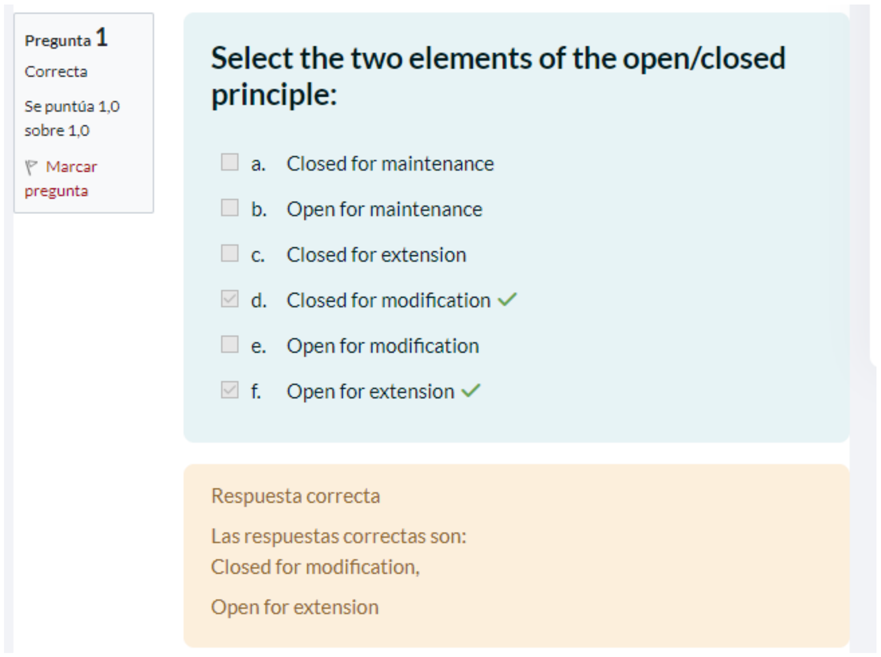
Partial correction:

Software Modeling

1.



Pregunta 1
Correcta
Se puntúa 1,0 sobre 1,0
Marcar pregunta

Select the two elements of the open/closed principle:

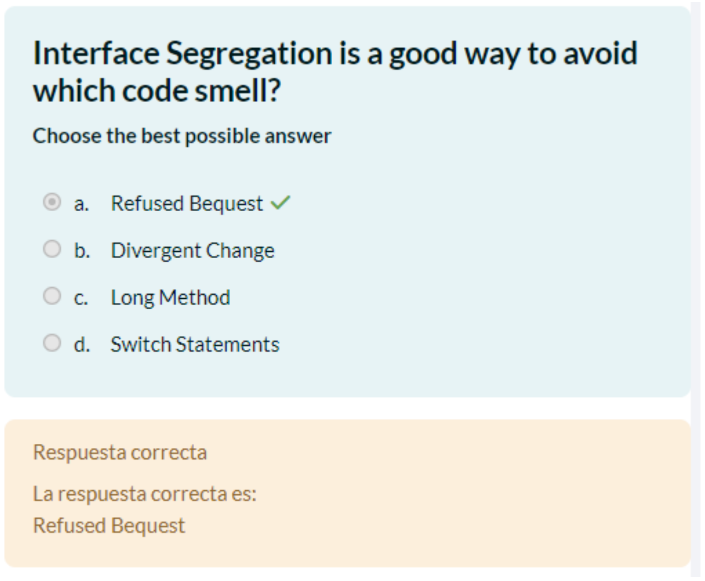
- [ ] a. Closed for maintenance
- [ ] b. Open for maintenance
- [ ] c. Closed for extension
- [✓] d. Closed for modification ✓
- [ ] e. Open for modification
- [✓] f. Open for extension ✓

Respuesta correcta
Las respuestas correctas son:
Closed for modification,
Open for extension

Justification:
I chose: Closed for modification and Open for extension.

Because the SOLID (Open/Closed) principle establishes that the code must be open for extension, that is, we must write it so that we can add new functionality, without changing the existing code that is closed for modification.

2.


Interface Segregation is a good way to avoid which code smell?

Choose the best possible answer

[✓ a. Refused Bequest ✓]
b. Divergent Change
c. Long Method
d. Switch Statements

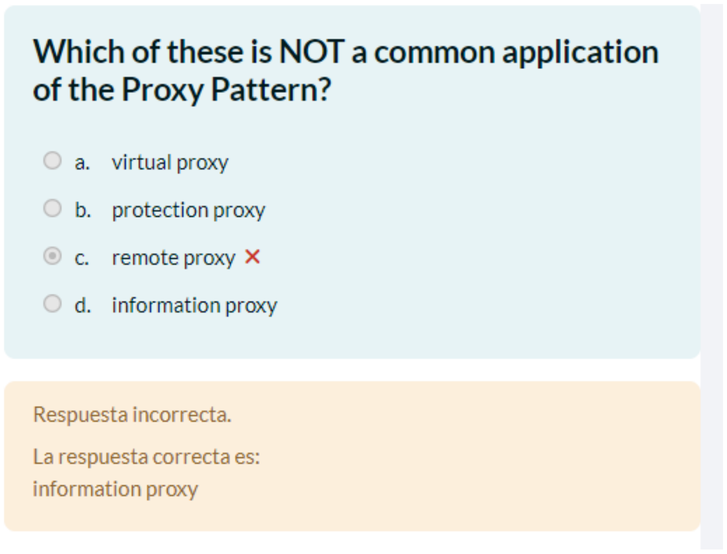
Respuesta correcta

La respuesta correcta es:
Refused Bequest

Justification:
I chose: Legacy rejected.

Because of the Interface Segregation Principle, as it suggests that clients should not be forced to rely on interfaces they do not use, i.e., that an interface should have only the methods that are relevant to the implementation class. And that's why you can avoid code odors related to unnecessary dependencies and bloated interfaces.

3.


Which of these is NOT a common application of the Proxy Pattern?

a. virtual proxy
b. protection proxy
[✓ c. remote proxy ✗]
d. information proxy

Respuesta incorrecta.

La respuesta correcta es:
information proxy

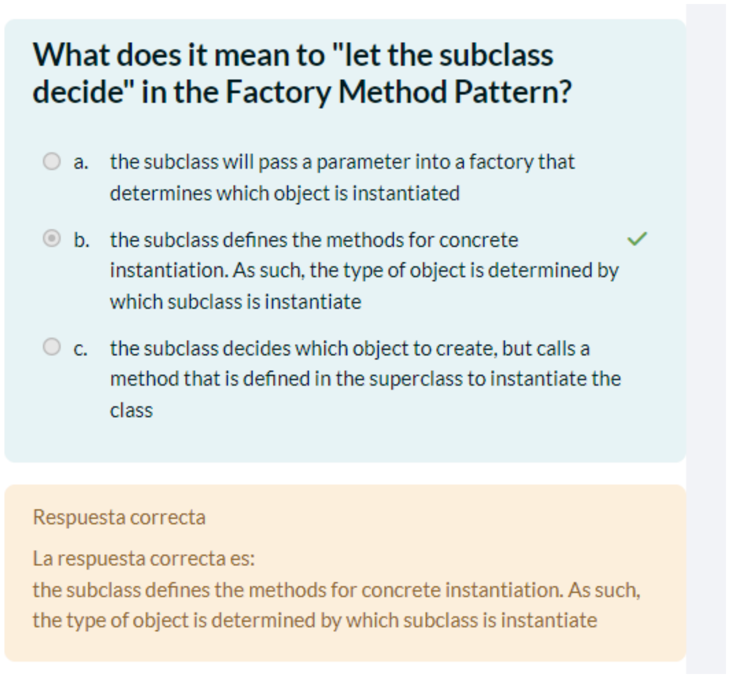
Justification:

The Proxy pattern is used to control access to an object or to provide a substitute for an expensive or remote object. Common types of proxy include Virtual Proxy, which creates expensive objects only when necessary; the Protection Proxy, which controls access to an object; and Remote Proxy, which represents objects in a different address space, such as remote objects on a network or web services. However, information proxy is not a common use of the Proxy pattern.

Error:

Error: select the remote proxy, since it was the only one I thought I had not heard of, and consider that the information proxy referred to the chace memory.

4.



What does it mean to "let the subclass decide" in the Factory Method Pattern?

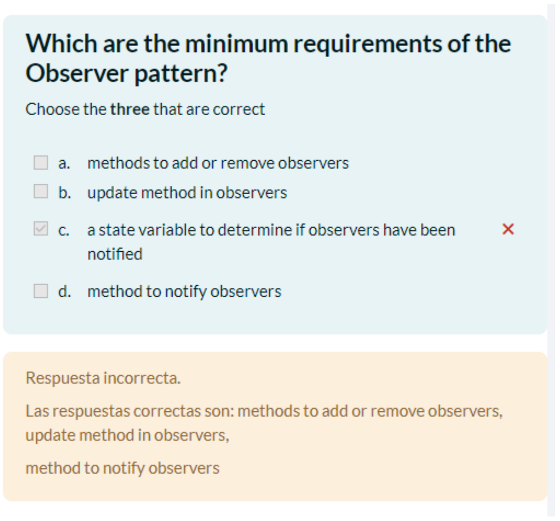
- a. the subclass will pass a parameter into a factory that determines which object is instantiated
- [✓ b.] the subclass defines the methods for concrete instantiation. As such, the type of object is determined by which subclass is instantiate ✓
- c. the subclass decides which object to create, but calls a method that is defined in the superclass to instantiate the class

Respuesta correcta

La respuesta correcta es:
the subclass defines the methods for concrete instantiation. As such, the type of object is determined by which subclass is instantiate

Justification:
Select option b, because in the Factory Method pattern, subclasses define the methods for the specific creation. That is, each specific subclass implements its own factory method to create specific objects.

5.



Which are the minimum requirements of the Observer pattern?

Choose the **three** that are correct

- [ ] a. methods to add or remove observers
- [ ] b. update method in observers
- [✓] c. a state variable to determine if observers have been notified ✗
- [ ] d. method to notify observers

Respuesta incorrecta.

Las respuestas correctas son: methods to add or remove observers, update method in observers,

method to notify observers

Justification:

Methods for adding or removing observers: The Observer pattern implies that an object the subject, maintains a list of observers interested in its changes. Therefore, it is necessary to have methods to add and remove observers from this list.

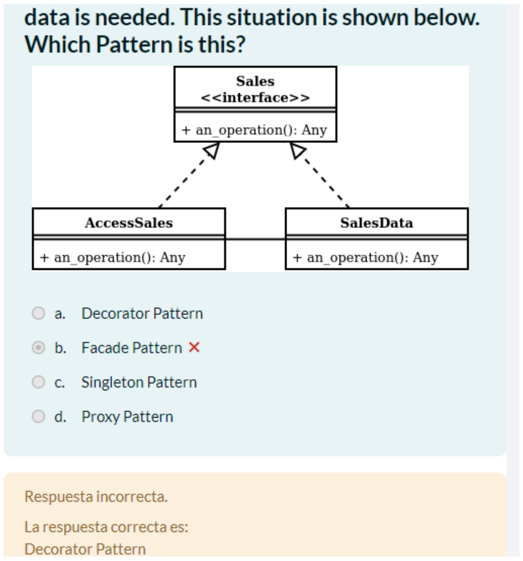
Update method in observers: Observers must have an update method that is called when the subject changes its state. This method allows observers to react to relevant changes.

Method for notifying observers: When the subject changes its status, it must notify all registered observers. Therefore, a method is needed to notify observers about changes.

Error:

I don't know how I could have been wrong because a state variable to determine if the observers have been notified is not something necessary to implement it.

6.


data is needed. This situation is shown below. Which Pattern is this?

a. Decorator Pattern

[✓ b. Facade Pattern] ✗

c. Singleton Pattern

d. Proxy Pattern

Respuesta incorrecta.

La respuesta correcta es:
Decorator Pattern

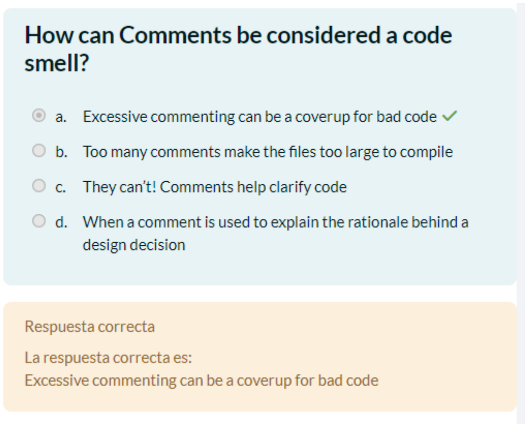
Justification:

The Decorator Pattern is best suited for this case because it allows adding additional functionality to an existing class without modifying its original structure. In other words, you can "decorate" an object with extra features without altering its code base. In this case, the object to decorate is the SalesData class. The pattern allows you to extend the capabilities of SalesData using decorators such as AccessSales, which provides curated sales data based on user credentials.

Error:

I realized that I did not understand the question well, or well I could not translate it correctly, since I took it as if two different calls were made to the interface depending on what we wanted to do.

7.


How can Comments be considered a code smell?

[✓ a. Excessive commenting can be a coverup for bad code] ✓

b. Too many comments make the files too large to compile

c. They can't! Comments help clarify code

d. When a comment is used to explain the rationale behind a design decision

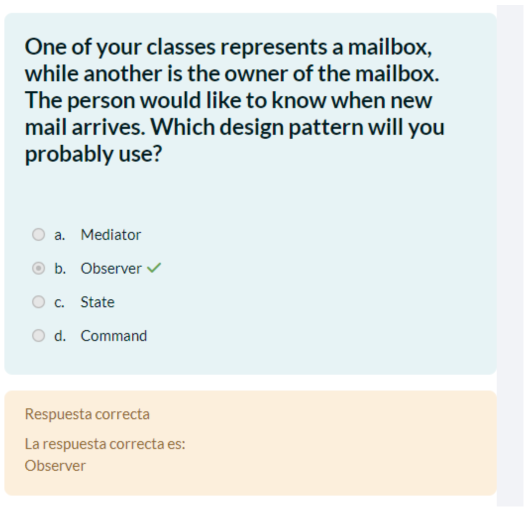
Respuesta correcta

La respuesta correcta es:
Excessive commenting can be a coverup for bad code

Justification:

I select Excessive Commenting, because if a code has too many comments, it is a red flag, as it could be a sign that the code itself is not clear enough. And the programmer, realizing that his code was not readable or self-explanatory, decided to go crazy and explain the code through comments.

8.



One of your classes represents a mailbox, while another is the owner of the mailbox. The person would like to know when new mail arrives. Which design pattern will you probably use?

a. Mediator
[✓ b. Observer ✓]
c. State
d. Command

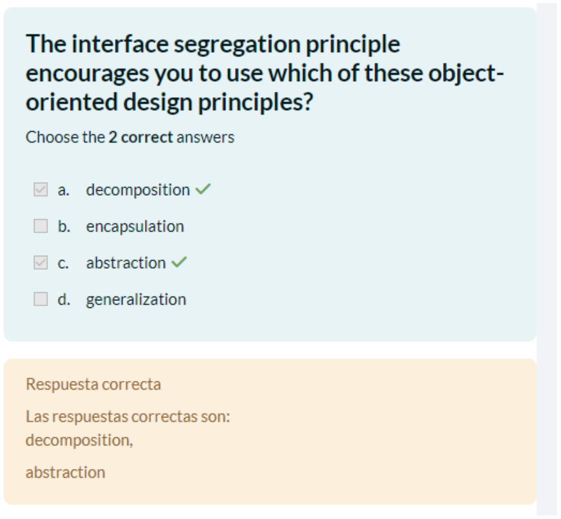
Respuesta correcta

La respuesta correcta es:
Observer

Justification:

The Observer Pattern is the most suitable for this case, since it allows establishing a one-to-many relationship between objects. In this scenario, we have one class that represents a mailbox and another that is the owner of the mailbox. The owner wants to know when new mail arrives in the mailbox. The Observer Pattern allows the owner (observer) to register to receive notifications when the mailbox (subject) changes its state. The mailbox will automatically notify the owner when there are changes.

9.



The interface segregation principle encourages you to use which of these object-oriented design principles?

Choose the **2 correct** answers

- [✓ a. decomposition] ✓
- b. encapsulation
- [✓ c. abstraction] ✓
- d. generalization

Respuesta correcta

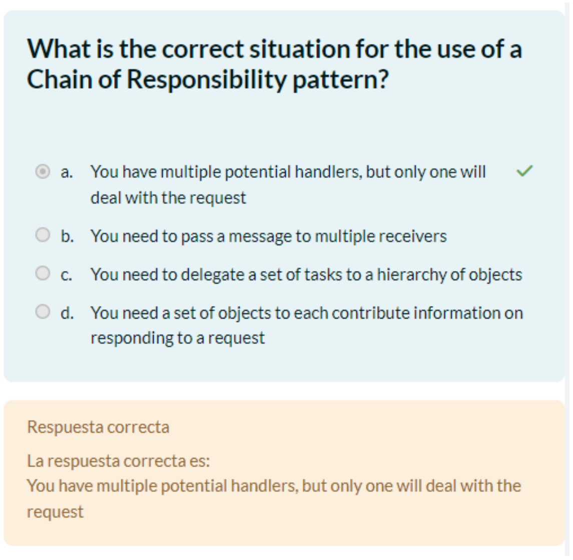
Las respuestas correctas son:
decomposition,

abstraction

Justification:

The Interface Segregation Principle suggests that we should divide large interfaces into smaller, specific parts (decomposition) and create abstract interfaces that are relevant to each client (abstraction). This improves cohesion and prevents classes from implementing unnecessary methods.

10.



What is the correct situation for the use of a Chain of Responsibility pattern?

- [✓ a. You have multiple potential handlers, but only one will deal with the request] ✓
- b. You need to pass a message to multiple receivers
- c. You need to delegate a set of tasks to a hierarchy of objects
- d. You need a set of objects to each contribute information on responding to a request

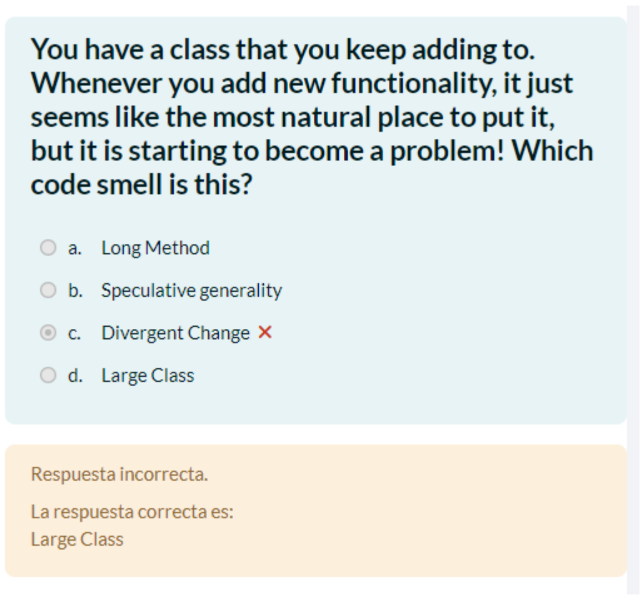
Respuesta correcta

La respuesta correcta es:
You have multiple potential handlers, but only one will deal with the request

Justification:

a, is the correct option and that by having multiple potential handlers, but only one will be in charge of the request, it is generally the case of using the “Chain of Responsibility” design pattern since this is used when you have a chain of objects that can handle a request, and each object in the chain decides whether it can handle the request or should pass it to the next object in the chain.

11.



You have a class that you keep adding to. Whenever you add new functionality, it just seems like the most natural place to put it, but it is starting to become a problem! Which code smell is this?

- a. Long Method
- b. Speculative generality
- [✓ c. Divergent Change] ✕
- d. Large Class

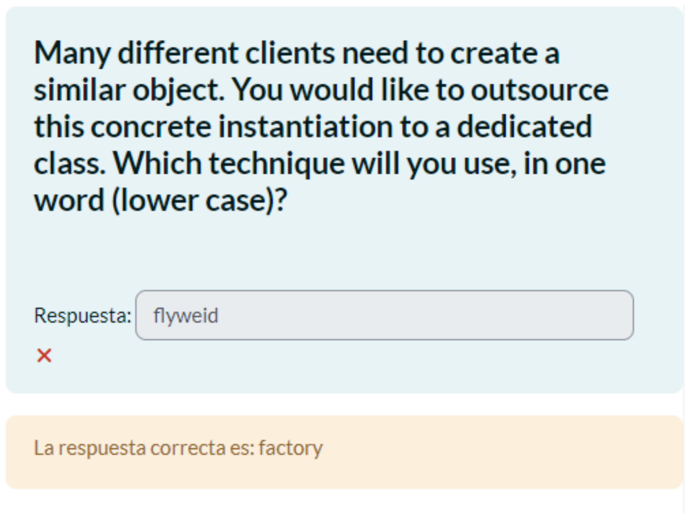
Respuesta incorrecta.

La respuesta correcta es:
Large Class

Justification:

The correct answer is the Large Class odor code. This smell refers to a class that has too many responsibilities and is doing too much. Being that ideally, a class should have only one responsibility (Single Responsibility Principle). When a class becomes large and spans many lines of code, it may be an indicator that it is taking on too many tasks.

12.



Many different clients need to create a similar object. You would like to outsource this concrete instantiation to a dedicated class. Which technique will you use, in one word (lower case)?

Respuesta: flyweid

✕

La respuesta correcta es: factory

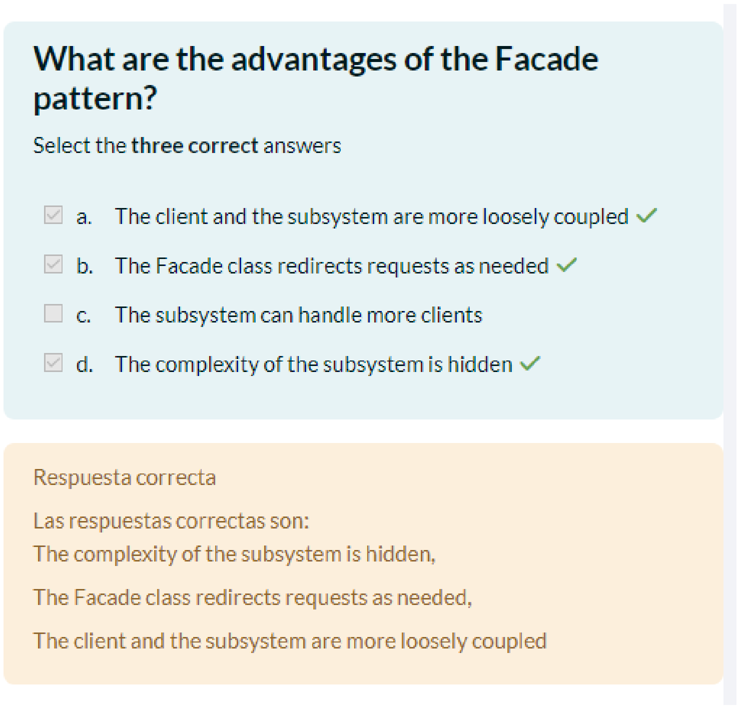
Justification:

The Factory pattern is suitable because it centralizes the creation of similar objects in a single class, simplifying client code and facilitating future changes to the creation logic without modifying the client code. Additionally, it improves code reusability and maintainability.

Error:

Again I couldn't read the question correctly, I thought it referred to reusing common parts.

13.



What are the advantages of the Facade pattern?

Select the **three correct** answers

- [✓ a. The client and the subsystem are more loosely coupled ✓]
- [✓ b. The Facade class redirects requests as needed ✓]
- c. The subsystem can handle more clients
- [✓ d. The complexity of the subsystem is hidden ✓]

Respuesta correcta

Las respuestas correctas son:
The complexity of the subsystem is hidden,

The Facade class redirects requests as needed,

The client and the subsystem are more loosely coupled

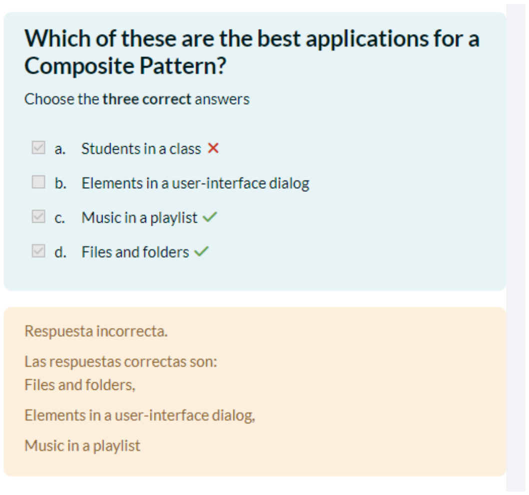
Justification:

The client and the subsystem are more loosely coupled: The Facade pattern reduces the direct dependency between the client and the subsystem, making maintenance easier.

The complexity of the subsystem is hidden: Hides the complexity of the subsystem, presenting a simple interface to the client.

The Facade class redirects requests as needed: Redirects client requests to the correct subsystem components, simplifying interaction.

14.


Which of these are the best applications for a Composite Pattern?

Choose the **three correct** answers

- [✓] a. Students in a class ✗
- [ ] b. Elements in a user-interface dialog
- [✓] c. Music in a playlist ✓
- [✓] d. Files and folders ✓

Respuesta incorrecta.

Las respuestas correctas son:
Files and folders,

Elements in a user-interface dialog,

Music in a playlist

Justification:

Elements in a user-interface dialog: The Composite pattern allows user interface elements to be treated uniformly, whether simple or composite, facilitating the management and manipulation of hierarchical structures.

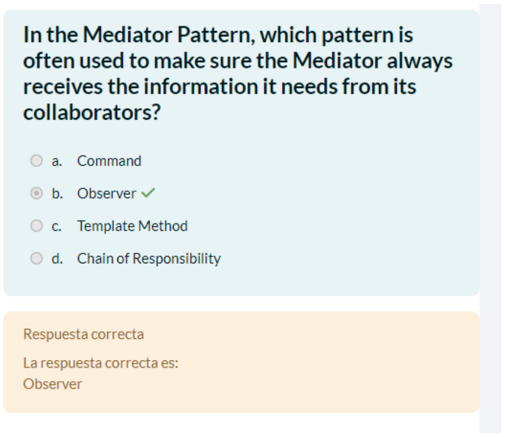
Music in a playlist: The Composite pattern is ideal for playlists, as it allows you to manage both individual songs and groups of songs (playlists) in the same way, simplifying organization and playback.

Files and folders: The Composite pattern is perfect for file systems, as it allows files and folders (which can contain other files and folders) to be treated uniformly, facilitating management operations.

Error:

I thought that because the music list was one of the correct ones, so would the student list, and I didn't want to risk it because of the interface because it was difficult for me to visualize it.

15.


In the Mediator Pattern, which pattern is often used to make sure the Mediator always receives the information it needs from its collaborators?

- a. Command
- [✓ b. Observer] ✓
- c. Template Method
- d. Chain of Responsibility

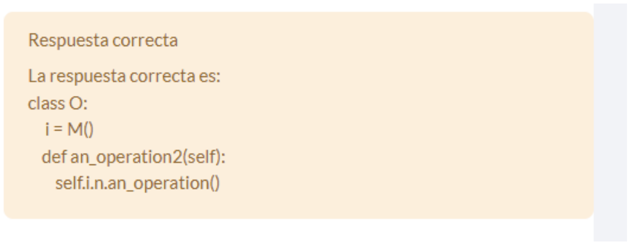
Respuesta correcta

La respuesta correcta es:
Observer

Justification:

It is a Mediator because the Observer pattern is often used to ensure that the Mediator always receives the necessary information from his collaborators. The Observer pattern allows objects (collaborators) to automatically notify the Mediator of any state changes, ensuring that the Mediator is always aware of relevant updates without requiring constant querying. This facilitates centralized communication and coordination without direct coupling between collaborators.

16.


Respuesta correcta

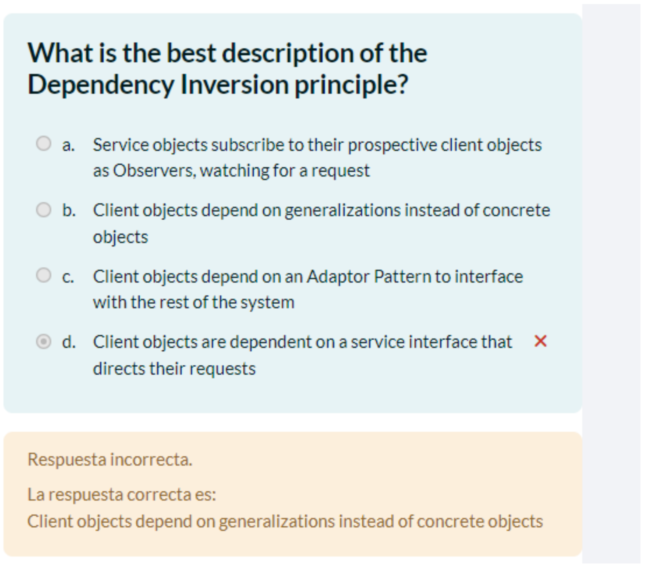
La respuesta correcta es:

```
class O:
  i = M()
  def an_operation2(self):
    self.i.n.an_operation()
```

Justification:

a was the correct option because it violates the Principle of Least Knowledge or Law of Demeter, which states that an object should interact only with its immediate friends and not with internal objects of other objects.

17.



What is the best description of the Dependency Inversion principle?

a. Service objects subscribe to their prospective client objects as Observers, watching for a request

b. Client objects depend on generalizations instead of concrete objects

c. Client objects depend on an Adaptor Pattern to interface with the rest of the system

[✓ d. Client objects are dependent on a service interface that directs their requests] ✗

Respuesta incorrecta.

La respuesta correcta es:
Client objects depend on generalizations instead of concrete objects

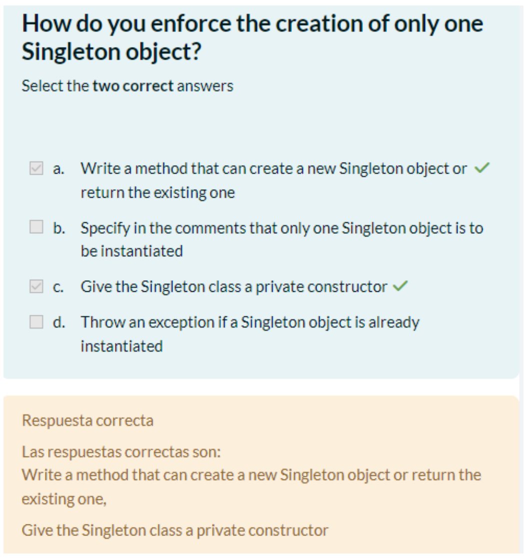
Justification:

The Dependency Inversion Principle states that high-level modules should not depend on low-level modules, but rather both should depend on abstractions (for example, interfaces or abstract classes). This principle also implies that abstractions should not depend on details, but rather that details should depend on abstractions.

Error:

In this one I didn't quite understand what the answers mean xd

18.



How do you enforce the creation of only one Singleton object?

Select the **two correct** answers

[✓ a. Write a method that can create a new Singleton object or return the existing one] ✓

b. Specify in the comments that only one Singleton object is to be instantiated

[✓ c. Give the Singleton class a private constructor] ✓

d. Throw an exception if a Singleton object is already instantiated

Respuesta correcta

Las respuestas correctas son:
Write a method that can create a new Singleton object or return the existing one,

Give the Singleton class a private constructor

Justification:

Write a method that can create a new Singleton object or return the existing one: A static method that handles the creation of the singleton instance.

Give the Singleton class a private constructor: By making the constructor private, you avoid creating instances outside the class.