Brad Bensch

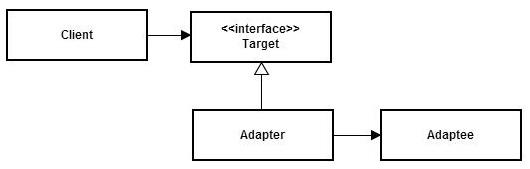
Kevin Mulligan

Jenny Zhen

William Spaw

Chad Koppes

1. **The C++ implementation of class Adapter uses multiple inheritance. Using a diagram show how this could be implemented in Java. Be specific about the Java features that you are using, and the association between participants in the pattern and classes in your answer.**

  
Chad - By using an interface the client can call known methods according to it’s implementation without knowing anything about how the adapter is making changes to interact with the adaptee. The adapter implements the Target interface and then, for each method that the client may call, does the necessary transformations and method calls to the Adaptee class doing the work. The benefit here is that the Client need not be changed if the functionality in the back end needs to change. Other benefits are that multiple Adapters can be implemented from the Target and the Adapter can use subclasses of Adaptee if needed.

1. **Using examples not found in the text, myCourses, or discussed in class, describe the distinctions between the intentions of the Proxy and Adapter patterns. A simple statement of the intentions of both patterns will not receive high marks.**

Chad - The biggest difference is that the Adapter pattern translates requests from a client to an Adaptee as opposed to the Proxy pattern which creates a stand-in object to simplify or speed-up operations when, for example, resources are limited. For example, if a vending machine operation wanted to be able to remotely monitor the data of its vending machines (inventory, coinage, etc.) it would need to use something like Java RMI to be able to access data from their server heap from wherever their remote device might be. In order to do that efficiently, a Proxy pattern could be used. The client might would make a request *as if* it is talking to the service object, but in reality it is making a request on a proxy RMI object which would then package the request, send it to the server side where a helper object would unpackage the request and call methods directly on the server object. The server would do the work, then return an answer in the same manner - packaging and sending it through proxy objects until it gets to the client. Although this proxying process has similarities to Adapter, the intent is different.

1. **Consider the Builder pattern. A new part is now able to be built into your product. This new part will be handled by all new ConcreteBuilders added to the system. What must change to accommodate the new parts? How can you let the current ConcreteBuilders ignore the part without rewriting any of those ConcreteBuilders? If an older ConcreteBuilder is in use and the Director requests for a new part type not known by the old builder to be put in the product what should happen?**

The ConcreteBuilders would have to change to build the different parts, and the interface would have to reflect the conditional changes in the product. If the current ones would ignore the specific part, you wouldn’t update the builder interface to tell them to make the new part. If the director requests a part that the ConcreteBuilder doesn’t know about, it doesn’t get it because the builder cannot build it.

1. **Argue against the statement "Coordinating behavior should be defined in the objects being coordinated." How does the Mediator pattern fit into this argument?**

Kevin - While in some cases the statement can be used to efficiently write software, in any application that needs to be scalable having each object define how it coordinates with every other object can be both tedious and inefficient. As an example, imagine a play being performed in a theater. If the audience were required to coordinate with themselves to find seating, buy tickets, ensure that only as many tickets exist as seats, and so on, the result would be an absolute mess. Instead, there are ushers, ticket sellers, and other organizers, or mediators, between people to ensure that everything runs smoothly. The mediator pattern fits into this argument in that it is a way of designing a system such that objects communicate with one mediator object, which coordinates communication between all of the other objects. This allows the system to be more efficient, and handle communication between objects without every object needing to know every other object.

1. **"The Mediator pattern trades-off reduction of coupling among the objects it coordinates by lowering cohesion in the Mediator itself." Argue for or against this statement, using information drawn from the description of the Mediator pattern itself.**

Chad - The Mediator pattern is very helpful in decoupling related objects in order to simplify complex inter-relationships between them. Instead of a complex web of object relationships all of the related objects interact with a mediator object that facilitates the necessary interactions between the objects. This increases the reusability of the objects and the ease in which the objects can be modified since the only thing that needs to be accounted for are the steady needs of the mediator object. It also allows for adjustments in communication between the objects to be accomplished in one place: the mediator. The problem is that as the number of related objects and types of communications increases, the mediator class becomes larger and more complex itself. This could reduce cohesion in the mediator class to an extent where it becomes difficult to manage.

1. **What does it mean to say a Memento object is opaque to its caretaker? What implications does this have on the caretaker's use of a memento?**  
     
   Jenny - When a memento object is said to be opaque to its caretaker, it means that the caretaker does not know the state of the memento. All of the information contained within the memento is hidden/not visible to the caretaker. This provides encapsulation for the data, and it restricts what the caretaker can do for the memento, as a result. The implications include the inability for the caretaker to modify the memento. The caretaker can still pass the data back to the originator when it needs to restore a previous state.
2. **How might the Memento and Command patterns be used in combination to provide undo functionality in an editor system?**

Kevin - Properly linked together, the two patterns can provide an undo by saving a memento each time a command is executed. In a text editor, for example, each time the cut or paste command is executed, a memento can be saved which contains the state the editor was in before the operation. If these mementos are saved on a stack structure, hitting an undo button can simply take the memento from the top of the stack and set the state of the editor to the state saved in the memento.

1. **Explain how Java's layout manager is an example of the Strategy class. In this example, what Java class or interface plays the role of each of the participants in the pattern?**  
     
   Jenny - Java’s LayoutManager is an interface within Java’s AWT library. It is an example of the Strategy pattern because the LayoutManager provides different ways to arrange the data while still allowing them to be used interchangeably. Each layout has its own tradeoffs depending on how each strategy has to handle the data.The participant associated with the LayoutManager is the strategy. The classes that use the LayoutManager interface are the concrete strategies, and those include BorderLayout, BoxLayout, CardLayout, FlowLayout, GridbagLayout, GridLayout, etc. The container would be the context.
2. **The Strategy object may inform the Context of the results of its work by either:**
   1. **Returning an object of an appropriate Result class to the Context, or**
   2. **Use a reference to the Context object and calling Context object methods to record the results.**

**What are the relative advantages and disadvantages of each of these approaches? Consider issues such as encapsulation, coupling, and cohesion in forming your answer.**

The advantage of returning a Result object to the Context is more cohesion and less coupling than the alternative. The disadvantage is that the Result class used to encapsulate the data can make updating the code difficult, if a future Context requires data that the Result does not have. On the other hand, using a reference to the Context object allows updates to be done easily, but increase coupling and decrease cohesion of the design.

1. **Let methodA() and methodB() both be methods declared in the current class or one of its super classes, and assume that methodA() calls methodB(). Is methodA() always, sometimes, or never an example of the Template Method pattern? Justify your answer in terms of the pattern as presented in the text.**This is only an example of the Template Method pattern if methodA() is in one of the superclasses and methodB() is in the current class. All other configurations do not line up with the Template Method pattern requirement of a method being altered by a method in a subclass.
2. **Explain the "Hollywood Principle" in Template Methods. Why is access claimed to be inverted? Use an example of the Template Method pattern not found in the text, myCourses, or discussed in class.**

Jenny - The “Hollywood Principle” is known to be the saying “don’t call us, we’ll call you.” In Template Methods, the parent classes are the ones who do the calling for operations on its subclasses. As a result, the subclasses do not need to call the parents directly. The access is inverted because, usually, the subclasses class the parents using the keyword, “super,” whenever the subclass needs to use the implementation and functionality provided by the parent class by default. An example of the Template Method pattern is building a computer. The different components don’t need to look for the right motherboard. The motherboard says what components are needed, and calls for them.