**Adapter and Proxy:**

* intent of Adapter: convert an interface of a class to another interface client expected and it let incompatible classes work together;
* Explain how adapter work:

Adapter let the incompatible classes work together, and sometimes we cannot change the source code of the library in this case adapter pattern can solve this problem. Also we use Adapter when we want to use an existing class that perform similar functions but with different interface.

Structure: client -> target implemented by Adapter -> adaptee

* ***False***: The reason we cannot change the interface of the adaptee is because we have no access to its source code.
* Intent of Proxy: Provide surrogate or placeholder for another object control access to it.
* Compare Adapter and Proxy:

Similarity:

* They both are structural pattern.
* They provide interface that client expected.
* Rely on object composition to implement it.

Difference:

* In proxy all classes has to implement the same interface while Adapter and Adaptee has difference interface.
* Difference intent.
* Difference structure.

**Template Methods and Prototype:**

* Intent of Template Method: Define the skeleton of algorithms of an operation and differing some steps to subclass without changing the algorithms structure.
* Intent of Prototype: specify the kind of object from prototypical instance and create new object by copy from this prototype
* Some consequence of Prototype:
  + It hides the concrete product class from client.
  + All client create class without programming.
  + Let client work with application specific class without modification.

**Factory Method and Abstract Factory:**

* Intend of Factory Method: define the interface for creating object by let subclass decide which class to instantiate.
* The main benefit of using Factory Method compared with constructor:
  + Decoupling
  + Hiding completely of created object.
  + Comparing to constructor client need to access all the concrete classes to create object but using factory method client just need to know the method that we can create object.
* Intent of Abstract Factory: Provide an interface for creating families of related or dependent object without specifying their concrete class.
* Compare Factory Method and Abstract Factorr:
  + Similarity:
    - Both are creational pattern
    - Both decoupling
    - Both remove if/else
  + Difference:
    - Factory Method creates one product.
    - Abstract Factory creates many product type.
* ***True:*** With the abstract Factory pattern, no matter how many concrete products are to be created, the client is always decoupled from the concrete types.
* ***False:*** With the Abstract Factory pattern, it is easier to add new product types because Abstract Factory support extension, not modification.
* ***True:*** Factory Method helps us remove if/else statements from client code.

**CoR and Command:**

* Intend of CoR: Avoid coupling the sender of a request to its receiving by allow more than one object a chance to handle the request. Chain the receiving object and pass the request along the chain until an object can handle it.
* Intend of Command: Encapsulation a request as object thereby letting you parameterize clients with different request, queue, or log request and support undoable operation.
* “callback” method in command patterns are “execute” “undo”.

**Iterator:**

* Intend of Iterator: Provide a way to access the element of an aggregation object sequentially without exposing it underlying representation.
* When to use Iterator pattern:
  + To support traversals of aggregate objects without exposing their internal representation.
  + To provide thread-safe, concurrent traversals of aggregate objects.
  + To provide a uniform interface for traversing different aggregate structure to support polymorphic iteration
  + To allow different traversal methods depending on what the client needs.

**State and Strategy:**

* Intend of State: Allow an object to alter it behavior when its internal state change. The object will appear to change it class.
* Intent of Strategy: Define a family of algorithms, encapsulate each one and make them interchangeable and it let algorithms vary independently form client that use it.
* Compare State and Strategy:
  + Similarity:
    - Both help to remove if/else statement from code
    - Both allow runtime change of behavior
    - Both are behavioral pattern.
  + Difference:
    - Intent is different.
    - Strategy is stateless while State is stateful.
    - State: Behavior change is caused by internal state change while Strategy’s behavior state change is caused by external/environment change.
* When will we consider using State pattern:
  + When an object’s behavior depends on its state, and it must change its behavior at run-time depending on that state.
  + When we want to model vending machines or implement computer video games.
  + When you want to treat each state as an object that can vary independently from others.
  + When you want to localize state-specific behavior in each concrete state.
  + When you want to break a multi-conditional statement and move related conditional branches into individual State classes.
* When will we consider using Strategy patter:
  + When you want the client to decouple from a strategy it uses to perform a function
  + When you want to make the change of strategy easier at runtime
  + When you need to provide run-time change of behavior for an application
  + When you want your application to use different strategies for different situations during runtime
  + When you want to break a multi-conditional statement and move related conditional branches into their own class.

**Mediator and Observer:**

* Intend of Mediator: Define an object of encapsulate how a set of objects interact. It promote loose coupling by keeping objects from referring to each other explicitly.
* Intent of Observer: define one-many dependency between objects so that when one object changes state all its dependents are notified and update automatically.
* What is the advantages and disadvantages of Mediator:
  + Advantages:
    - Each subject no need to know each other
    - Decoupling
    - Centralize complex components.
  + DisAdavantages:
    - Make application hard to maintain
    - Centralization since any code change effect to others.
    - Monolith
* Explain the motivation of the observer pattern:
  + Model one-many publisher and subscriber relationship application
  + Multiple subscriber are dependent of the state of publisher
  + Reuse Publisher and Subscriber their relation must be decouple.

**All others:**

* ***False:*** In strategy pattern, strategy objects are stateful
* ***True:*** Strategy pattern helps us eliminate condition statements from client by encapsulating the behavior in separate strategy classes.
* ***False:*** Mediator Centralize control which can make the mediator itself a monolith that’s easy to maintain.
* ***True:*** Observer pattern support layering of software application.
* ***False:*** Visitor makes adding new operations hard but adding a new concrete elements class easy.
* ***True:*** the MVC pattern decouples among the model, view, and controller.
* ***False:*** in the state pattern, you usually set the state only once.
* ***False:***  in strategy pattern, the strategy is independent of the surrounding context.
* ***False:*** in the strategy pattern, if a strategy does not work, work is a fallback to the next strategy (if any)
* (***Visitor***)Explain double dispatch: is a mechanism that dispatches two function calls depending on the real type of the references on which the methods are called.
* When will you consider using the visitor pattern:
* What is decoupled from what in each of the following pattern:
  + ***FM:​*** client – subclass.
  + ***AF:*** client – subclass
  + ***COR:*** sender of request form its receiver
  + ***Command:*** object invoke from the one know how to perform it.
  + ***Iterator:*** client - elements
  + ***Mediator:*** colleagues
  + ***Observer:*** object and subscribers (Publisher / Subscriber)
  + ***Visitor:*** externalize operation
* Which pattern help to remove if/else statements:
  + Strategy, State, Factory Method, Abstract Factory, CoR, command
* Three pattern that help us avoid a lot of subclassing:
  + ***Mediator, Decorator, Composit***
* What we have learnt from Gof pattern so far:
  + ***Decoupling:*** Separation of concern
  + ***Reuse:*** use existing api
  + ***Simplification:***
  + ***Trade off:***
  + ***Enforce business rules:***