Factory Method	How does this promote loosely coupled code? Factory methods offers a structure including decisions on which concrete object to use. If there are some changes in the constructor, we can break it within the class without changing other classes, since it is encapsulated in factory.
Proxy	If a Proxy is used to instantiate an object only when it is absolutely needed, does the Proxy simplify code? It does not simplify the code, since a lot of codes should be added to keep things like the record of requests which will increase the complexity.
.	.
Strategy	(i) What happens when a system has an explosion of strategy objects? Is there some better way to manage these strategies? It's difficult to manage codes when a system has an explosion of strategy objects. One better way is grouping some similar strategies, and each time we just use the strategy classes based on the similar classes. (ii) In the implementation section of this pattern, the authors describe two ways in which a strategy can get the information it needs to do its job. One way describes how a strategy object could get passed a reference to the context object, thereby giving it access to context data. But is it possible that the data required by the strategy will not be available from the context's interface? How could you remedy this potential problem? A member variable of that strategy object type is made in the context. When a strategy object needs to access to context data, the context just changes its member variable to the reference of that strategy object.

Decorator	In the Implementation section of the Decorator Pattern, the authors write: A decorator object's interface must conform to the interface of the component it decorates. Now consider an object A, that is decorated with an object B. Since object B "decorates" object A, object B shares an interface with object A. If some client is then passed an instance of this decorated object, and that method attempts to call a method in B that is not part of A's interface, does this mean that the object is no longer a Decorator, in the strict sense of the pattern? Furthermore, why is it important that a decorator object's interface conforms to the interface of the component it decorates? We can't say that it is no longer a Decorator in the strict sense of the pattern. The reason that decorator object's interface conforms to the interface is that its presence is transparent to the component's clients
Adapter	Would you ever create an Adapter that has the same interface as the object which it adapts? Would your Adapter then be a Proxy? Yes, though it is unnecessary, but it is pretty useful if there is some extensions. 2) No, though it looks similar but proxy is intended to have an access to the real object and may have some specific implementations.
Bridge	How does a Bridge differ from a Strategy and a Strategy's Context? Strategy is going to define a family of algorithms, encapsulate each one, and make the run-time decision, but bridge is able to separate an abstraction from its implementation.
Facade	(i) How complex must a sub-system be in order to justify using a facade? When there is a lot of interactions between two systems. A lot of in and out data transit, which can be combined in a Façade.

	(ii) What are the additional uses of a facade with respect to an organization of designers and developers with varying abilities? What are the political ramifications? Façade is like an interface, the developer could use it without knowing what is undergoing in this system. The ramifications would be a decoupling of the design from the implementation, which may make the system hard to maintain.
.	.
Composite	(i) How does the Composite pattern help to consolidate system-wide conditional logic? Considering a basic conditional logic combined with several branch conditions. The yes/no of the base condition relies on the branch conditions. (ii) Would you use the composite pattern if you did not have a part-whole hierarchy? In other words, if only a few objects have children and almost everything else in your collection is a leaf (a leaf that has no children), would you still use the composite pattern to model these objects? (ii) Yes, considering a situation that we will have some extensions to this system. Some children will be added depending on the development, in which case, a composite structure would be better.
Iterator	Consider a composite that contains loan objects. The loan object interface contains a method called "AmountOfLoan()", which returns the current market value of a loan. Given a

	requirement to extract all loans above, below or in between a certain amount, would you write or use an Iterator to do this? It's fine to use iterator, since it requires getting the amount of all loans and validate against the condition.
Template Method	The Template Method relies on inheritance. Would it be possible to get the same functionality of a Template Method, using object composition? What would some of the tradeoffs be? Composition is able to realize the same functionality. But template pattern has more control on the implementation when they have already done the things and sub-classes handle others. Object composition may also make the pattern harder to understand which increases difficulty to maintain it.
Abstract Factory	In the Implementation section of this pattern, the authors discuss the idea of defining extensible factories. Since an Abstract Factory is composed of Factory Methods, and each Factory Method has only one signature, does this mean that the Factory Method can only create an object in one way? No, Factories create objects which are determined by the requirements for object creation. It would be possible to pass a parameter to a factory method and return one of many sub classed objects based on that parameter. Consider the MazeFactory example. The MazeFactory contains a method called MakeRoom, which takes as a parameter one integer, representing a room number. What happens if you would also like to specify the room's color & size? Would this mean that you would need to create a new Factory Method for

	your MazeFactory, allowing you to pass in room number, color and size to a second MakeRoom method? You could create another function with the same name but different signature and the parameters are different and this would be defining another Factory method so, definitely yes, you would need a new Factory. Of course, nothing would prevent you from setting the color and size of the Room object after is has been instantiated, but this could also clutter your code, especially if you are creating and configuring many objects. How could you retain the MazeFactory and keep only one MakeRoom method but also accomodate different numbers of parameters used by MakeRoom to both create and configure Room objects? Just create a single MakeRoom that takes many parameters. Which differ in number and type.
Builder	Like the Abstract Factory pattern, the Builder pattern requires that you define an interface, which will be used by clients to create complex objects in pieces. In the MazeBuilder example, there are BuildMaze(), BuildRoom() and BuildDoor() methods, along with a GetMaze() method. How does the Builder pattern allow one to add new methods to the Builder's interface, without having to change each and every sub-class of the Builder? Since builder separates the constructor from its implementation, we could implement a function shared by all subclasses in the builder which inherited by the sub classes. So the modification in the sub-classes does not affect other sub classes.
Singleton	The Singleton pattern is often paired with the Abstract Factory pattern. What other creational or non-creational patterns would you use with the Singleton pattern?

	One possible way is using a mediator with a singleton to obtain a handler for the system of classes. Facade objects are usually Singletons since only one Facade object is required.
Mediator	Since a Mediator becomes a repository for logic, can the code that implements this logic begin to get overly complex, possible resembling speggheti code? How could this potential problem be solved? In the case that the sender and receiver have a very complex interaction, so yes, To fix this problem, we can couple it with another design pattern like façade or composition.
Observer	(i) The classic Model-View-Controller design is explained in Implementation note #8: Encapsulating complex update semantics. Would it ever make sense for an Observer (or View) to talk directly to the Subject (or Model)? (i). Talking directly to Subject is not a good design. Observer is here to keep View and Model independently. What are the properties of a system that uses the Objserver pattern extensively? How would you approach the task of debugging code in such a system? The property is this system has a very complex transmissions and relationships, Debugging here will use a lot of thoughts, we should guarantee that the Observer can get the information. If there is status change, the observer should receive correct messages. Observer should gives correct response to the subject. (iii) Is it clear to you how you would handle concurrency problems with is pattern? Consider an Unregister() message being sent to a subject, just before the subject sends a Notify() message to the ChangeManager (or Controller). A simple way is prohibiting the message as above sending. When a Notify() message is sent out, there is no permission to do

	Unregister(). Or we can use a queue buffer to buffering the unhandled messages, then process them based on the time stamp.
Chain of Responsibility	(i) How does the Chain of Responsibility pattern differ from the Decorator pattern or from a linked list?. The passing type is different, in the case of a chain, the message in the chain will response or not response and it could be passed to the next handler. But in decorator, each node in the list will have the responsibility for the object. (ii) Is it helpful to look at patterns from a structural perspective? In other words, if you see how a set of patterns are the same in terms of how they are programmed, does that help you to understand when to apply them to a design? Some patterns will be similar, and the way to use them are pretty flexible, so we can't use merely how they are programmed to understand when to apply.
Mememto	The authors write that the "Caretaker" participant never operates on or examines the contents of a memento. Can you consider a case where a Caretaker would in fact need to know the identity of a memento and thus need the ability to examine or query the contents of that memento? Would this break something in the pattern? The memento accessible to an object instead of the originator when caretaker are attempting to make a decision based on the history information and yes, it breaks the rule of memento.
Command	In the Motivation section of the Command pattern, an application's menu system is described. An application has a Menu, which in turn has MenuItems, which in turn execute commands when they are clicked. What happens if the

	command needs some information about the application in order to do its job? How would the command have access to such information such that new comamnds could easily be written that would also have access to the information they need? State pattern could be used to deal with it. The command can obtain information from MenuItem's current state and commands can execute based on the MenuItem's current state. .
Prototype	(i) When should this creational pattern be used over the other creational patterns? Creating an instance of a given class is either expensive or complicated. (ii) Explain the difference between deep vs. shallow copy. Deep copy does copy the actually data but shallow copy only copies the address or the pointer.
State	If something has only two to three states, is it overkill to use the State pattern? Answer: No. It is fine to use the State pattern when the transitions between the states are complicated. Also the states will become increasing if the system keeps expanding. Thus applying this pattern also leave space for future improvement.
Visitor	One issue with the Visitor pattern involces cyclicality. When you add a new Visitor, you must make changes to existing code. How would you work around this possible problem?

	We set visitors should inherit from a basic visitor to break the cyclicality and leverage the code reuse. Then no changes needed every time a new visitor is created.
Flyweight	(i) What is a non-GUI example of a flyweight? Considering a public switched telephone network, some resources like dialogue generator, dialogue charger, dialogue receiver are shared by customers. When they call, they don't know the usage of resources, for the users, all they should do is make the call and go through the call. (ii) What is the minimum configuration for using flyweight? Do you need to be working with thousands of objects, hundreds, tens? The minimum configuration relies on the cost to create the object but not depends on the number of objects.
Interpreter	As the note says in Known Uses, Interpreter is most often used "in compilers implemented in object-oriented languages...". What are other uses of Interpreter and how do they differ from simply reading in a stream of data and creating some structure to represent that data? They use a parsing style not the reading in. The difference is that interpreter are trying to translate the data in one format with one specific meaning to another format means another thing.