Software Design Patterns

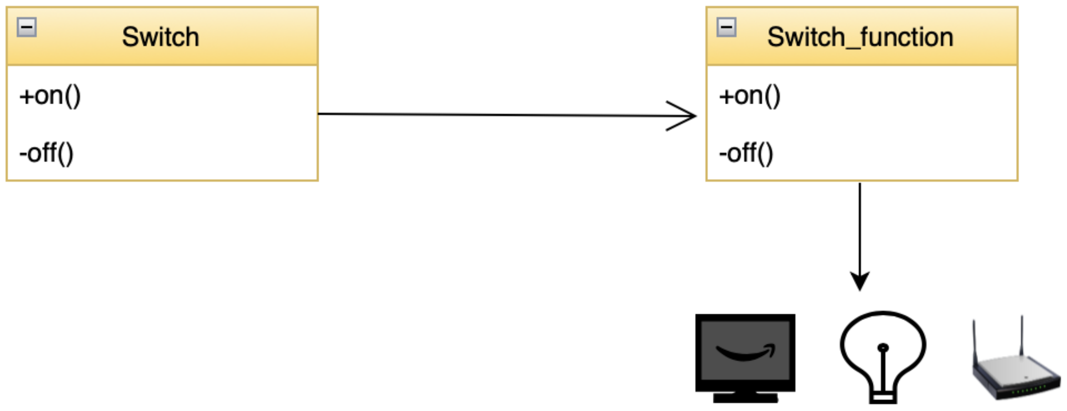
1. Bridge, Command and Object adapter Pattern

Bridge Pattern:

By decoupling an abstraction from its implementation so that the two can vary independently, as stated by the Gang of Four, the bridge pattern is a design pattern used in software engineering. [1] The bridge can employ inheritance, aggregation, and encapsulation to divide tasks into various classes.

Example: A household switch controlling lights, ceiling fans, etc. is an example of the Bridge. The purpose of the switch is to turn a device on or off.

How and why the pattern is applied: The Bridge can be seen in a switch that controls lights, ceiling fans, etc. in a home. The switch's function is to switch a gadget on or off. A pull chain, a straightforward two-position switch, or a variety of dimmer switches can be used as the switch itself. By separating an abstraction from its implementation, the Bridge pattern allows for independent change in each.

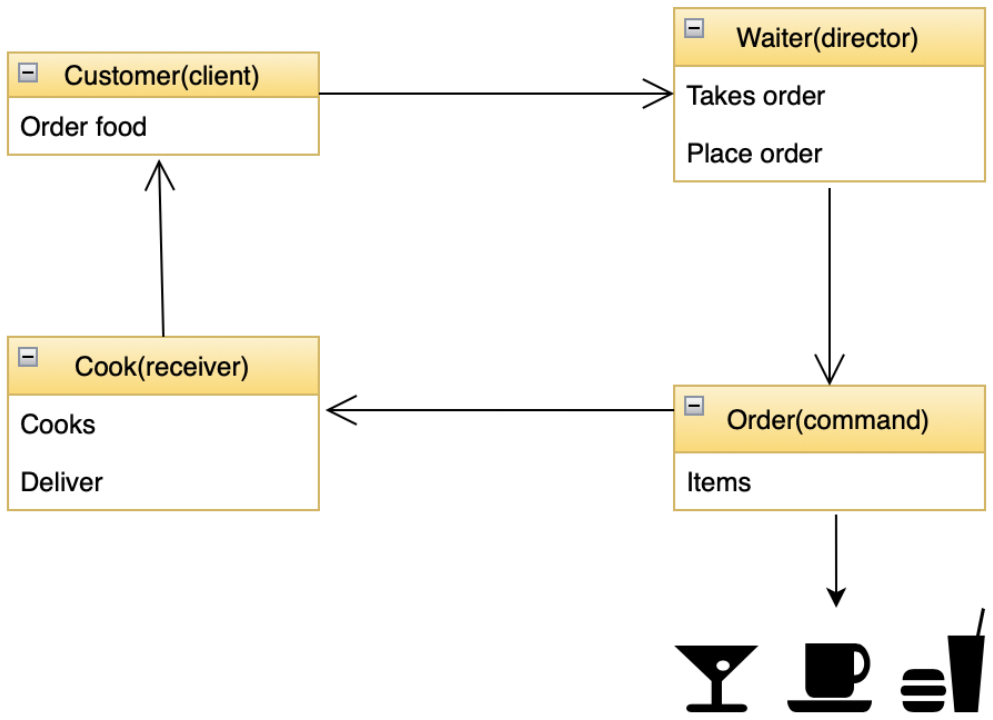


Command Pattern:

The command pattern is a behavioral design pattern used in object-oriented programming that involves using an object to hold all the data necessary to carry out an action or start an event later. The name of the method, the object that owns the method, and the values for the method arguments are all included in this data.

Example: Ordering the food in restaurant can be the example of the command pattern.

How and why the pattern is applied: By allowing requests to be contained into objects, the Command pattern enables clients to be parametrized with various requests. A Command pattern is an example of the "check" at a diner. The waiter or waitress takes a customer's request or command and records it by writing it on the check. Then a short order cook is put in line to take the order. Keep in mind that each waiter uses a pad of "checks" that is independent of the menu and can therefore accept requests to cook a wide variety of products.

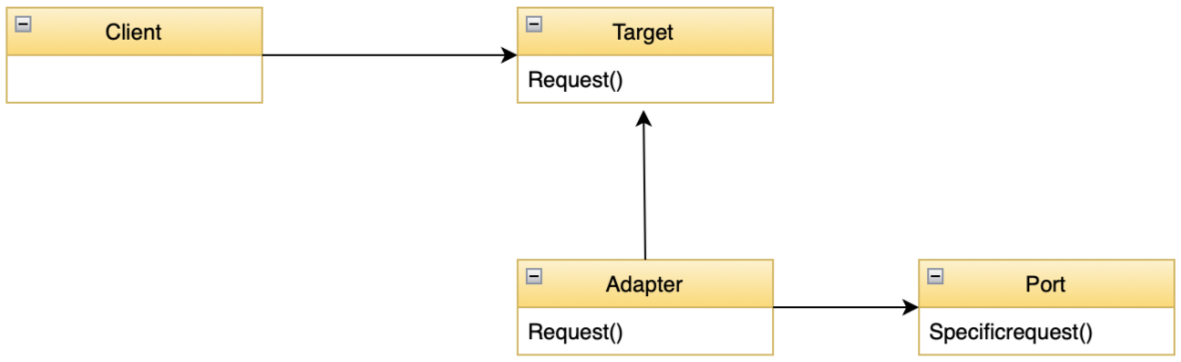


Object Adapter Pattern:

It is one of the adapter patterns. Object adapter pattern transforms a class type of interface into a different interface that clients anticipate. Classes that would not have been able to interact due to conflicting interfaces can now do so thanks to adapters. The object adapter employs composition and can wrap classes, interfaces, or both.

Example: An USB to Ethernet adapter is an example for object adapter pattern. When we have a USB port on one end and an Ethernet interface on the other, we require this.

How and why the pattern is applied: By using the target interface to call a method on the adapter, the client requests it for something. Using the adapter interface, the adapter translates that request for the adapter. Client will receive call results but is not aware of the presence of the adaptor. When an object (the Adapter) offers the same functionalities as the Client but exposes a different interface and the Client expects a certain sort of object, adapters are employed in design.



Differences and Similarities

Bridge pattern and Command Pattern:

Differences:

- Bridge Pattern is utilized for structural decisions, whereas Command Pattern is utilized for behavioral decisions.
- While Command Pattern is focused with making algorithms more interchangeable, Bridge Pattern will separate the abstract aspects from the implementation specifics.

Similarities:

- There are very few similarities between both the patterns which are not useful in the real world like structure of them.

Command and Object Adapter Pattern:

Differences:

- In the Command pattern, we switch an object's interface to a standard Command interface and provide an execute method that calls the actual, functional methods.
- Using the Object adapter pattern, we can use an object from a different interface by changing its interface.

Similarities:

- Two related design patterns, Command and the Object Adaptor, both offer a practical method of indirection that

conceals needless complexity and yet offers a clean interface.
- The Command and Object Adaptor patterns can both be very helpful in preventing the need to alter legacy code and helping to protect it. This outstanding quality emphasizes how crucial it is to have a thorough understanding of design patterns.

Object Adapter and Bridge Pattern:

Differences:
- Only until the application is designed may a bridge pattern be used. allows an abstraction and implementation to change separately.
- But an adapter pattern enables the coexistence of classes that are incompatible with one another.
- The key difference between these patterns lies in their intents.

Similarities:
- Both encourage flexibility by giving another item some degree of indirection.
- Both entails sending this object queries from interfaces other than its own.

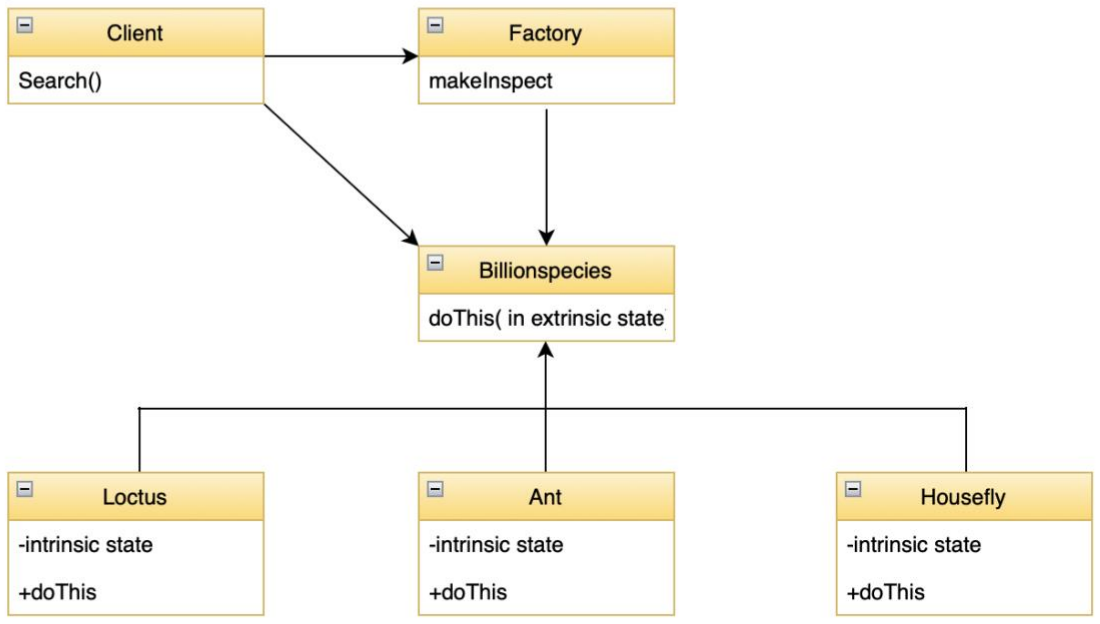
2. Flyweight, Prototype, and Singleton Pattern

Flyweight Pattern:

The flyweight software design pattern describes an item that uses the least amount of memory by sharing part of its data with other things that are like it. These designs encourage flexible object-oriented programming, which is simpler to use, modify, test, and reuse. When it is necessary to produce a huge number of identical things, the flyweight pattern is utilized.

Example: Finding out the species of the animal is an example of flyweight pattern.

How and why the pattern is applied: The repository of a Factory houses flyweights. Instead of directly producing Flyweights, the customer restrains herself and asks the Factory for them. It is impossible for a Flyweight to stand alone. When a request is made of the Flyweight, any characteristics that might prevent sharing must be provided by the client. The Flyweight pattern delivers suitable leverage if the situation allows for "economy of scale" (i.e., the client can quickly compute or search up the required characteristics).

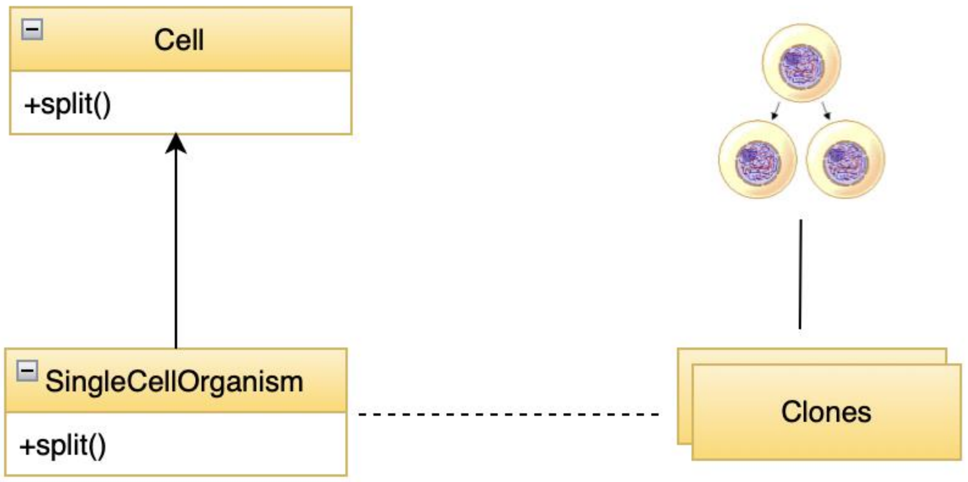


Prototype Pattern:

The prototype pattern is a creational design pattern used in software development. It is employed when a prototype instance, which is cloned to make new objects, specifies the kind of objects to create. Like the factory method design, this pattern is used to prevent object creator subclasses in client applications and to avoid the inherent cost of generating a new object the conventional way (for example, by using the "new" keyword) when it is too costly for a certain application.

Example: A cell's mitotic division, which produces two identical cells, is an example of a prototype.

How and why the pattern is applied: Using a prototype instance, the Prototype pattern determines the types of objects to produce. Before going into full production, prototypes of new items are frequently created, however in this case, the prototype is passive and does not actively reproduce itself. One instance of a prototype that actively participates in reproducing itself and hence exhibits the Prototype pattern is the mitotic division of a cell, which yields two identical cells. Two cells with the same genotype are produced when a cell divides. Alternatively put, the cell duplicates itself.

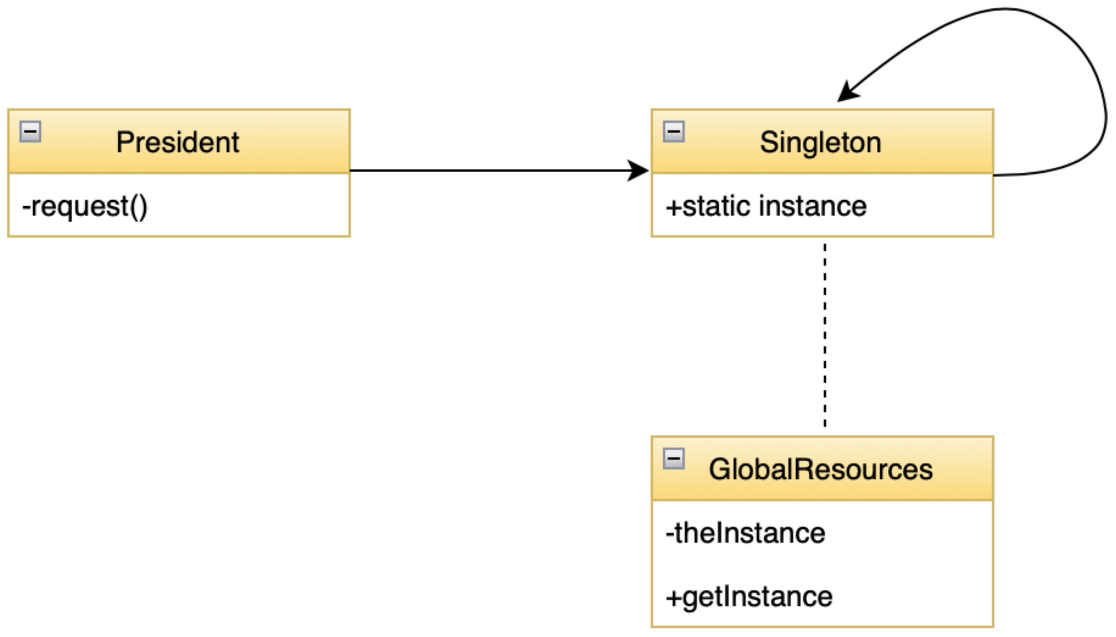


Singleton Pattern:

The singleton pattern is a design pattern that limits the number of objects that may be created from an instance of a class. One of the simplest design patterns in Java is the singleton pattern. This kind of design pattern falls under the category of a creational pattern since it offers one of the finest ways to make an object.

Example: Giving access to president to identify person in the president office is an example of singleton pattern.

How and why the pattern is applied: A class is guaranteed to have just one instance according to the Singleton pattern, which also gives a single point of access to that instance globally. It bears the same name as the singleton set, which is known as a set with only one element. The position of president is a Singleton. The United States Constitution outlines the process by which a president is chosen, establishes a term limit, and sets the succession rules. As a result, there can only ever be one president in office at any given moment. No matter who the current president is personally, the title "The President" serves as a universal point of access to identify the person holding the position.



Differences and Similarities

Flyweight and Prototype Pattern:

Differences:

- When using the prototype pattern, we simply clone one object repeatedly.
- In the Flyweight pattern, newly created objects are only created when there isn't a similar or identical object already present in the application.
- The prototype is a design pattern that was created, whereas flyweight is a structural pattern.

Similarities:

- In Flyweight and Prototype, an immutable object will be shared across the clients rather than being created from scratch.
- Flyweight design makes sense to utilize when using many objects in the application, whereas with Prototype we might clone even one object.

Prototype and Singleton Pattern:

Differences:

- All stateful beans should utilize the prototype scope, whereas stateless beans should use the singleton scope.
- A singleton bean definition is one for which only one instance is produced for each Spring IoC container, and the same object is shared for each request made for that bean.

- Every time a request is made for a certain bean definition, a prototype—a fresh instance—is constructed.

Similarities:

- Creational Design Patterns, which deal with the creation of objects, include the Singleton and Prototype patterns.

Singleton and Flyweight Pattern:

Differences:

- Singleton is a creation pattern whereas Flyweight is a structural pattern.
- Usually but not always, the singleton pattern is used to limit the number of instances of a class that may be produced to a preset constant.
- Utilizing sharing allows for the effective support of several fine-grained objects in the flyweight pattern.

Similarities:

- Both the singleton and flyweight pattern represent the same object.
- When extraordinarily many objects both the patterns will be same.

3. Proxy, Strategy and Visitor

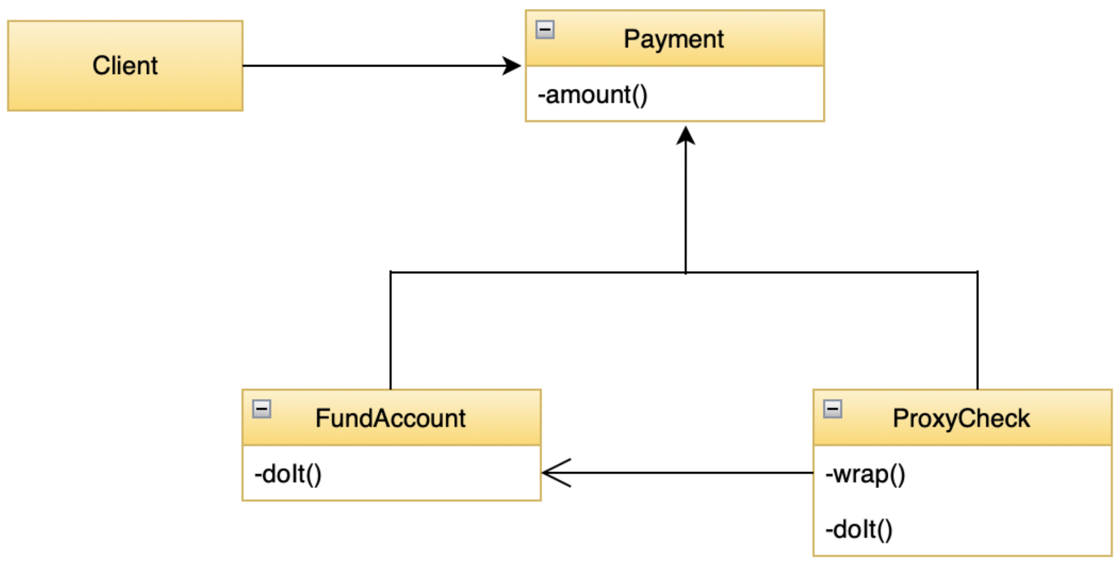
Pattern

Proxy Pattern:

A pattern for software design is the proxy pattern. In its broadest sense, a class acting as an interface to another object is what constitutes a proxy. A class represents the functionality of another class in a proxy design. The structural pattern category includes this kind of design pattern. By creating an object using the original object, we may interface its functioning with the outside world.

Example: The money in an account is represented by a check or bank draft.

How and why the pattern is applied: By specifying a Subject interface, the client is unaware that the Proxy object is there in lieu of the Real Subject. The Proxy offers a stand-in or substitute to grant access to an item. The money in an account is represented by a check or bank draft. The ability to access funds in the issuer's account is ultimately controlled by a check, which can be used to make transactions instead of cash.

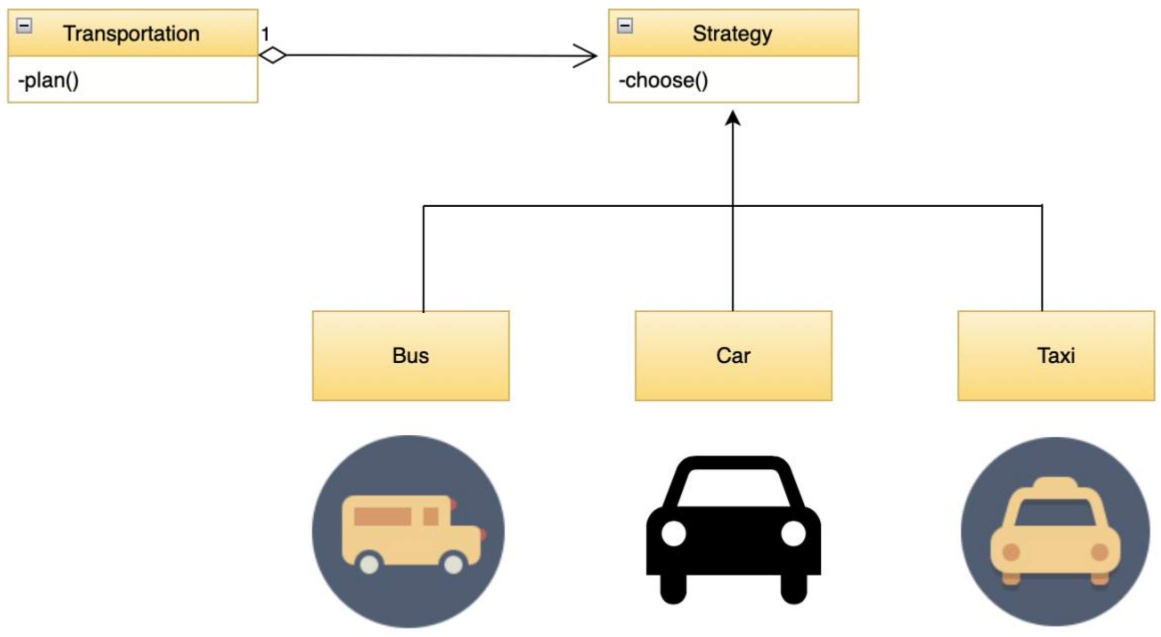


Strategy Pattern:

A behavioral software design pattern that allows choosing an algorithm at runtime is the strategy pattern (sometimes referred to as the policy pattern). Code receives the run-time instructions on which algorithm to utilize from a family instead of implementing a single method directly. Using a strategy, the algorithm may change without affecting its users' customers.

Example: Planning to travel to airport is an example of strategy pattern.

How and why the pattern is applied: A collection of the interchangeable algorithms are defined by strategy. An example of a strategy is the means of transportation to an airport. Driving one's own vehicle, using a cab, an airport shuttle, a city bus, or a limousine service are some choices. Subways and helicopters are additional options for getting to some airports' terminals. A traveler can reach the airport using any of these methods of transportation, and they are all interchangeable. The traveler's Strategy must consider trade-offs between price, comfort, and time.

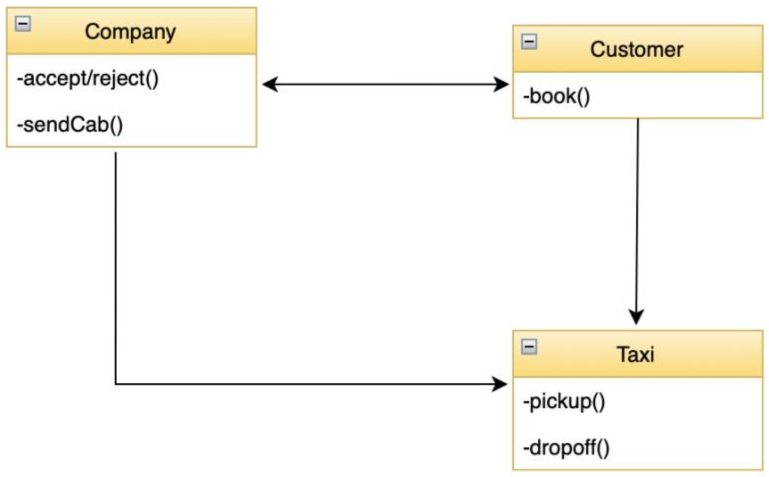


Visitor Pattern:

The visitor design pattern is a technique used in software engineering and object-oriented programming to decouple an algorithm from the object structure it relies on. The ability to add new operations to existing object structures without changing the structures is a useful outcome of this separation. A visitor pattern's objective is to specify a new action without changing an existing object structure.

Example: A person booking a taxi by the calling taxi service company is an example of visitor pattern.

How and why the pattern is applied: The Visitor pattern denotes an action that may be carried out on an object structure's components without altering the classes on which it depends. This pattern may be seen in how a taxi firm runs. When a consumer phones a taxi company, the business sends out a cab to pick them up. Once inside the taxi, the client (or visitor) loses control of their own transportation and is instead under the responsibility of the taxi (driver).



Differences and Similarities

Proxy and Strategy Pattern:

Differences:

- While Strategy offers a different interface that is compatible with its client, proxy offers the same interface.
- Behavioral decisions are made using the strategy pattern, while structural decisions are made using the Proxy pattern.

Similarities:

- Both the patterns are responsible for representing the object located remotely.
- Both the patterns provide different interface to the clients.

Strategy and Visitor Pattern:

Differences:

- Different algorithms are exposed to a standardized interface using a strategy pattern.
- An object can accept a reference to another object (the visitor) that exposes a predetermined interface that the target object can call upon itself using a mechanism described in the Visitor pattern.

Similarities:

- The approach known as a Visitor has numerous procedures and allows for double dispatch.
- Additionally, reliable run-time coupling between two physical objects is made possible by the Visitor.

Visitor and Proxy Pattern:

Differences:

- Proxy pattern deals with the structural decisions while Visitor pattern deals with composite structural decisions.
- Visitor pattern works with composite whereas Proxy works with structural pattern.

Similarities:

- Both the patterns work on some type of structural decisions.

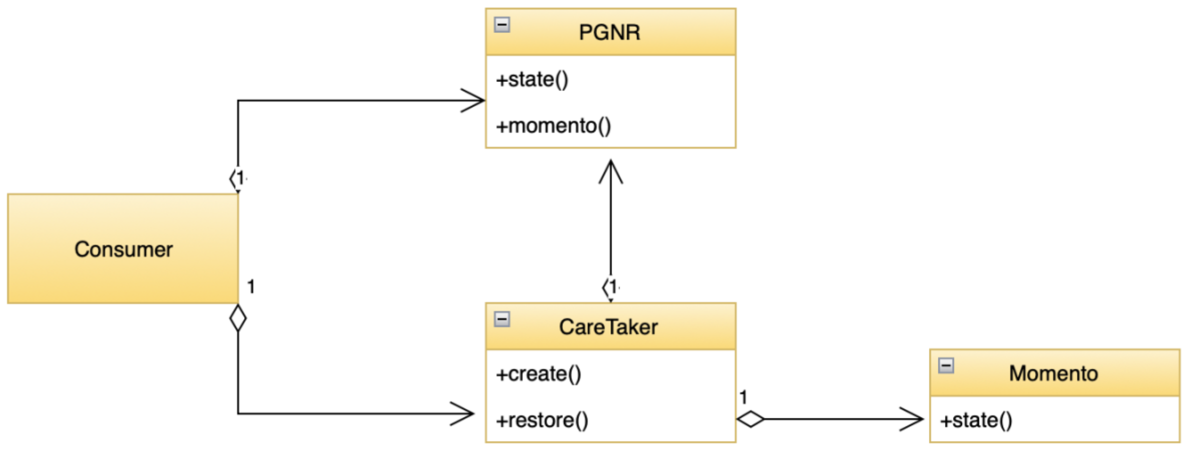
4. Momento and State Pattern

Momento Pattern:

Software design pattern known as the memento reveals an object's internal private state. The ability to roll back an item to its prior state is one use of this; other examples include versioning and custom serialization. The creator, a caretaker, and a memento are the three things used to implement the memento pattern.

Example: A pseudorandom number generator is one of the classic examples of the memento pattern.

How and why the pattern is applied: Every PRNG consumer acts as a caretaker who may initialize the originator (the PRNG) with the same seed (the memento) to generate the same series of pseudorandom numbers and the state in a finite state machine. Three actor classes are used in the Memory pattern. A restored state of an item is contained in Memento. Memento objects' originators generate and store states, and caretakers oversee retrieving those states from Memento. Memento, Originator, and Caretaker are classes that we have developed.

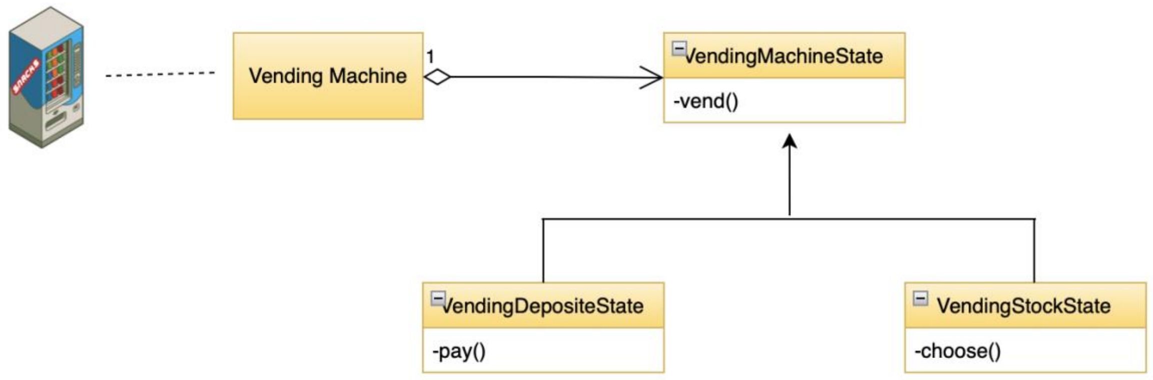


State Pattern:

A behavioral software design pattern called the state pattern enables an item to modify its behavior when its internal state does. Finite-state machines are a notion that is like this pattern. The state pattern may be thought of as a strategy pattern that can change a strategy by calling methods that are specified in the pattern's interface.

Example: Vending machine working process is an example of a state pattern.

How and why the pattern is applied: The State pattern enables an object to alter its behavior in response to changes in its internal state. A vending machine will display this pattern. The inventory, money placed, capacity to make change, item picked, etc., all affect the states of vending machines. When money is deposited and a choice is made, a vending machine will either provide a product with no change, a product and change, no product because there is not enough money put, or no product because the inventory has run out.



Differences and Similarities

Momento and State Pattern:

Differences:

- The main goal of Memento is to reveal private state in an unchangeable opaque object. Undo is only one specific use that it may be used for.
- State patterns deal with how an entity behaves differently when its internal private state shifts. This is purely a behavioral tendency and has nothing to do with state preservation.

Similarities:

- Both the momento and state patterns falls under the behavioral patterns.