○ [p1.01] requirements: use cases (text)

? Which step in the following use case does something a well-written use case should **not** do?

```
System: PosiIt (an online forum)
Use case: Post comment in a thread
Actor: user
Main success scenario:
    1. User clicks the 'post' button.
    2. PostIt requests for the comment details.
    3. User enters the comment details.
    4. PostIt requests for confirmation.
    5. User confirms.
    6. PostIt saves the comment and displays the updated discussion thread.
  Use case ends
```

Follow-up question: What's the problem?

○ 2

○ 3

○ 4

○ 5

○ 6

✓ 1

Answer to the follow-up question: Contains UI details.

○ [p1.02] Design fundamentals

? Which statement is **incorrect**?

Follow-up question: Why is it incorrect?

- ○ *Abstraction* helps us deal with information overload. We can even use multiple levels of abstraction.
- ○ High *cohesion* is better than low cohesion.
- ○ It is possible to improve both *coupling* and *cohesion* at the same time.
- ○ Loose *coupling* is better than tight coupling.
- ✓ We can achieve *zero coupling* if we do a good job with the design.

Answer to the follow-up question: It is not realistic to achieve *zero* coupling. Different parts of the software need to work together in any non-trivial software.

○ [p1.03] design: about models

? Choose the **incorrect** statement about *models* (e.g., UML diagrams) used in software projects.

Follow-up question: Why is it incorrect?

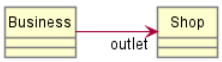
- ○ Models can be used as a tool to deal with complexity of software projects.
- ○ Models can be used for documenting software.
- ○ An architecture diagram is a model.
- ○ A sequence diagram is an abstraction.
- ✓ Models should be as detailed as possible.

Answer to the follow-up question: Models should be as simple as possible, as they are used for dealing with complexity

○ [p1.04] UML: CD: attribute vs association

? Three of these diagrams are equivalent. Which one is different from the other three?

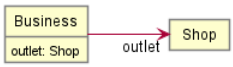
Follow-up question: Why is it different?

○ 

○ 

○ 

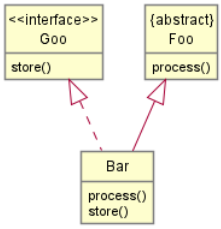
○ 

● 

Answer to the follow-up question: association is also shown as attribute

○ [p1.05] UML: CD: interpret diagram

? Based on the UML diagram below, which statement is **incorrect**? Assume all methods are shown in the diagram and the `Goo` interface doesn't have any default method implementations.



Follow-up question: Why is it incorrect?

○ `Bar` class is overriding the `process()` method.

○ Removing the `process()` method from the `Bar` class will not cause a compile error.

○ Removing the `store()` method from the `Bar` class will cause a compile error.

✓ The code below causes a compile error because we cannot instantiate abstract classes.

```
Foo foo = new Bar();
```

Answer to the follow-up question: Abstract classes can be used as a type.

☐ [p1.06] UML: OD to CD

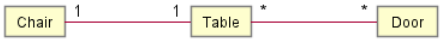
?

[**SELECT ALL**] This object diagram is compliant with which class diagrams?

Follow-up question: Can an object diagram show Navigability?

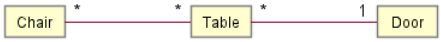
☑

☑



☐

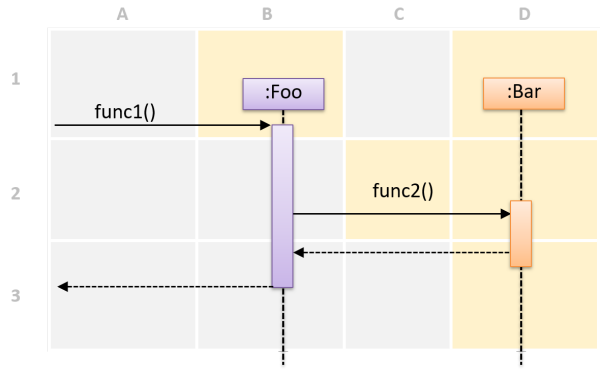
☐



Answer to the follow-up question: Yes

☐ [p1.07] UML: SD: detect notation errors

? [SELECT ALL] Which cells contain notation errors?



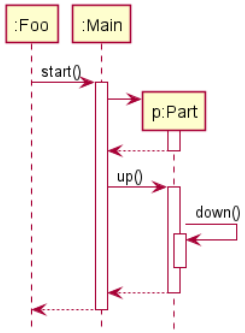
Follow-up question: Which of the objects existed before the interaction began?

- ☐ B1
- ☐ C2
- ☐ D1
- ☑ D2
- ☑ D3

Answer to the follow-up question: Both objects (examiner comment: the activation bar starts too early in D2, and ends too late in D3)

○ [p1.08] UML: SD: Interpret notation

? Choose the **incorrect** statement about this diagram.



Follow-up question: Why is it incorrect?

○ The `down()` method belongs to the `Part` class.

○ Only one object in the diagram has been given a name.

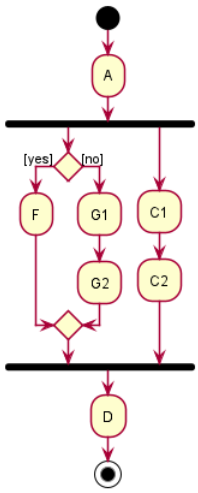
○ There were only two objects when this interaction started.

✓ The `Main` object is calling its own `start()` method.

Answer to the follow-up question: The start() method is called by the Foo object.

☐ **[p1.09]** UML: AD: interpret forks and branches

?



[**SELECT ALL**] Which sequences are **allowed** by this activity diagram?

Follow-up question: For one of the sequences not allowed, explain why it is not allowed.

☐ A C1 C2 D

☑ A C1 C2 F D

☑ A C1 C2 G1 G2 D

☑ A C1 F C2 D

Answer to the follow-up question: A C1 C2 D: both parallel paths need to complete

☐ [p1.10] code/design quality attributes

? [**SELECT ALL**] Which of these are to be avoided/minimized?

Follow-up question: Which two are more related to coding than design?

- ☐ SLAP
- ☐ cohesion
- ☐ abstraction
- ☑ magic numbers
- ☑ coupling

Answer to the follow-up question: magic literals, SLAP

☐ [p1.11] Java: coding standard

? [SELECT ALL] Which three lines in the following code have **coding standard violations** (as per the coding standard using in the module)?

You may ignore any violations related to indentation, or spacing around operators.

```
1  /*
2   * Print the given string with the prefix ">".
3   *
4   * @param abcd String to be printed.
5   */
6  public void printWithPrefix(String abcd) {
7      if(abcd.length() > 3)
8          print(">" + abcd);
9  }
```

Follow-up question: For one of the lines you selected as an answer, state what needs to change to make it comply with the coding standard.

- ☐ 4
- ☐ 6
- ☐ 8
- ☑ 1
- ☑ 2
- ☑ 7

Answer to the follow-up question: line 1: /* -> /** or line 2: Print -> Prints or line 5: missing {

○ [p1.12] implementation: refactoring

? Given below are some changes Tom did to his code. Which one is **not** a *refactoring*?

Follow-up question: Why is it not a refactoring?

○ Tom merges the `Greet` class and the `Prompt` class into one bigger class.

○ Tom finds that the variable name `find` is misleading. He changes it to `isFound`.

○ Tom thinks the `add()` function is too long. He applies the SLAP technique to it.

○ Tom removes braces around an `if` block because there is only one statement in the block.

✓ Tom finds the `sort()` function doesn't work when the list is empty. He adds an exception to handle that case.

Answer to the follow-up question: it changes the behavior

○ [p1.13] Java: JUnit keywords

? Which of these Java keywords or method calls is **not** likely to be in the UI class of your software?

Follow-up question: Why not?

- ○ `throws`
- ○ `catch`
- ○ `assert`
- ○ `throw`
- ○ `instanceof`
- ✓ `assertEquals`

Answer to the follow-up question: This is a JUnit method; should only be used in test classes.

○ [p1.14] testing: types, test case design

? Which of these is **most** likely to be *glass-box* testing?

Follow-up question: Why?

- ○ system testing
- ○ acceptance testing
- ○ alpha testing
- ○ beta testing
- ✓ unit testing

Answer to the follow-up question: only unit testing is done by developers

○ [p1.15] PM: statements about rcs

? Which statement is **incorrect**?

Follow-up question: Why is it incorrect?

○ A Git repository can *pull* from one remote repository and *push* to a different remote repository, provided all repos involved have a shared history.

○ When you *push* code to your *fork*, any PRs created from it get updated with the latest code.

○ GitHub is an online service that offers Git features and more.

○ It is possible to edit the commit message of a past commit.

✓ *Forking* creates a local copy of a repository.

Answer to the follow-up question: Cloning (not forking) a remote repo creates a local copy.

○ [p1.16] PM: statements about rcs

? Which statement is **incorrect**?

Follow-up question: Why is it incorrect?

- ○ An architecture diagram can use emoji symbols to represent components of a system.
- ○ The *n-Tier* architectural style is also known as the *layered* architectural style.
- ○ The MVC pattern is an example of the *Separation of Concerns* principle being applied.
- ○ GitHub Actions plays the role of a *continuous integration* tool.
- ✓ High-level *user stories* that cover bigger functionalities are called *legends*.

Answer to the follow-up question: They are called epics or themes.