Review Test Submission: Chapters 8 - 10

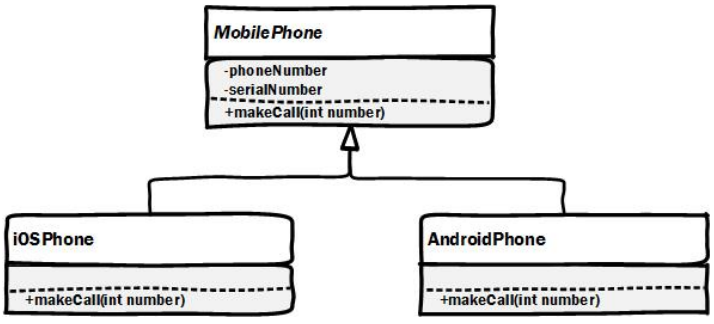
User	Kevin Li
Course	SE-310-001/003-XLIST-201915
Test	Chapters 8 - 10
Started	10/20/19 1:41 PM
Submitted	10/20/19 2:24 PM
Due Date	10/21/19 11:59 PM
Status	Completed
Attempt Score	40 out of 120 points
Time Elapsed	42 minutes
Instructions	This will not count as a real quiz. If you complete it and get over a 70 you will get full credit. If you do not get over a 70, simply retake it. It is meant for a review and to reinforce concepts introduced in the JAVA review.
Results Displayed	Submitted Answers, Feedback, Incorrectly Answered Questions

Question 1

0 out of 10 points

This is one of a series of true/false questions relating to the following UML and encapsulation. It the following statement true or false:

`MobilePhone`'s attribute `phoneNumber` is an example of encapsulation.



Selected Answer: False

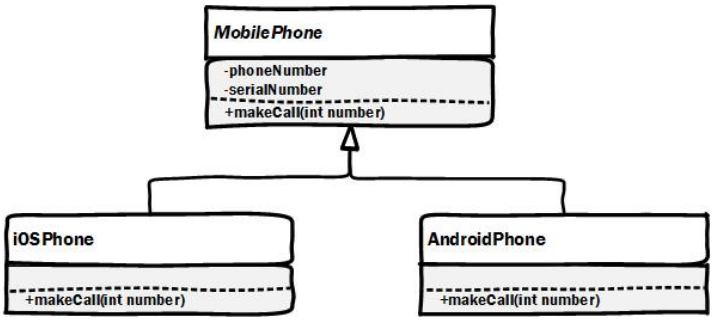
Response Feedback: This example is the most straightforward form of encapsulation. `phoneNumber` is a private variable in a class. It definitely is a form of encapsulation.

Question 2

10 out of 10 points

This is one of a series of true/false questions relating to the following UML and encapsulation. It the following statement true or false:

`MobilePhone`'s method `makeCall` is an example of encapsulation, but `iOSPhone's` method `makeCall` is not because the class `iOSPhone` has no classed derived from it.



Selected Answer: False

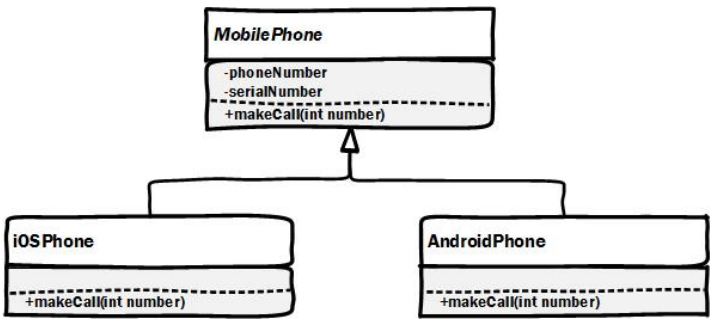
Response Feedback: Both methods exhibit encapsulation. Whatever the details of the actual method are encapsulated in both cases. It's possible that `iOSPhone`'s `makeCall` method also encapsulates the details of `MobilePhone`'s `makeCall` method as well.

Question 3

10 out of 10 points

This is one of a series of true/false questions relating to the following UML and encapsulation. It the following statement true or false:

An object declared as a `MobilePhone` but instantiated as an instance of the class `iOSPhone` is an example of encapsulation.



Selected Answer: True

Response Feedback: A derived class is an example of encapsulation. Since `iOSPhone` is derived from `MobilePhone` an object declared of type `MobilePhone` could be of type `iOSPhone` or `AndroidPhone`. Therefore, the details of what class and is instantiated is encapsulated in the class hierarchy with `MobilePhone` as the base class.

Question 4

0 out of 10 points

Given the following image: The _____ class is an abstraction and the _____ class is an implementation.

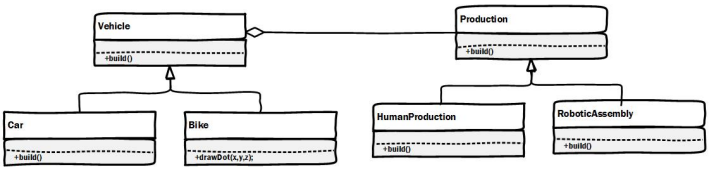
Selected Answers: House, 3DPrinter respectively.
3DPrinter, House respectively.

Response Feedback: An abstraction is a representation of an object without a manifestation. The 3DPrinter could be used to materialize the house. Therefore the house is the abstraction and the 3DPrinter is the implementation.

Question 5

10 out of 10 points

For the following figure, which statement is true:



Selected Answer: This is a bridge pattern.
Vehicle is an abstraction in a bridge pattern.
car and bike are refined abstraction in a bridge pattern.
Production is an implementation in a bridge pattern.
HumanProduction and RoboticAssembly are refined implementations in a bridge pattern.

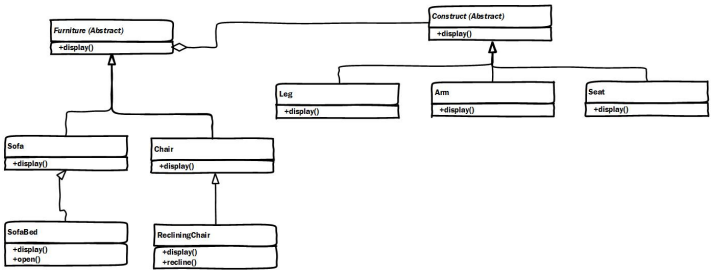
Response Feedback: The `Vehicle` hierarchy defines the characteristics of what is to be manifested, in this case a `Car` or `Bike`. If details were added there would be a parts list and a list of steps as to where each part goes. Regardless of what is being built, it could be assembled by

humans or machines that would take the same parts list and list of steps. Therefore the `Production` hierarchy is the implementation. `Car` and `Bike` are refinements of a vehicle, therefore the vehicle is the abstraction and the `Car` and `Bike` are refined abstractions. `HumanProduction` and `RoboticAssembly` are refinements of `Production` therefore `Production` is an implementation and `HumanProduction` and `RoboticAssembly` are refined implementations.

Question 6

0 out of 10 points

For the following figure, which statement is true:



Selected Answer:

This is a bridge pattern.
`Furniture` is an abstraction in a bridge pattern.
`Sofa` and `Chair` are refined abstractions in a bridge pattern.
`Construct` is an implementation in a bridge pattern.
`Leg` and `Arm` are refined implementations in a bridge pattern.

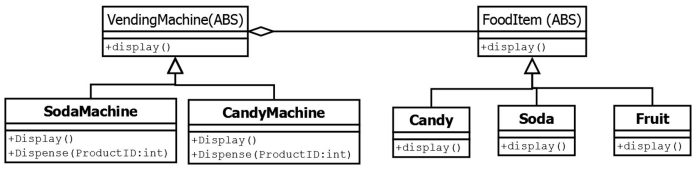
Response Feedback:

This is not a bridge. Although `Furniture` is indeed an abstraction and `Sofa` and `Chair` are refinements of that abstraction, the class hierarchy to the right are not implementations of the abstraction to the left. They are simply components of that abstraction. Therefore, this UML is only showing inheritance with composition.

Question 7

0 out of 10 points

For the following figure, which statement is true:

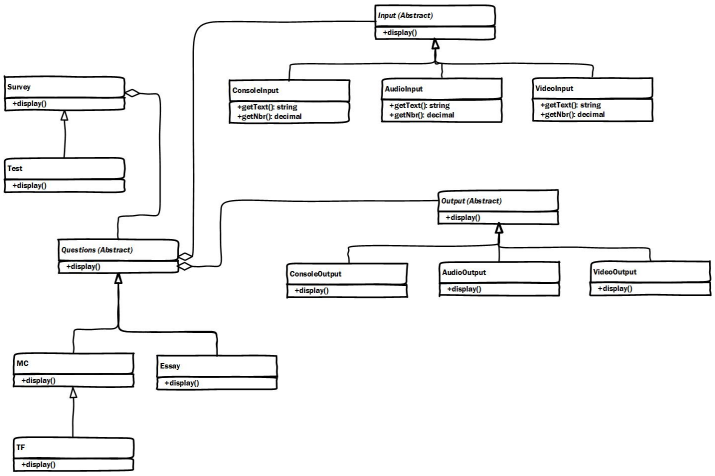


Selected Answer: This is a Bridge.
`VendingMachine` is an implementation in a bridge pattern.
`SodaMachine` and `CandyMachine` are refined implementations in a bridge pattern.
`FoodItem` is an abstraction in a bridge pattern.
`Candy`, `Soda`, and `Fruit` are refined abstractions in a bridge pattern.

Question 8

0 out of 10 points

For the following figure, which statement is true:



Selected Answer: This is a bridge.
`Questions` is a refined abstraction in a bridge pattern.
`MC` and `Essay` are abstractions in a bridge pattern.
`Input` and `Output` are refined implementations in a bridge pattern.
`ConsoleInput`, `ConsoleOutput`, `AudioInput`, `AudioOutput`, `VideoInput`, etc.. are implementations in a bridge pattern.

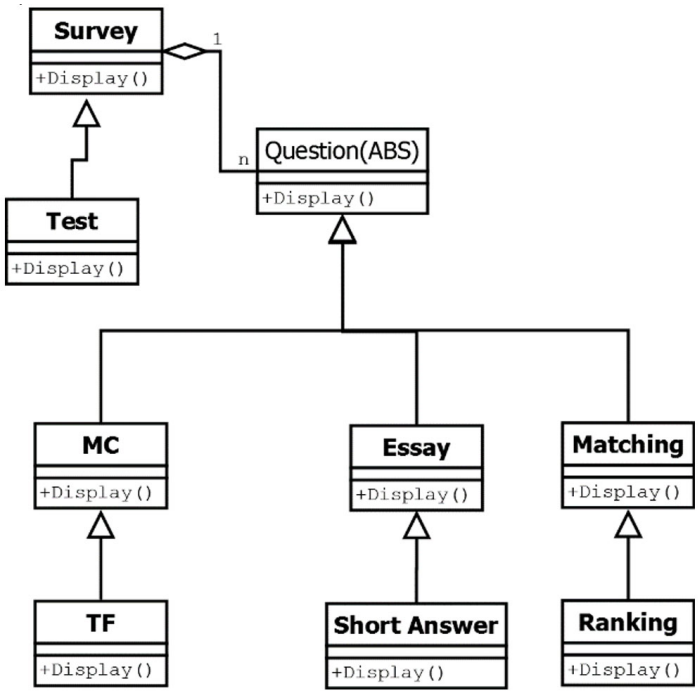
Response Feedback: This is a bridge. The `Questions` hierarchy defines the characteristics of what is to be manifested, in this case a `MC` or `Essay`. If details were added there would be a choices for the multiple choice question. Regardless of what type of question is being created we could have different formats to its manifestation. Therefore the `Output` and `Input` hierarchies are the implementation. `ConsoleOutput, ConsoleInput,`

`AudioOutput`, etc... are refinements of a `Input` and `Output`, therefore the Questions is the abstraction and the `MC` and `Essay` are refined abstractions. `ConsoleOutput`, `ConsoleInput` and `AudioOutput` are refinements of `Input` and `Output` therefore `Input` and `Output` are implementations and `ConsoleOutput`, `ConsoleInput` and `AudioOutput` etc are refined implementations.

Question 9

0 out of 10 points

For the following figure, which statement is true:



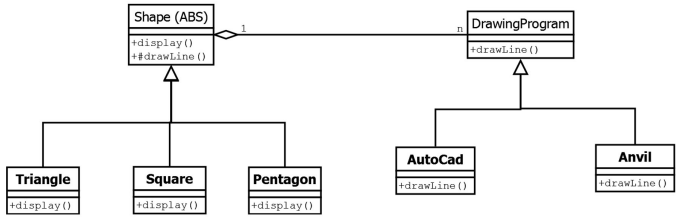
Selected Answer: This is a bridge.
`Survey` is an implementation in a bridge pattern.
`Test` is a refined implementation in a bridge pattern.
`Question` is an abstraction in a bridge pattern.
`MC`, `Essay`, `Matching` , etc.. are refined abstractions in a bridge pattern.

Response Feedback: A `Survey` contains `Questions`. This is not the same as being implemented by the `Questions` class. Both the Survey/Test hierarchy and the Questions/MC.... hierarchy are classes that represent abstractions. There are no implementation or refined implementation classes shown in the figure. Therefore this is not an example of a bridge pattern.

Question 10

0 out of 10 points

For the following figure, which statement is true:



Selected Answer:

This is a bridge.
`Shape` is a refined abstraction in a bridge pattern.
`Triangle`, `Square`, `Pentagon` are abstractions in a bridge pattern.
`DrawingProgram` is a refined implementation in a bridge pattern.
`AutoCad`, `Anvil` are implementations in a bridge pattern.

Response Feedback:

The `Shape` hierarchy defines the characteristics of what is to be manifested, in this case a `Triangle`, `Square` or `Pentagon`. Regardless of what is being created, it could be displayed by different CAD programs. . Therefore the `DrawingProgram` hierarchy is the implementation. `Triangle, Square` and `Pentagon` are refinements of a `Shape`, therefore the `Shape` is the abstraction and the `Triangle, Square` and `Pentagon` are refined abstractions. `AutoCad` and `Anvil` are refinements of `DrawingProgram` therefore `DrawingProgram` is an implementation and `Triangle, Square` and `Pentagon` are refined implementations.

Question 11

0 out of 10 points

The bridge pattern does not promotes loose coupling because the abstractions are tightly coupled to the implementations by the interface.

Selected Answer: True

Response Feedback:

False. The brdige pattern promotes loose coupling. It's in the definition. Separate the abstraction from the implementation so the two can vary independently. As long as you don't vary the interface you can add as many abstractions or implementations without having to modify any other classes.

Question 12

10 out of 10 points

The strategy pattern does not promotes loose coupling because the clients are tightly coupled to the algorithms by the interface.

Selected Answer: False

Response Feedback: The strategy pattern promotes loose coupling. It's in the definition. Separate the client from the algorithms so the two can vary independently. As long as you don't vary the interface you can add as many algorithms or clients without having to modify any other classes.

Sunday, October 20, 2019 2:24:20 PM EDT

← OK