1. Decorator, strategy and visitor.

1. Similarities

They can change the properties of an object or change the behavior of an object.

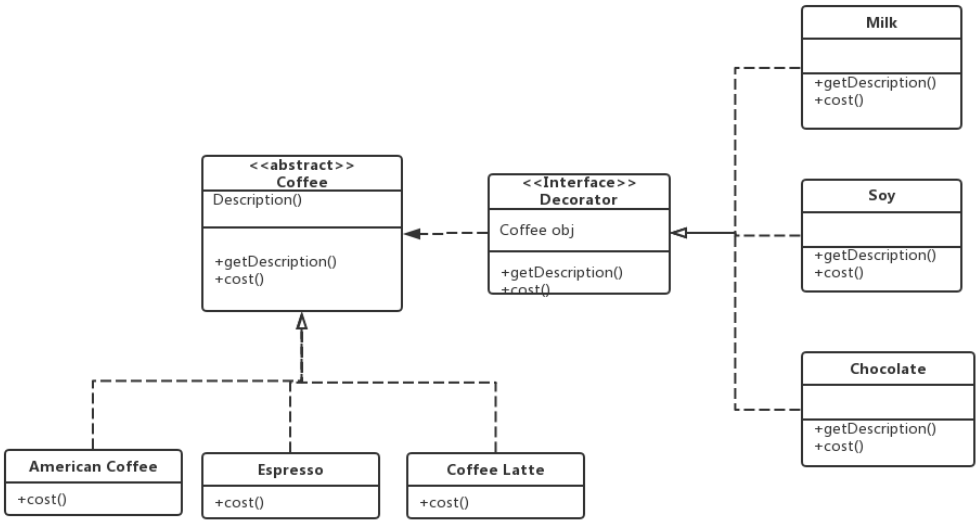
2. Difference

Decorator	Strategy	Visitor
A Decorator could add new functionality to an existing object without changing its structure.	The behavior of a class or its algorithm can be changed at runtime.	Using a visitor class that changed the execution algorithm of the element class.

a) Decorator dynamically add some additional responsibilities to an object. In terms of adding functionality, the decorator mode is more flexible than generating subclasses. Versus to the Strategy, The decorator pattern is generally used where it is necessary to extend the functionality, and each decor is an extension or enhancement. Strategy pattern tends to encapsulate the implementation method or algorithm. Versus to the Visitor, decorator pattern is to extend and enhance the behavior of an object, and does not change the original algorithm. Visitor changed the algorithm of the object through an visitor class.
b) Strategy define a series of algorithms, wrap them one by one, and make them interchangeable. Versus to the Visitor, strategy pattern allow different algorithms to apply to the same data structure, let the client select the algorithm. Visitor pattern use the expert pattern allow each concrete visitor store information that is needed to fulfil the functionality. Also, those concrete visitor operations are irrelevant rather than strategy.
c) Visitor performing different unrelated operations on an object through an access class.

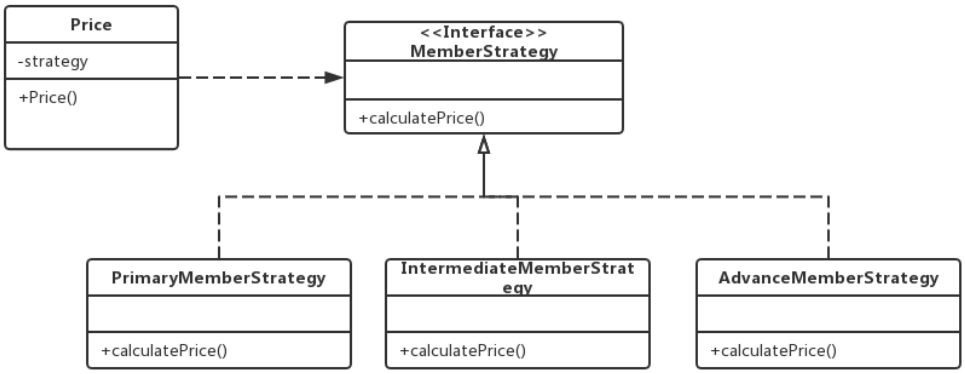
3. Exclusively Applying Situation

a) Decorator



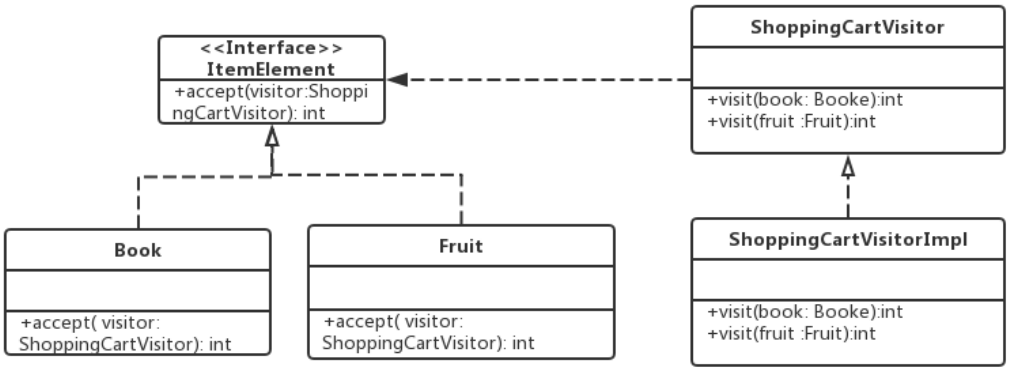
We can use decorator pattern to implement a coffee ordering system, use can choose a coffee with or without milk or soy and etc., Finally calculate the costs. In this case, decorator is used to extends the type of order. Is no reason can use the strategy, cause we focus on the calculate the different coffee costs. Also visitor doesn't work, visitor is used in doing a different thing on an object.

b) Strategy



When designing website logic, we can use the strategy mode to give different users different discount prices, for example, if this user is a "Advance" member, than 25% off. In this case strategy help us to manager the discount of different kind of the member, we do not need to enhance the "operation of the discount" , and we just focus on the discount of the member's type. Thus, decorator and visitor can not applied in this case.

c) Visitor



Add a shopping cart of different types of merchandise, when clicked on the settlement, it calculates the cost of all the different merchandise. Now, the calculation logic is to calculate the price of these different types of goods. Or by the visitor pattern we transfer this logic to another class. In this case, we don't need to enhance or extends this operation, decorator doesn't work, we use the same algorithm to calculate the payment, so strategy cannot applied yet.

2. Abstract factory, builder and factory method.

1. Similarities

They all provide object creation and management responsibilities.

2. Difference

Abstract factory	Builder	Factory
Provides an uniform interface for the client to creating products of different families	Separating a complex build from its representation, it make the same build process to create different representations	Deal with the problem of creating objects without having to specify the exact class of the object that will be created.

a) **Abstract factory** is more focus on produce the product family, targeting multiple product hierarchy. Versus to the Builder, Abstract factory is more concerned with the creation of similar products of different brands. Versus to the Factory, the factory realization of a common factory is a product. The factory realization of an abstract factory is multiple product families.

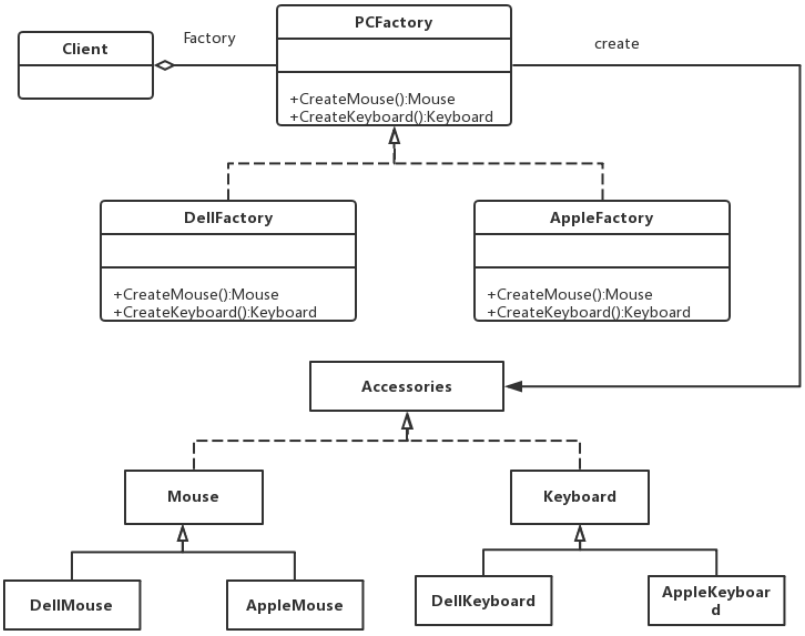
b) **Builder** mainly solved in software systems, sometimes faced with the creation of "a complex object", which is usually composed of sub-objects of various parts with certain

components, due to changes in requirements, the various parts of this complex object are often faced with dynamic change, but the main part combined by components is relatively stable. Versus to the Abstract Factory, builder is the extension of the abstract factory, builder can use abstract factory to vary the parts used by the build steps. Versus to the Factory, builder focuses on the process of component construction, which aims to create a complex object by step-by-step precision construction.

c) **Factory** targeted by a product hierarchy, If the product is single, the most suitable factory pattern. Versus to the Abstract Factory, The key to abstract factory is in the abstract relationship between products, so at least two products are required. The factory method is to generate products, not to pay attention to the relationship between products, so only one product can be generated. Versus to the Builder, builder can change the construction process, but factory cannot, because of template method.

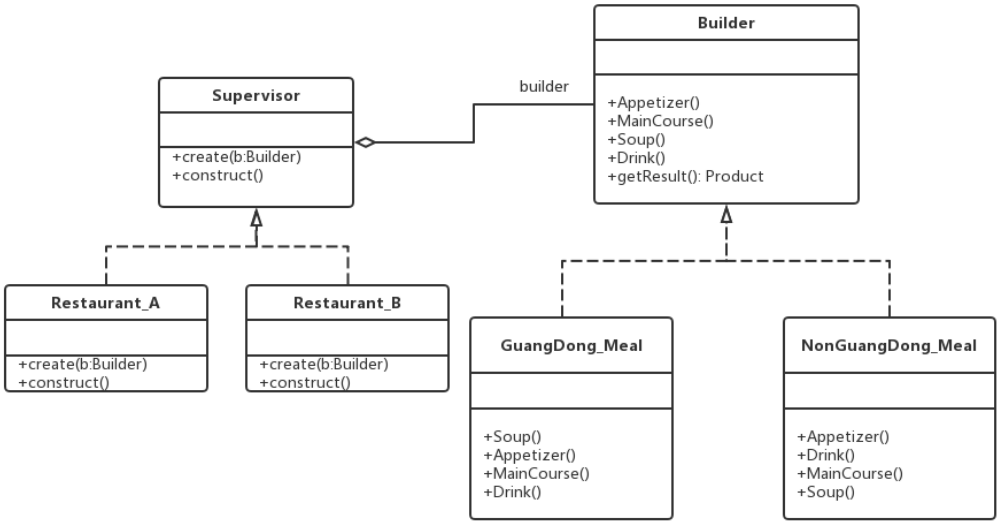
3. Exclusively Applying Situation

a) Abstract factory



This picture is talking about using Abstract Factory pattern to create the personal computer accessories model. The reason why I using “Abstract Factory” is there are many product need to create in a “PC” family. Factory pattern can just create a mouse or a keyboard, but not both. Builder pattern focus in create a complex object or a product rather than a family products.

b) Builder



In this case, we can use builder pattern change the restaurant “A” or “B”, also we can order like GuangDong’s culture(GuangDong one of a Province of China, “canton”, one of the diet culture in GuangDong is Soup first, main course later.) or just like other people appetizer first. Here we cannot apply factory/abstract factory pattern, because the way they create object is template, they cannot change the process steps, but builder can.

c) Factory When we designing a framework for connecting servers requires three protocols, "POP3", "IMAP", and "HTTP". These three can be used as product classes to implement an interface in using factory pattern. Because we just need one framework for connecting, there is no reason use abstract factory, neither builder.

3. Bridge, command and object adapter.

1. Similarities

Decoupling the system, hiding the information in the interface class.

2. Difference

Bridge	Command	Adapter
Separate the abstract part from the implementation part so that they can all change independently.	Encapsulate a request into an object, allowing you to parameterize the client with different requests.	Convert the interface of one class to another interface that the client wants. The adapter mode allows those classes that would otherwise not work together due to incompatible interfaces to

		work together.

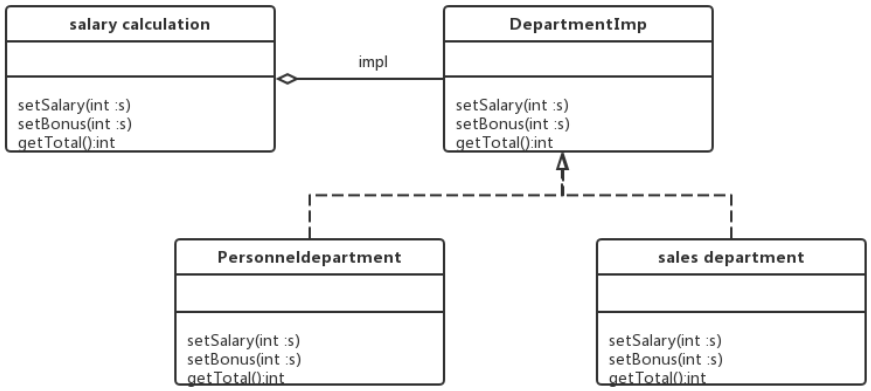
a) **Bridge** versus to command pattern, bridge allow the implement has the same function, command pattern implements is difference. Versus to Adapter, bridge focus on separate abstract and implements, make them independently. Adapter change the existing two interfaces to make them compatible.

b) **Command** we can encapsulate our request to decoupling the client from the command object. Versus to Adapter, command defines a uniform interfaces for different operations. But Adapter convert a interface to another.

c) **Adapter** convert the interface to another allows classes with incompatible interfaces to work together

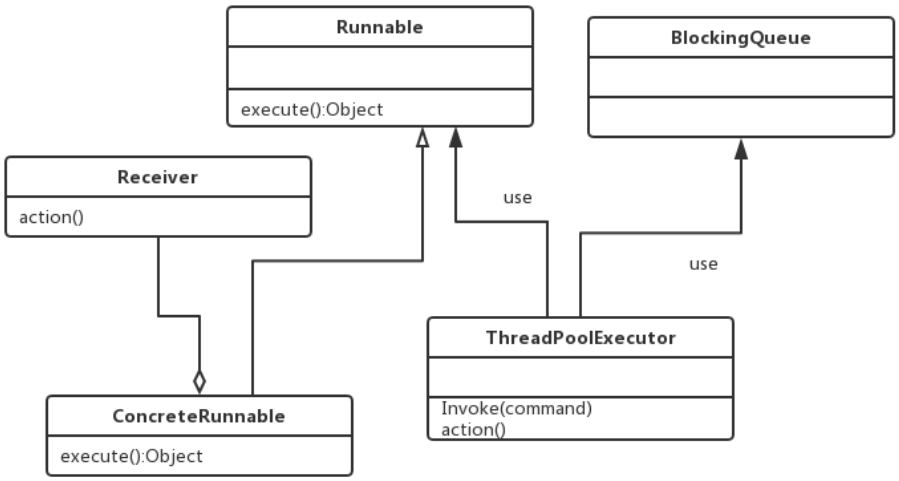
3. Exclusively Applying Situation

a) **Bridge**



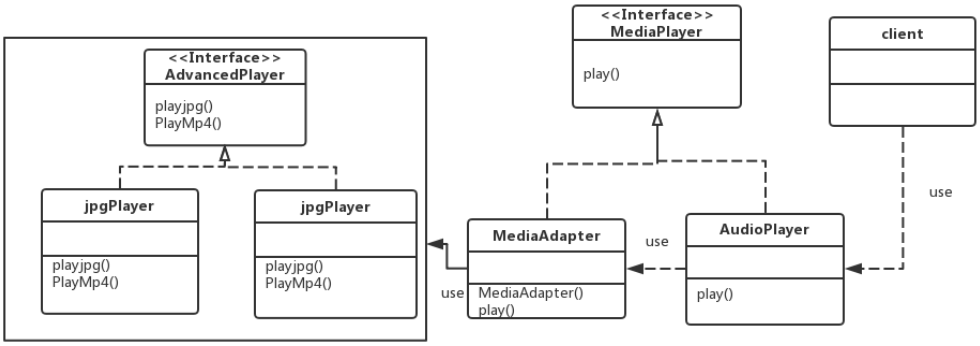
The bridging pattern separates the salary calculation methods of each department from the salary calculation class, so that it is convenient to add new parts in DepartmentImp, and we can also add a new calculation class that inherits the salary calculation. Here we do not use the Command and Adapter pattern because we don't need to convert any interface or using request encapsulation to handle with.

b) **Command**



In the ThreadPoolExecutor, it use command pattern decouples client code from Thread. Here Runnable is encapsulated by a command, ThrePoolExecutor is a invoker, ConcreteRunnable is a specify command implements. In this case, we cannot use bridge, because every command has different method, they do different thing. we also don't need to convert any interface or using request encapsulation to handle with.

d) Adapter



AudioPlayer can play audio files in mp3 format.We also have another interface, AdvancedMediaPlayer, and an entity class that implements the AdvancedMediaPlayer interface. By using Adapter that this class can play files in jpg and mp4 format. Here we use adapter just change the existing two interfaces to make them compatible. So is exclusively, the other two patterns doesn't work on here.

4. Singleton, flyweight and prototype.

1. Similarities

Optimization of the creation of objects for frequent use.

2. Difference

Singleton	Flyweight	Prototype
Create at most one, or a limit number of globally accessible instances of a class	Create a numerous instances of an object without server memory and performance	Create a behaviorally identical object repeated, reduce number of class, or avoid the construction cost

a) Singleton pattern involves a single class that is responsible for creating one objects while ensuring that only a single object is created. Versus to Flyweight, singleton just create one and the only one object. Flyweight create a numerous object to the system. Versus to the Prototype, singleton privatize the constructor, and provide a public method to access the object. Prototype pattern is based on the specified objects, and then new objects are created by copying these prototype objects.

b) Flyweight mainly used to reduce the number of created objects to reduce memory footprint and improve performance. Also, most of the state of an object can be externalized, and these external states can be passed into the object. Versus to the Prototype pattern, flyweight pattern reduce the creation of the object, there are the same business request, directly return the existing objects in memory, avoid re-creation. Prototype reduce the number of classes, use objects instead of classes.

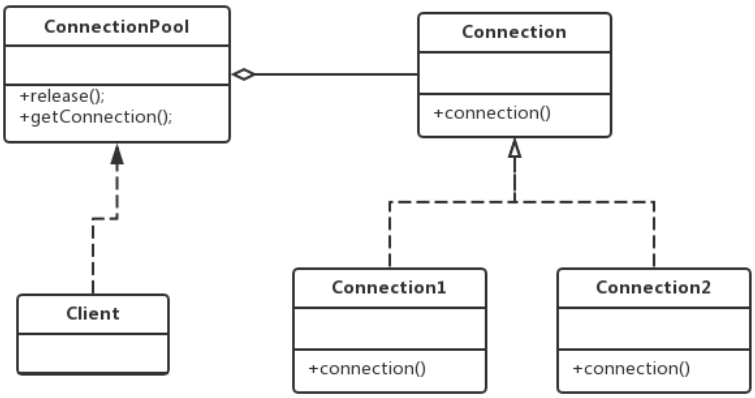
c) Prototype use prototype instances to specify the kinds of objects you create, and create new ones by copying them. What more, class initialization requires a large number of resources, including data, hardware resources, etc., to avoid these consumption through prototype copying.

3. Exclusively Applying Situation

a) Singleton

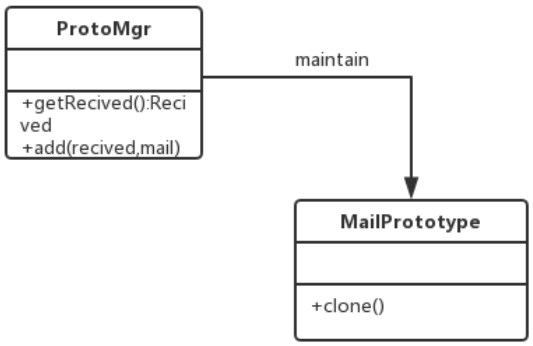
The website counter is implemented in singleton. If there are multiple counters, each user's access will refresh the counter value, so that the real count value is difficult to synchronize. However, use the singleton pattern implementation, there will be no such problem, and also provides thread safety. It is clear that the other two pattern doesn't work on this case.

b) Flyweight



The database connection pool has url, driverClassName, username, password and dbname. Those attributes are the same for each connection, so it is suitable to use the flyweight mode to process. Create a factory class, the above similar properties as internal data, the other as external data, when the method is called, as parameters are passed in, which saves space and reduces the number of instances. Here we could create more than one object, singleton is not suit for this case. Besides here we focus on reduce the object creation, prototype means copy than new, it doesn't work in this one.

c) Prototype



In the banking system, it is necessary to send electronic bills to all users on a regular basis every month. In that case, a large number of mail objects are needed. Through the prototype mode, you can quickly clone objects and add recipients. Reduce the consumption of creating new objects. Because each object will have a different attribute value, singleton doesn't work. What's more, we apply the prototype pattern using the clone method just pass the index , is more efficiency than flyweight create the object.