefit some stakeholder.
52. Refactoring can cause regressions.

Week 6: Sequence Diagrams, Architecture Diagrams, IDEs, Logging

https://nus-cs2103-ay2122s1.github.io/website/schedule/week6/topics.html

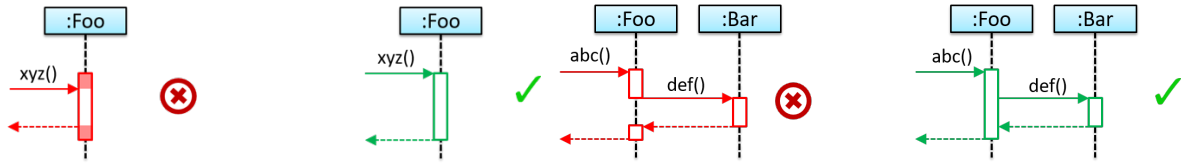


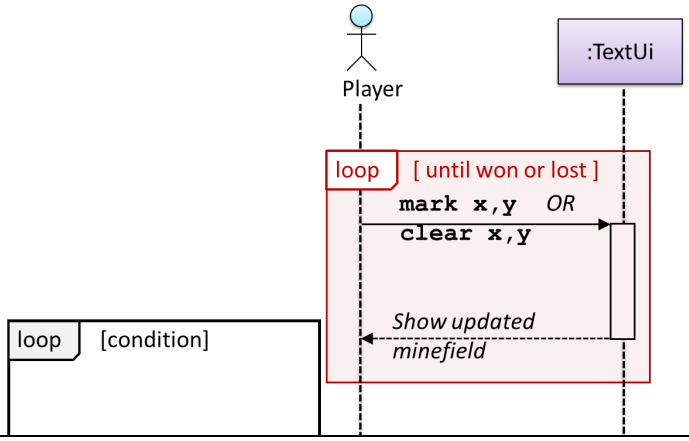
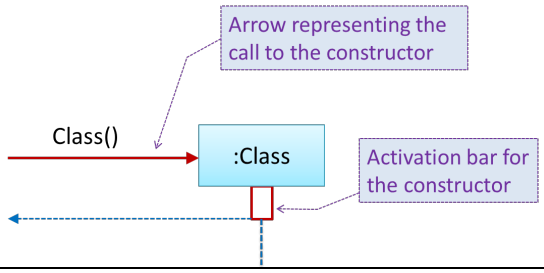
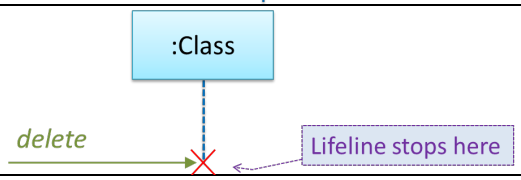
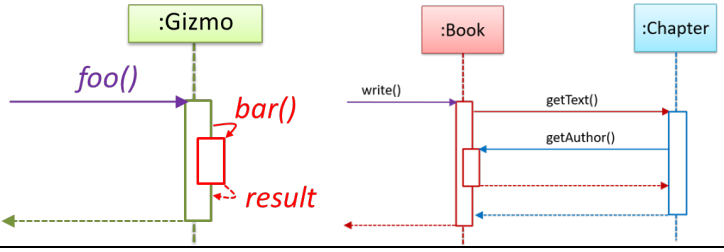
方法调用是实线箭头，方法返回是虚线箭头

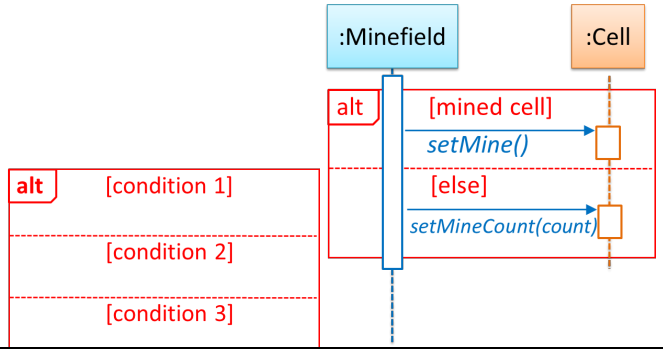
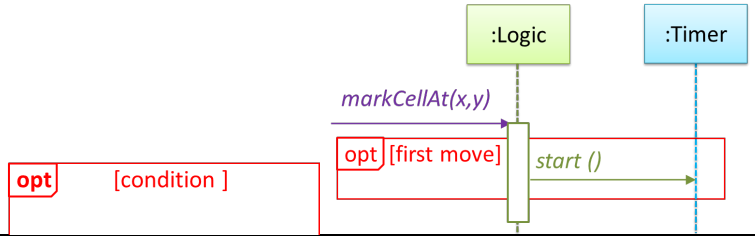
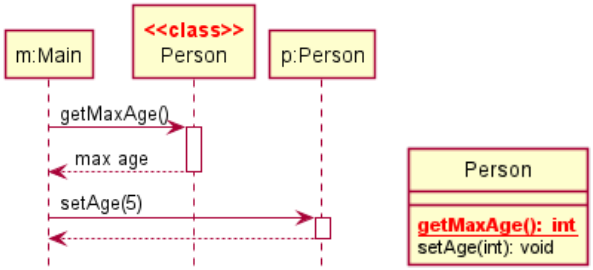
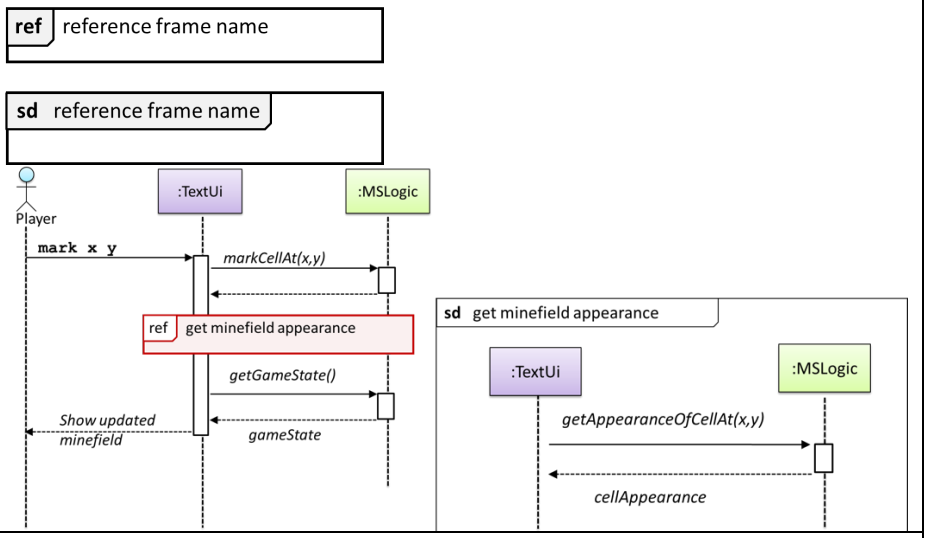
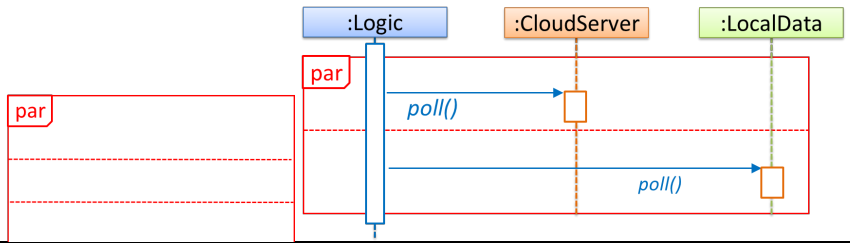
instance 有冒号无下划线，例:Abc

activation bars 和 return arrows 可省略

注意下面错误实例



Loops	**右下角不是直角，没有 for while 之分，条件中括号[]不能漏了** 
Object Creation	**箭头指向 Entity，调用 Constructor，Activation Bar 紧贴 Entity** 
Object Deletion	
Self-Invocation	
Alternative Paths	**右下角不是直角，类似 if...else...，条件中括号[]不能漏了**

	
Optional Paths	**右下角不是直角，类似 if...，条件中括号[]不能漏了** 
Static Methods	
Reference Frames	**UML 使用 ref 允许省略交互的一部分并显示为单独的序列图** 
Parallel Paths	**注意并行路径的运行不分先后** 

What are the advantages of using use cases (the textual form) for requirements modelling?

- [x] a. They can be fairly detailed but still natural enough for users to understand and give feedback.
- [x] b. The UI-independent nature of use case specification allows the system designers more freedom to decide how a functionality is provided to a user.
- [x] c. Extensions encourage us to consider all situations a software product might face during its operations.
- [x] d. They encourage us to identify and optimize the typical scenario of usage over exceptional usage scenarios.

(a) (b) (c) (d)

Which of these are correct?

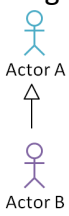
- [x] a. Use cases are not very suitable for capturing non-functional requirements.
- [x] b. Use case diagrams are less detailed than textual use cases.
- [ ] c. Use cases are better than user stories.
- [x] d. Use cases can be expressed at different levels of abstraction.

(a)(b)(d)

Explanation: It is not correct to say one format is better than the other. It depends on the context.

Top-down	Big and novel systems.
Bottom-up	A variation of an existing system or re-purposing existing components to build a new system.
Mix	Design the top levels using the top-down approach but switch to a bottom-up approach when designing the bottom levels.
Agile 敏捷	

Work Breakdown Structure (WBS): depicts information about tasks and their details in terms of subtasks.
Centralized RCS (CRCS for short): uses a central remote repo that is shared by the team.
Distributed RCS (DRCS for short, also known as Decentralized RCS): allows multiple remote repos

31. Architecture diagrams are free-form diagrams.
32. The software architecture shows the overall organization of the system. It can be viewed as a very high-level design of the software.
34. Use cases should not be used as the sole means to specify requirements because they are not good for capturing requirements that do not involve a user interaction.
35. 

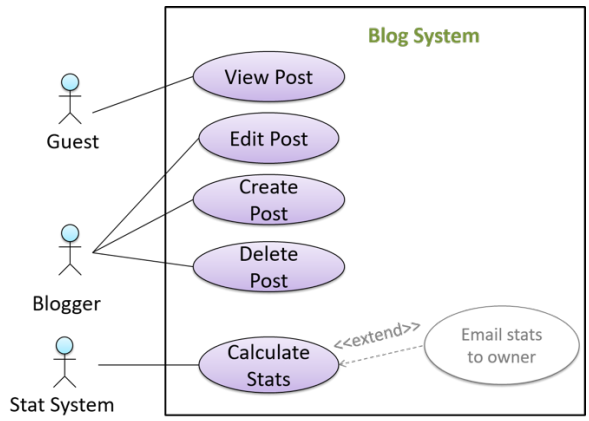
This means all use cases available to Actor A are also available to Actor B.

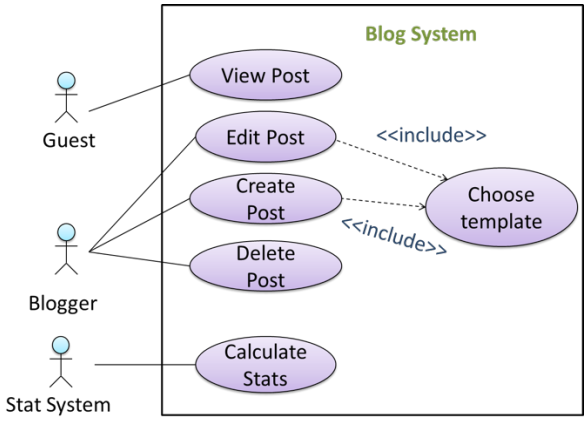
Week 7: Use Cases, High-Level View, Abstraction, Coupling, Cohesion, Integration Approaches, Project Mgt
https://nus-cs2103-ay2122s1.github.io/website/schedule/week7/topics.html

Use Case: capture the functional requirements of a system.
UI details are usually omitted.

- Software System
- Use case
- Actor
- Preconditions
- Guarantees: what the use case promises to give us at the end of its operation.
- Main Success Scenario (MSS)

- **Inclusions**: A use case can include another use case. Underlined text.
- Extensions: describe exceptional/alternative flow of events.

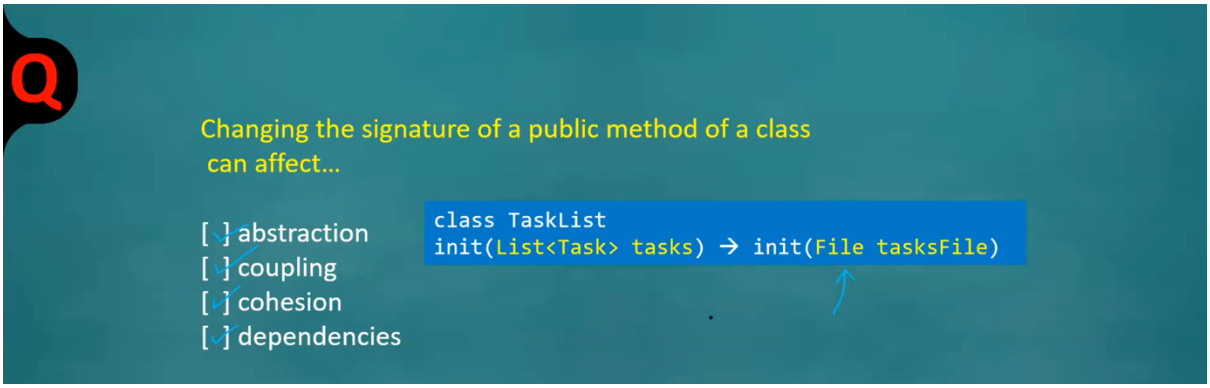




箭头方向: Email... is an extension to Calculate... **箭头方向: Edit... include Choose...**

15, 17. An actor (in a use case) is a role played by a user. An actor can be a human or another system. Actors are not part of the system; they reside outside the system.
42. The agile design approach does not produce a fully-documented set of design models before coding begins.
51. X is coupled to Y if a change to Y can **potentially (but not necessarily always)** require a change in X.
61. It is not recommended to inflate task estimates to create hidden buffers.
68. Git is a DRCS. Subversion (SVN) is a CRCS.

Week 8: Integration Testing, System Testing, Automated testing of GUIs, Acceptance testing (aka User Acceptance Testing (UAT)), Alpha/Beta Testing, Exploratory vs Scripted Testing, Dependency Injection (stub), TDD
https://nus-cs2103-ay2122s1.github.io/website/schedule/week8/topics.html
Integration tests are additional test cases that focus on the interactions between the parts, **not using stub**.
Test-Driven Development(TDD) advocates writing the tests before writing the SUT, while evolving functionality and tests in small increments.



Here are some examples of different coverage criteria:

- **Function/method coverage** : based on functions executed e.g., testing executed 90 out of 100 functions.
- **Statement coverage** : based on the number of lines of code executed e.g., testing executed 23k out of 25k LOC.
- **Decision/branch coverage** : based on the decision points exercised e.g., an `if` statement evaluated to both `true` and `false` with separate test cases during testing is considered 'covered'.
- **Condition coverage** : based on the boolean sub-expressions, each evaluated to both true and false with different test cases. Condition coverage is not the same as the decision coverage.

`if(x > 2 && x < 44)` is considered one decision point but two conditions.

For 100% branch or decision coverage, two test cases are required:

- `(x > 2 && x < 44) == true` : [e.g. `x == 4`]
- `(x > 2 && x < 44) == false` : [e.g. `x == 100`]

For 100% condition coverage, three test cases are required:

- `(x > 2) == true` , `(x < 44) == true` : [e.g. `x == 4`] [see note 1]
- `(x < 44) == false` : [e.g. `x == 100`]
- `(x > 2) == false` : [e.g. `x == 0`]

Note 1: A case where both conditions are `true` is needed because most execution environments use a *short circuiting* behavior for compound boolean expressions e.g., given an expression `c1 && c2` , `c2` will not be evaluated if `c1` is `false` (as the final result is going to be `false` anyway).

- **Path coverage** measures coverage in terms of possible paths through a given part of the code executed. 100% path coverage means all possible paths have been executed. A commonly used notation for path analysis is called the *Control Flow Graph (CFG)*.
- **Entry/exit coverage** measures coverage in terms of possible *calls to* and *exits* from the operations in the SUT.

Part 2:

2. Integration testing is **not simply** a case of repeating the unit test cases **using the actual dependencies** (instead of the stubs used in unit testing). Instead, integration tests are **additional test cases** that focus on the interactions between the parts.

3. The difference between pure unit/integration testing and hybrid unit/integration testing is that the pure approach has one extra testing step, **hybrid remove stub**.

6. Usability testing and portability testing is part of system testing. So is performance testing.

9. Acceptance testing (aka User Acceptance Testing (UAT) also be done by professional testers representing the users.

10. In some projects, one document serves as both the requirements specification and the system specification.

12. **Acceptance testing focus more on positive test cases.**

25. Coverage analysis is usually a dynamic analysis.

26. 100% path coverage can require more test cases than 100% statement coverage of the same code. A test suite can execute all statements without necessarily executing all possible paths of the code.

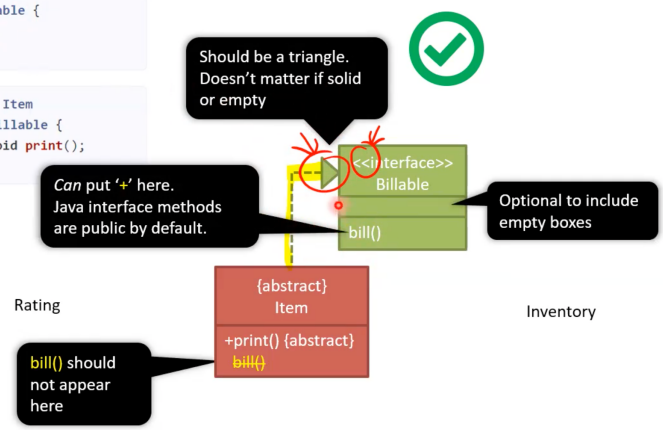
27. 100% condition coverage can require more test cases than 100% branch coverage of the same code. A branch may require more than one condition to be true.
28. Test coverage is a metric used to measure the extent to which testing exercises the SUT.

https://nus-cs2103-ay2122s1.github.io/website/schedule/week8/tutorial.html

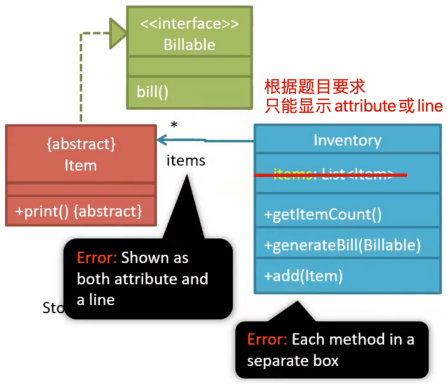
Class Diagram

```java
public interface Billable {
    void bill();
}
```

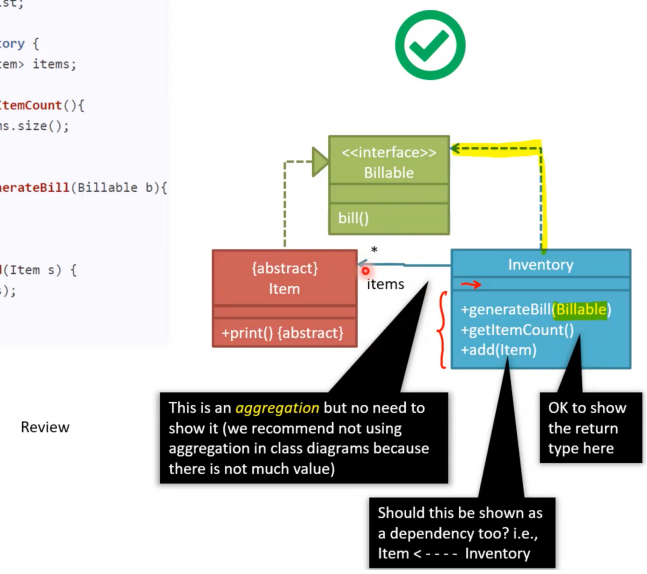
```java
public abstract class Item
        implements Billable {
    public abstract void print();
}
```




```java
import java.util.List;

public class Inventory {
    private List<Item> items;

    public int getItemCount(){
        return items.size();
    }

    public void generateBill(Billable b){
        // ...
    }

    public void add(Item s) {
        items.add(s);
    }
}
```




```java
import java.util.List;

public class Inventory {
    private List<Item> items;

    public int getItemCount(){
        return items.size();
    }

    public void generateBill(Billable b){
        // ...
    }

    public void add(Item s) {
        items.add(s);
    }
}
```

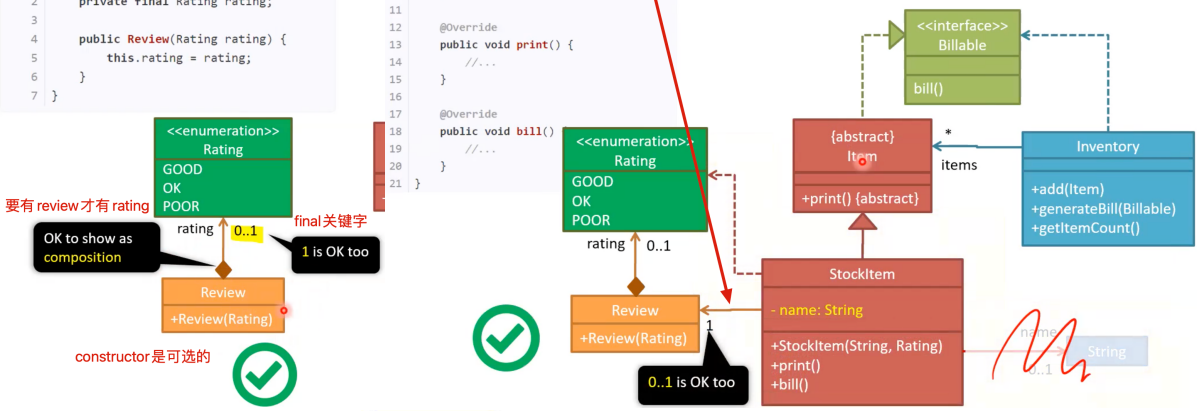


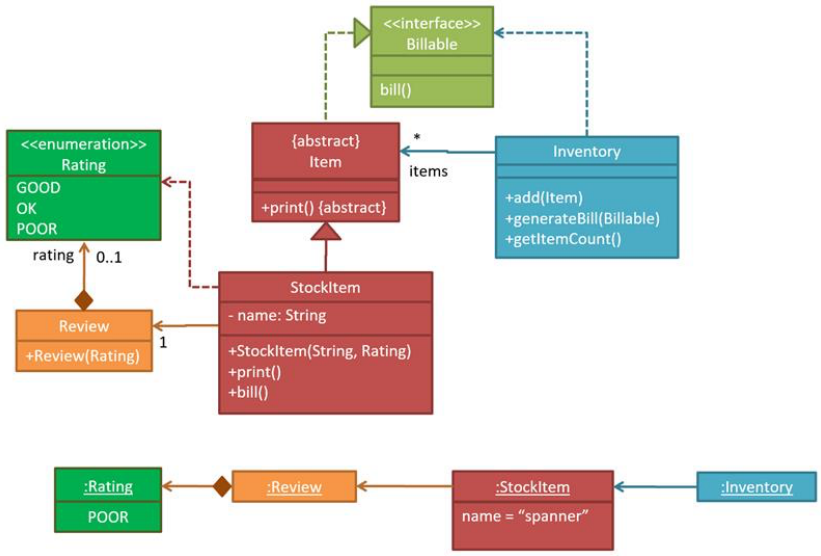
```java
public enum Rating {
    GOOD, OK, POOR
}
```

```java
public class Review {
    private final Rating rating;

    public Review(Rating rating) {
        this.rating = rating;
    }
}
```

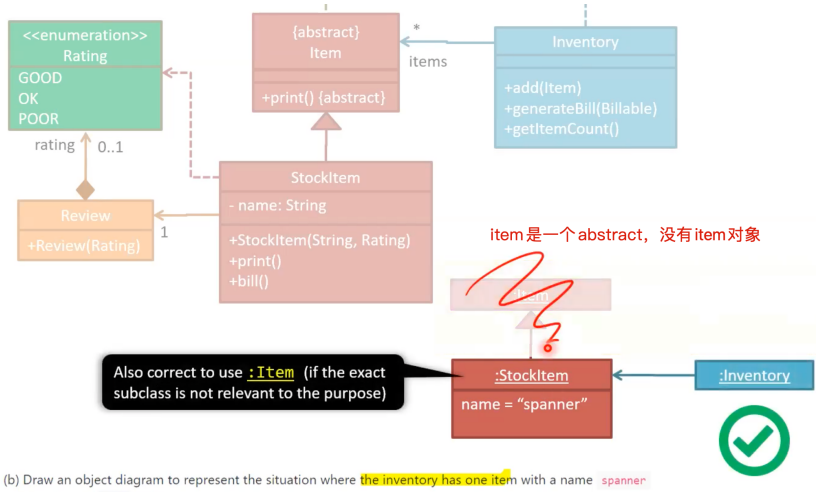
```java
public StockItem(
    String name, Rating rating){

    this.name = name;
    this.review = new Review(rating);
}

@Override
public void print() {
    //...
}

@Override
public void bill()
    //...
}
```

总是会new一个review对象



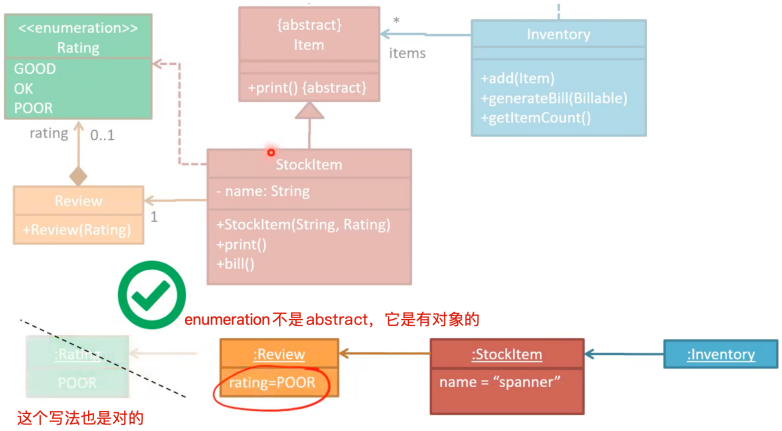


Object Diagram



(b) Draw an object diagram to represent the situation where the inventory has one item with a name spanner and a review of POOR rating

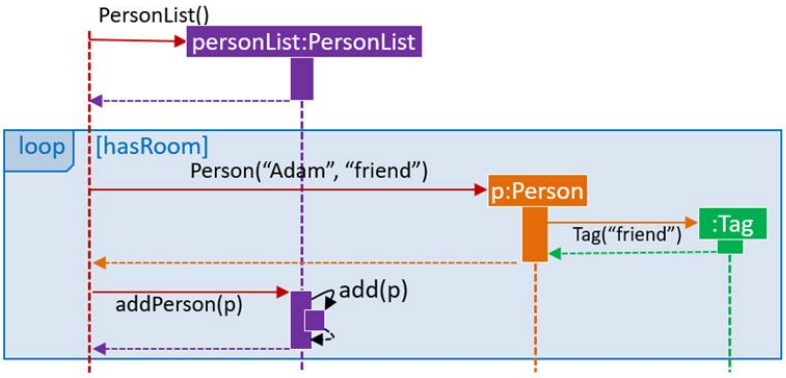
i.e. new Inventory().add(new StockItem("spanner", Rating.POOR))



(b) Draw an object diagram to represent the situation where the inventory has one item with a name `spanner` and a review of `POOR` rating

i.e. `new Inventory().add(new StockItem("spanner", Rating.POOR))`

Sequence Diagram



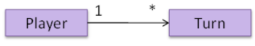
Week 9: OO Domain Models, Activity Diagrams, Architecture Diagrams, Design Principles, SDLC Agile, Developer Guide

https://nus-cs2103-ay2122s1.github.io/website/schedule/week9/topics.html

Class diagrams that are used to **model the problem domain (do not contain solution-specific classes)** are called conceptual class diagrams or OO domain models (OODMs). OODM notation is similar to class diagram notation but **omit methods (operations) and navigability**.

This diagram is,

- [x] a. A class diagram.
- [ ] b. An object diagram.
- [ ] c. An OO domain model, also known as a conceptual class diagram.
- [ ] d. Can be either a class diagram or an OO domain model.



(a)

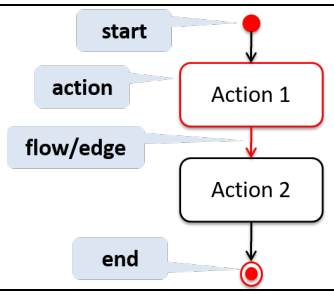
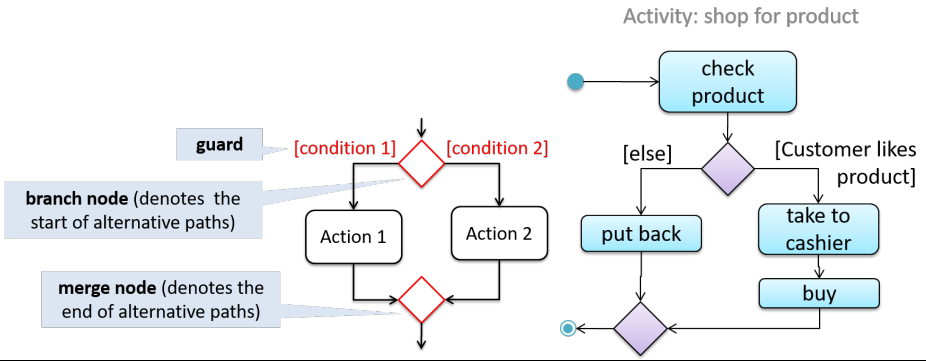
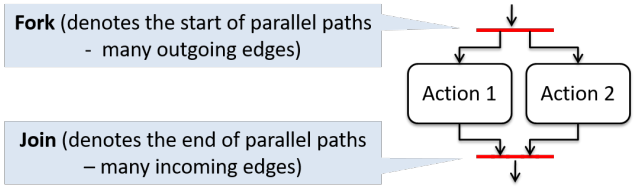
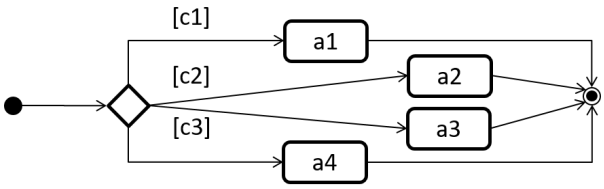
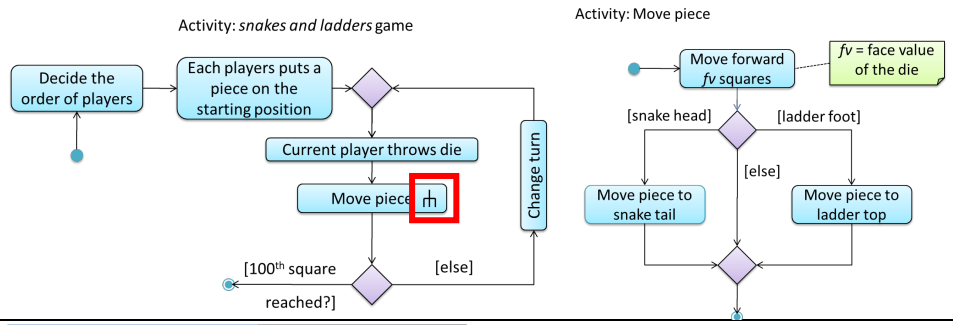
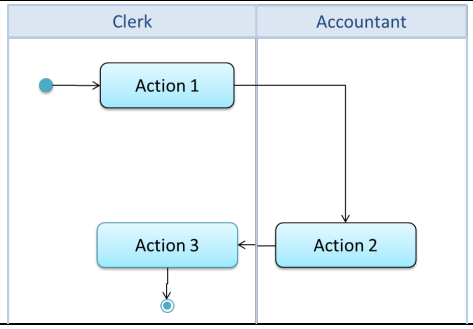
Explanation: The diagram shows navigability which is not shown in an OO domain model. Hence, it has to be a class diagram.

What is the main difference between a class diagram and an OO domain model?

- (●) a. One is about the problem domain while the other is about the solution domain.
- ( ) b. One has more classes than the other.
- ( ) c. One shows more details than the other.
- ( ) d. One is a UML diagram, while the other is not a UML diagram.

(a)

Explanation: Both are UML diagrams, and use the class diagram notation. While it is true that often a class diagram may have more classes and more details, the main difference is that the OO domain model describes the problem domain while the class diagram describes the solution.

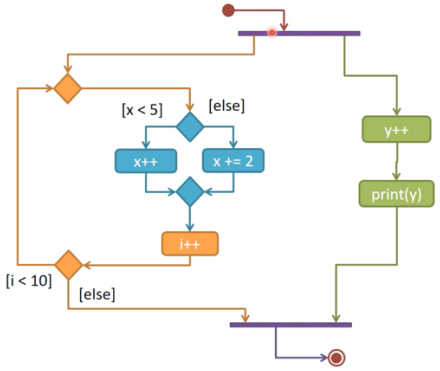
Linear Paths	
Alternate Paths	**条件中括号[]不能漏了** 
Parallel Paths	**必须要把两个 Action 都 Join 了后才能继续下一步** 
省略	**省略合并节点, 多个箭头可以从同一个分支节点开始, 省略[Else]条件** 
Rakes	
Swim Lanes	

do-while

Draw an activity diagram to illustrate that the following two code snippets are executed in parallel.

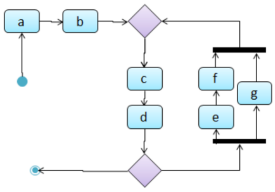
```
do {
    x < 5 ? x++ : x += 2;
    i++;
} while (i < 10);
```

```
y++;
print(y);
```



Which of these sequences of actions is not allowed by the given activity diagram?

- [ ] i. start a b c d end
- [ ] ii. start a b c d e f g c d end
- [ ] iii. start a b c d e g f c d end
- [x] iv. start a b c d g c d end



(iv)

Explanation: `-e-f-` and `-g-` are parallel paths. Both paths should complete before the execution reaches `c` again.

Separation of concerns principle (SoC)	Higher cohesion and lower coupling.
Single responsibility principle (SRP)	A class should have one, and only one, reason to change.
Liskov substitution principle (LSP)	A subclass should not be more restrictive than the behavior specified by the superclass. 只要父类能出现的地方，子类就可以出现，**并且替换为子类也不会产生任何错误或异常**，使用者可能根本就不需要知道是父类还是子类。但反之，未要求。
Open-closed principle (OCP)	A module should be open for extension but closed for modification.
Law of Demeter (LoD)	对于 object O 中的 method m, m 可以调用 1. The object O itself 2. Objects passed as parameters of m 3. Objects created/instantiated in m (directly or indirectly) 4. Objects from the direct association of O

SOLID:
Single Responsibility Principle (SRP)
Open-Closed Principle (OCP)
Liskov Substitution Principle (LSP)
Interface Segregation Principle (ISP)
Dependency Inversion Principle (DIP)

SDLC Agile (**Individuals and interactions, Working software, Customer collaboration, Responding to change**): eXtreme Programming (XP) and Scrum.
Developer Guide (**tutorials, how-to guides, explanation and technical reference**): A **top-down breadth-first explanation** is easier to understand than a bottom-up one.
Architecture Diagrams **don't use double-headed arrows indiscriminately**.

Q3

What to consider when deciding if an association is a composition?

- [ ] If `Foo` objects act as containers for `Bar` objects, it is composition.
- [ ] As long as `Bar` objects get deleted when `Foo` objects are deleted, it is composition.
- [ ] Concepts represented by the two classes form a *whole-part* relationship.
- [ ] Not always possible to judge from the code alone.

delete cascading

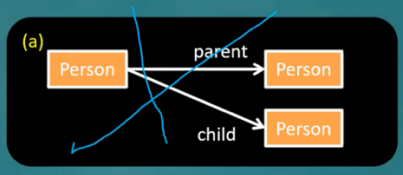
In the exam, there can be additional clues in the description/comments
e.g., `// a Foo is composed of Bars`

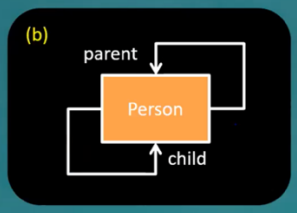
Q4

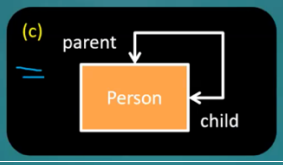
Context: A software to manage a competition. Contestants are parent-child pairs.

Which class diagram is the best match to the code?

```
class Person {
    Person parent;
    Person child;
    // …
}
```

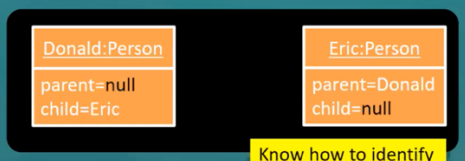




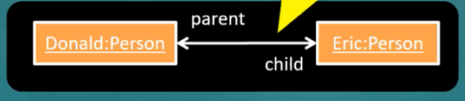


Just one parent only?

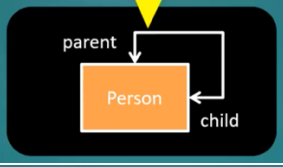
```
class Person {
    Person parent;
    Person child;
    // …
}
```



Know how to identify bidirectional associations



Both ends points to the same class, but not necessarily the same object



Context: A software to manage a competition. Contestants are parent-child pairs.

Q

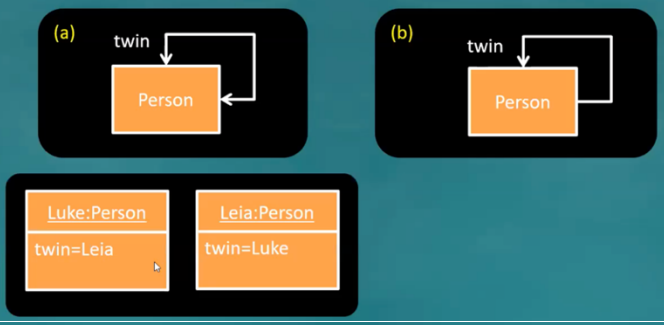
Dependency injection is most related to:

- ○ Unit testing
- ○ Integration testing
- ○ System testing

Q

Which class diagram is the best match for the code?

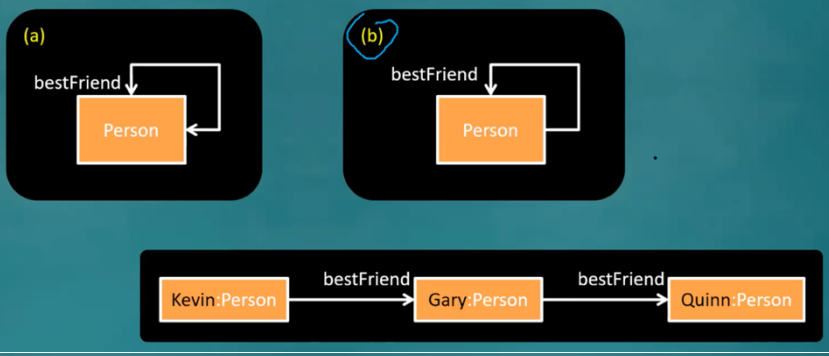
```
class Person {
    Person twin;
    // …
}
```



Q

Which class diagram is the best match for the code?

```
class Person {
    Person bestFriend;
    // …
}
```



Q

A statement in the code can be *covered* by these:

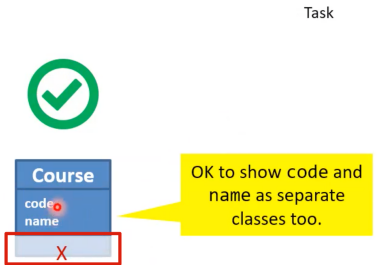
- ☑ Unit testing
- ☑ Integration testing

Can be done in hybrid mode (i.e., combined)

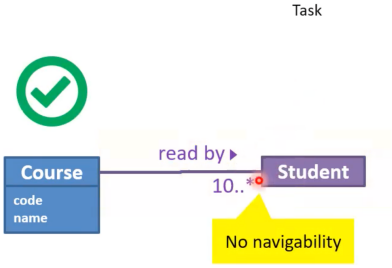
- ☑ System testing
- ☑ Smoke testing
- ☑ Acceptance testing
- ☑ Alpha/beta testing
- ☑ Regression testing
- ☑ TDD

Why not just do this one?
- Too late
- Too costly
- Too slow
- Doesn't test some things

A course has a name and a code. A course is read by 10 or more students, and taught by one or more instructors one of whom is the coordinator. A course can have a number of tasks which can be assignments or tests. Some assignments are compulsory. When a student attempts a task, a grade and an optional feedback is given.



A course has a name and a code. A course is read by 10 or more students, and taught by one or more instructors one of whom is the coordinator. A course can have a number of tasks which can be assignments or tests. Some assignments are compulsory. When a student attempts a task, a grade and an optional feedback is given.



Part 2:

2. As per SRP, a **class** should have only one responsibility.

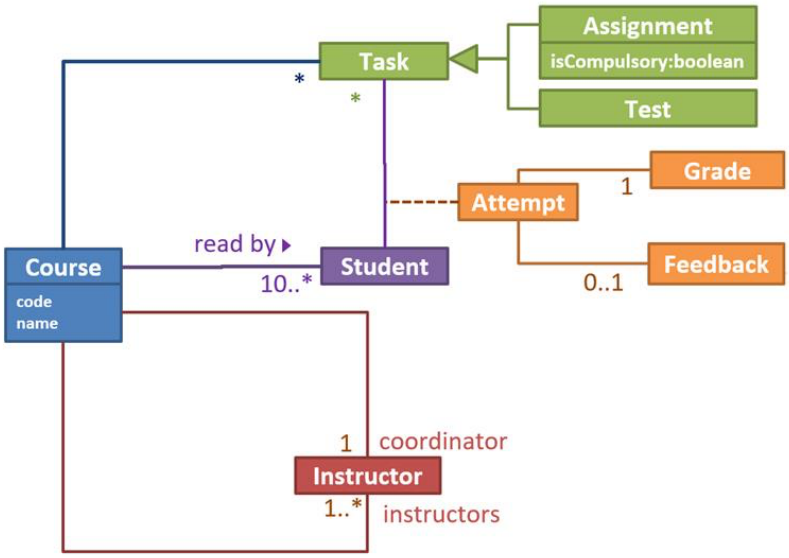
20. In the SCRUM process, iterations are called sprints.

22. A key principle of Scrum is its recognition that during a project the customers can change their minds about what they want and need (often called requirements churn), and that unpredicted challenges cannot be easily addressed in a traditional predictive or planned manner.

23. It is the Product Owner who represents the stakeholders and the business.

https://nus-cs2103-ay2122s1.github.io/website/schedule/week9/tutorial.html

A course has a name and a code. A course is read by 10 or more students, and **taught by one or more instructors one of whom is the coordinator.** A course can have a number of tasks which can be assignments or tests. Some assignments are compulsory. When a student attempts a task, a grade and an optional feedback is given.

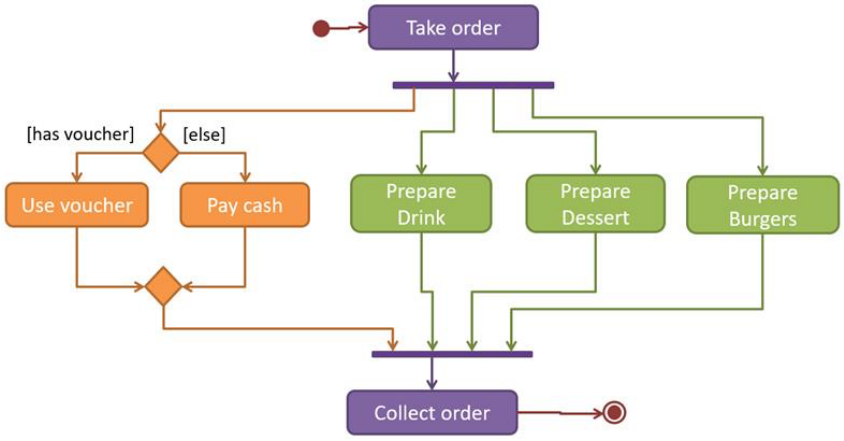


(ii) Which type of a UML diagram would you use to illustrate the following situation?

The course *CS101 Intro to CS* is taught by Prof Lee. It has two optional assignments and one test.

Draw an activity diagram to represent the following workflow a burger shop uses when processing an order by a customer.

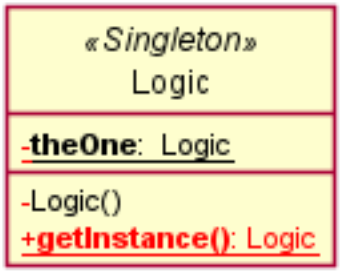
- First, a cashier takes the order.
- Then, three workers start preparing the order at the same time; one prepares the drinks, one prepares the burgers, and one prepares the desserts.
- In the meantime, the customer pays for the order. If the customer has a voucher, she pays using the voucher; otherwise she pays using cash.
- After paying, the customer collects the food after all three parts of the order are ready.



Week 10: Design Patterns, Defensive Programming, Equivalence Partitioning, Boundary Value Analysis

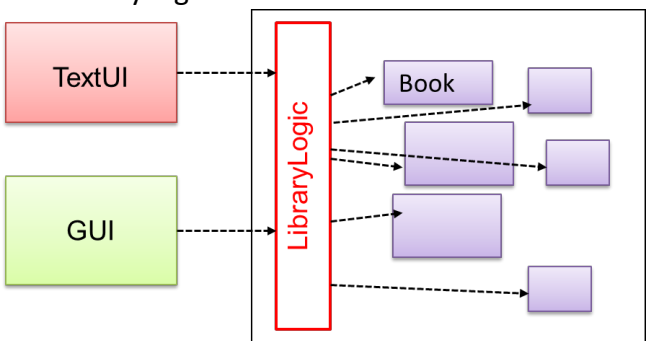
https://nus-cs2103-ay2122s1.github.io/website/schedule/week10/topics.html

Singleton Pattern



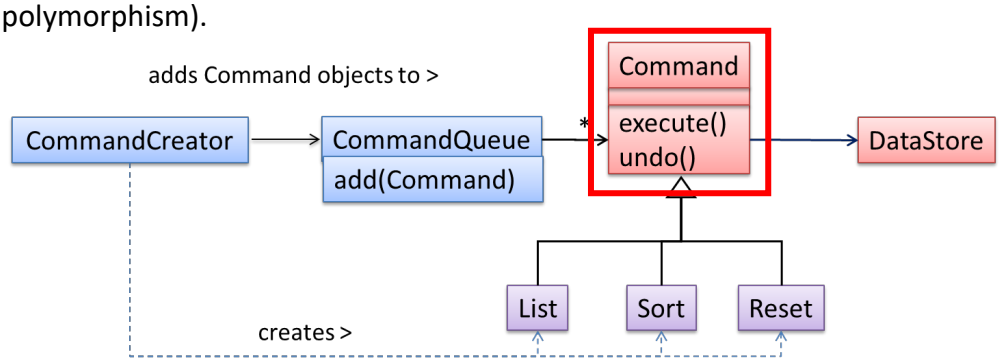
Façade Pattern

The LibraryLogic class is the Facade class.



Command Pattern

The essential element of this pattern is to have a general <<Command>> object that can be passed around, stored, executed, etc without knowing the type of command (i.e. via polymorphism).



https://www.zhihu.com/question/35451773

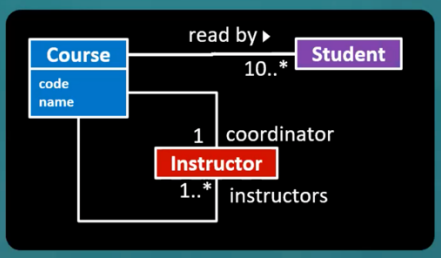
Black-box (aka **specification-based** or **responsibility-based**) approach	Test cases are designed exclusively based on the SUT's specified external behavior, i.e. Acceptance Testing
White-box (aka glass-box or structured or implementation-based) approach	Test cases are designed based on what is known about the SUT's implementation, i.e. the code
Gray-box approach	

Consider the `Logic` component of the Minesweeper application. What are the EPs for the `markCellAt(int x, int y)` method? The partitions in **bold** represent valid inputs.

- `Logic` : PRE_GAME, **READY**, **IN_PLAY**, WON, LOST
- `x` : [MIN_INT..-1] **[0..(W-1)]** [W..MAX_INT] (assuming a minefield size of WxH)
- `y` : [MIN_INT..-1] **[0..(H-1)]** [H..MAX_INT]
- `Cell` at `(x,y)` : **HIDDEN**, MARKED, CLEARED

Q2

Which statements about this OODM are correct?



OODM is a model of the real world.
The design and the implementation of the solution can be guided by it, but it's not the same.
→ **avoid implementation-specific terminology** when discussing the problem domain.

- ☐ A. The `instructors` variable in the `Course` class can be of type `ArrayList<Instructor>`
- ☐ B. The `Course` class can have methods to add students to a specific course.
- ☐ C. There are alternative designs that can be more efficient to implement.
- ☐ D. Although not shown in the diagram, it is likely the `Student` class has attributes such as name, student number, gender etc.
- ☐ E. Instructor objects can play the *role* of coordinator.

Q3

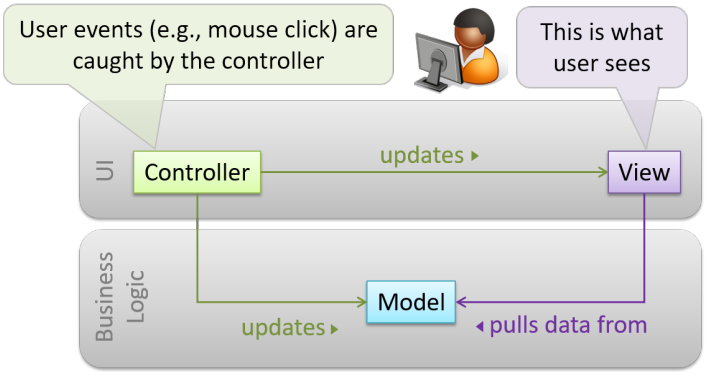
Which can be used for *domain* modelling?

- ☑ An OODM
- ☑ An object diagram
- ☑ An activity diagram
- ☑ A sequence diagram
- ☐ A class diagram
- ☑ A use case diagram

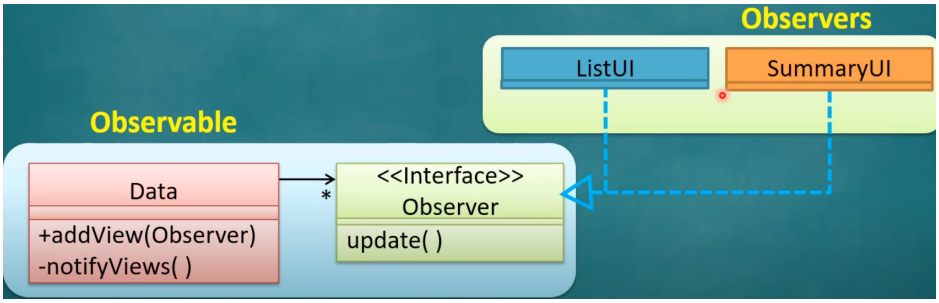
29. Test cases set X has 40 test cases and finds 10 defects in a SUT, Test cases set Y has 20 test cases and finds 10 defects in the same SUT. --> **They are equally effective (finds the same number of defects)** but Y is more efficient as it can find the same number of defects using fewer test cases.
32. More test cases is not always better. We need to consider cost of testing as well.
38. When deciding EPs of a method of an object, the state of the target object should be considered too.

Week 11: Design Patterns(MVC, Observer Pattern), Architectural Styles, QA Techniques, Reuse

https://nus-cs2103-ay2122s1.github.io/website/schedule/week11/topics.html



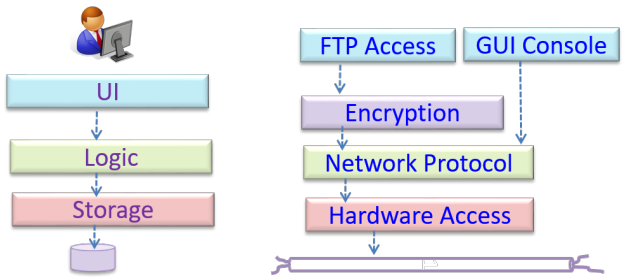
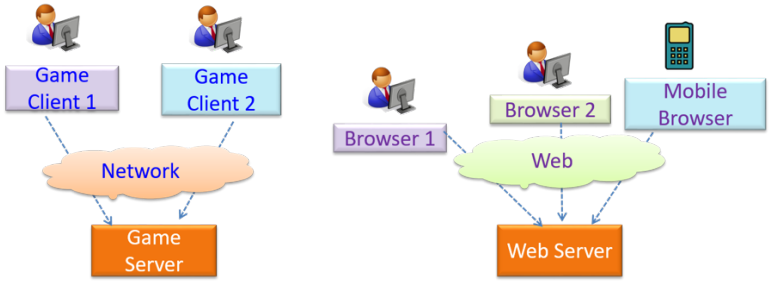
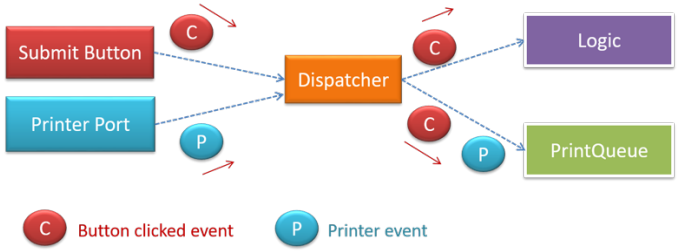
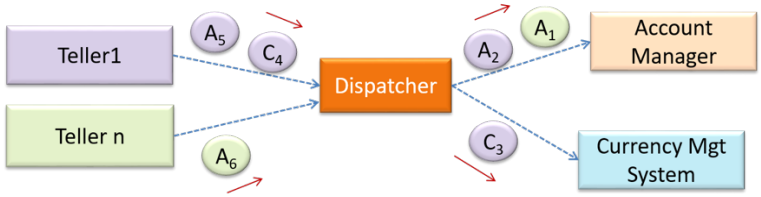
模型 View 不用依赖 Data, View 是独立的

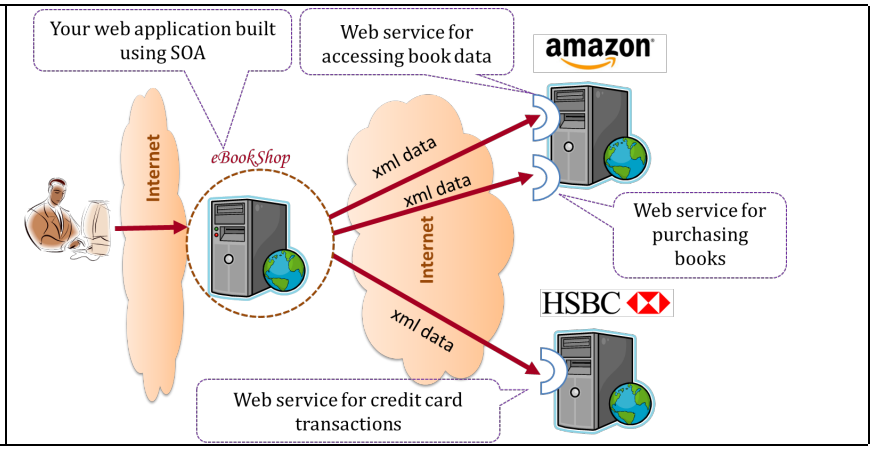



```
Data data = new Data();
ListUI listUI = new ListUI();
SummaryUI summaryUI = new SummaryUI();

// 注意 listUI 和 summaryUI 都 implement 了 Observer 接口，因此可以执行 addView
data.addView(listUI);
data.addView(summaryUI);

public void addView(Observer o) {
   observerList.add(o);
}

// 注意 update 这里用了 polymorphism，ListUI 和 SummaryUI 都分别 override 了 update
public void notifyViews() {
   for (Observer o: observerList) {
```

```
        o.update();
    }
}
```

Software architectures follow various high-level styles (aka architectural patterns)

n-Tier Style	Operating systems and network communication software often use n-tier style. 
Client-Server Style	The online game and the web application use the client-server style. 
Event-driven Style	When the "button clicked" event occurs in a **GUI**, that event can be transmitted to components that are interested in reacting to that event. 
Transaction Processing Style	In this example from a banking system, transactions are generated by the terminals used by tellers, which are then sent to a central dispatching unit, which in turn dispatches the transactions to various other units to execute. 
Service-Oriented Style (SOA)	SOA builds applications by combining functionalities packaged as programmatically accessible services.



Effective 有效(得到较好的效果, 与总量无关) and Efficient 高效(短时间内得到效果)

All combinations strategy: 测试所有组合

At least once strategy: 每个输入至少出现一次

All pairs strategy: 任意选取两个输入，最终的测试用例必须包含这两个输入的所有组合

还可以有其它的测试策略

Heuristic 启发式

正向测试用例(Positive Test Case): 系统支持的输入

反向测试用例(Negative test Case): 系统不支持的输入

每个 valid 有效的输入至少有一次出现在一个**正向测试用例**, 注意关键词: **至少有一次, valid, 正向测试用例**

在一个测试用例不超过一个 invalid 无效的输入, 注意关键词: **不超过一个, invalid**

注意: 若输入存在其它的依赖关系, 则需要额外的测试用例测试

Quality Assurance = Validation + Verification

Validation(确认): are you building the right system i.e., are the requirements correct?

Verification(验证): are you building the system right i.e., are the requirements implemented correctly?

Choose the correct statements about software frameworks.

- [x] a. They follow the Hollywood principle, otherwise known as 'inversion of control'.
- [x] b. They come with full or partial implementations.
- [x] c. They are more concrete than patterns or principles.
- [x] d. They are often configurable.
- [x] e. They are reuse mechanisms.
- [ ] f. They are similar to reusable libraries but bigger.

(a)(b)(c)(d)(e)~~(f)~~

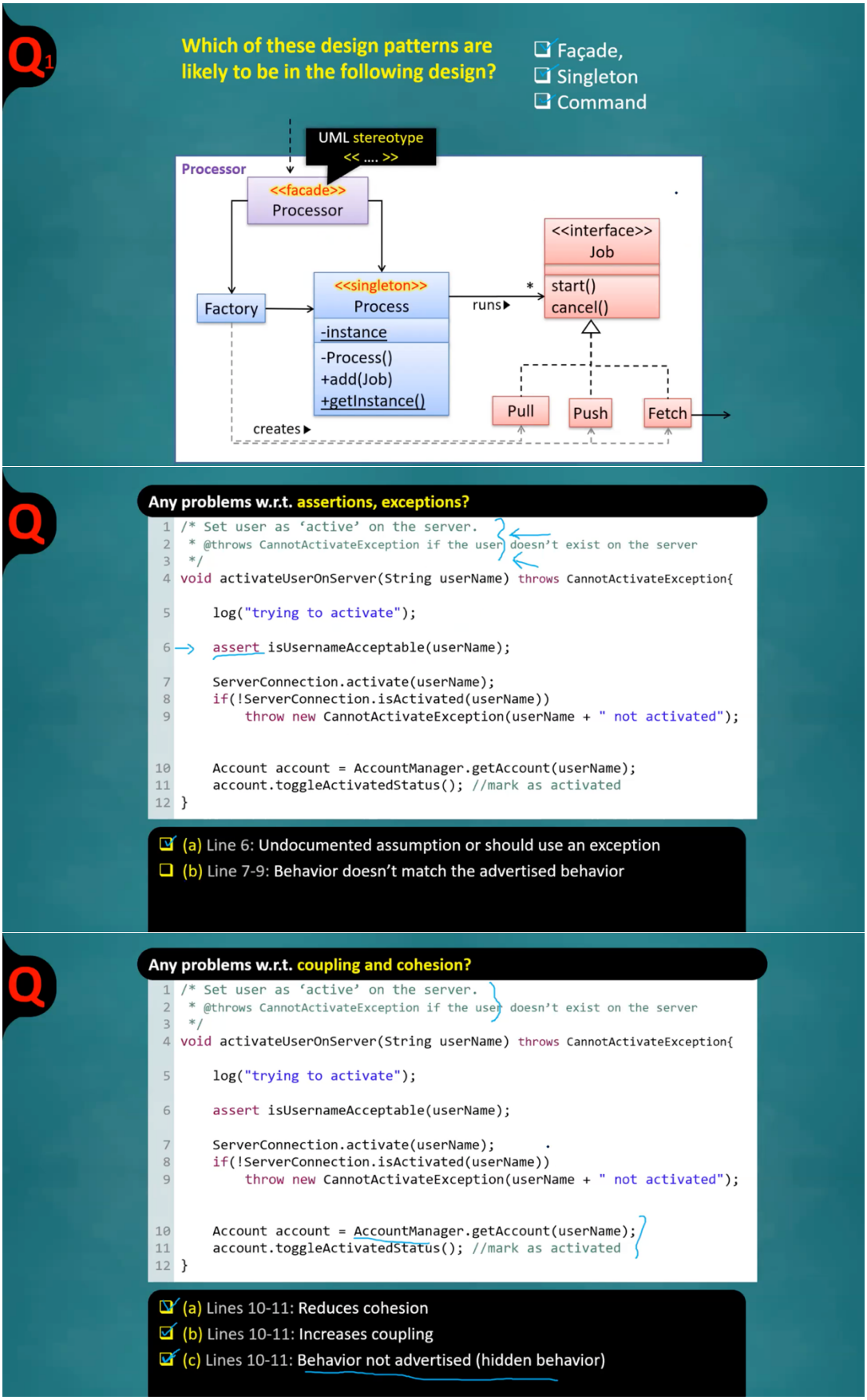
Explanation: While both libraries and frameworks are reuse mechanisms, and both are more concrete than principles and patterns, libraries differ from frameworks in some key ways. One of them is the 'inversion of control' used by frameworks but not libraries. Furthermore, frameworks do not have to be bigger than libraries all the time.

JavaFX is a framework for creating Java GUIs. Tkinter is a GUI framework for Python.

More examples of frameworks

- Frameworks for web-based applications: Drupal (PHP), Django (Python), Ruby on Rails (Ruby), Spring (Java)
- Frameworks for testing: JUnit (Java), unittest (Python), Jest (JavaScript)

我的代码调用 library, 而 framework 调用我的代码(Hollywood principle)



29. Testing all possible combinations **is effective but not efficient**.
31. Formal methods, when compared to testing, is more expensive in general.
33. Whether something belongs under validation or verification is not that important. What is more important is both are done, instead of limiting to verification.
40. A platform comes with a runtime. That is exactly what distinguishes a platform from a library/framework.
41. Libraries are meant to be used 'as is' while frameworks are meant to be customized/extended.