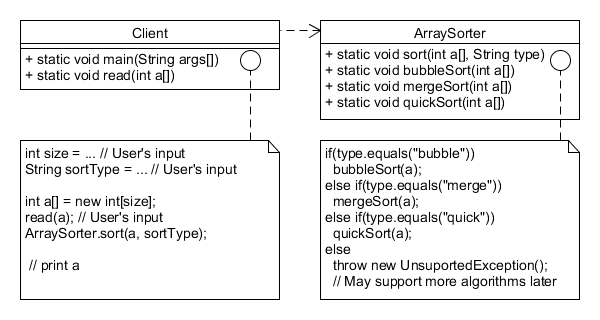
Answer all of the questions in this quiz using the following design of an algorithm sorting module as a reference.



**Question 1**

Partially correct

2.20 points out of 4.00

question

Question text

Which of the following design principles have been violated in the design of the sorting module.

Select one or more:

a. The **O**pen-**C**losed **P**rinciple (OCP)

b. Favor composition over inheritance

This principle is not applicable because we neither use inheritance, nor composition.

c. Identify the aspects of your application that vary and separate them from what stays the same

Here the sorting algorithms may vary but it is not separated from the rest of the code of the **ArraySorter** class.

d. Strive for loosely coupled designs between classes/objects that interact

The **Client** class is highly coupled with the class-specific implementation of the**ArraySorter.sort()** class.

e. The **D**ependency **I**nversion **P**rinciple (DIP)

There is no common abstraction (supertype) that  the Client or ArraySorter classes depend on. The Client class is coupled with class-specific (static) implementation of the **sort()** method.

Feedback

Your answer is partially correct.

You have correctly selected 3.

The correct answer is: Identify the aspects of your application that vary and separate them from what stays the same, The **D**ependency **I**nversion **P**rinciple (DIP), Strive for loosely coupled designs between classes/objects that interact, The **O**pen-**C**losed **P**rinciple (OCP)

**Question 2**

Partially correct

3.33 points out of 4.00

Flag question

Question text

What kind of coupling is present in the current design?

Select one or more:

a. Control

Yes, because the type parameter regulates which branch of the **sort()** method gets executed.

b. External

c. No Coupling

d. Content

No because the array is not an internal or local data of **Client**. **Client** does not store the array in any fields.

e. Message

f. Common

g. Data

Yes, because **Client** and **ArraySorter** share the array to be sorted through parameters.

h. Stamp

Feedback

Your answer is partially correct.

You have selected too many options.

The correct answer is: Control, Data

**Question 3**

Partially correct

3.00 points out of 4.00

Flag question

Question text

What kind of cohesion is present in the current design?

Select one or more:

a. Procedural

b. Coincidental

c. Logical

Yes, because they logically do the same thing in **ArraySorter**. We can, however, further modularize this class to allow implementation variation through polymorphism.

d. Communicational

Yes, all of the sort methods work on the same data.

e. Sequential

No! Input and output of methods within the same class are not linked.

f. Temporal

g. Functional

Yes, because methods in the two classes contribute to their own single well-defined purpose of their corresponding class.

Feedback

Your answer is partially correct.

You have selected too many options.

The correct answer is: Logical, Communicational, Functional

**Question 4**

Incorrect

0.00 points out of 2.00

Flag question

Question text

Which of the following pattern(s) is(are) appropriate to improve the current design?

Select one or more:

a. Decorator

b. Observer

Notification mechanism is not-critical in the current context, so, Observer is not the right approach.

c. Strategy

d. None of these patterns are appropriate. There is probably another one which could help.

Feedback

Your answer is incorrect.

The correct answer is: Strategy

**Question 5**

Complete

Points out of 6.00

Flag question

Question text

Re-design the existing system and turn in an improved version of your design here. You must turn-in the following:

1. Design.png (image file). Here is the UMLet file [[uxf](http://moodle.rose-hulman.edu/pluginfile.php/253625/mod_folder/content/0/Design.uxf?forcedownload=1)] of the existing design if you want to use it rather than starting from scratch.
2. In a few lines, describe your design and explain why it is better than the original one.

The new design is very similar to the old design. The client still takes the input data, reads it into an array, calls the ArraySorter, and then prints the sorted data. The ArraySorter only has one method now, sort(int a[], String type). This sort method passes the string to its map and will return an ISorter as the key which is then used to sort the array. If no key is returned, then that type of sort is not supported. The ISorter interface has a sort(int a[]) method which is customized by each Sorter that implements the interface. Each implementation of the interface customizes its sort method with the appropriate algorithm.

My design is better than the original because you remove a lot of duplicate code and increase maintainability and modification. Mapping the string to an ISorter gets rid of all of the if/else statements. It also allows you to more easily add a new Sorter since all you have to do is add that sort type's name and corresponding sorter to the map to implement the new algorithm. Interfacing the ISorter also makes it easier to implement new sorting algorithms because you only have to make a new implementation of the ISorter before customizing the sort method with a few lines of code. You don't have to keep modifying the ArraySorter class and adding an if/else statement.

[[Image (PNG)](http://moodle.rose-hulman.edu/pluginfile.php/262007/question/response_attachments/122733/6/2892057/NewDesign.png?forcedownload=1) NewDesign.png](http://moodle.rose-hulman.edu/pluginfile.php/262007/question/response_attachments/122733/6/2892057/NewDesign.png?forcedownload=1)