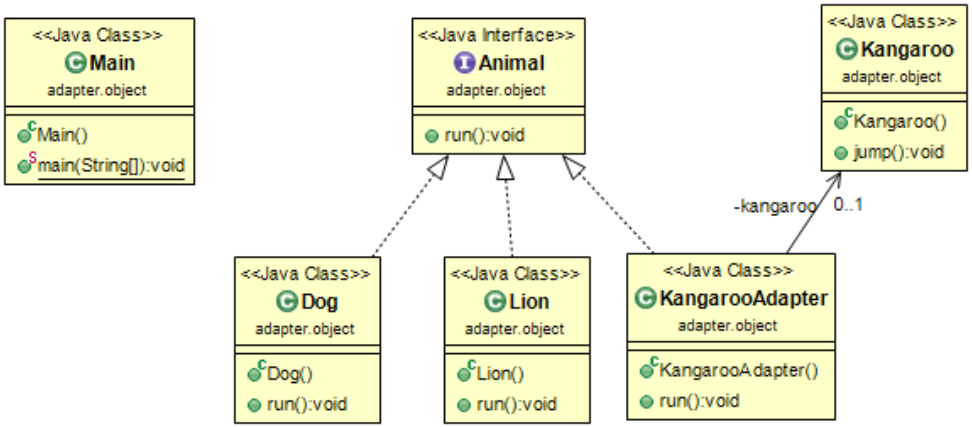
ADAPTER

An adapter helps to join two incompatible interfaces to work together. So, if you have an interface with implementing classes. If you were asked later to add additional sub class(es), but they have incompatible Interface, then, adapter pattern could be useful. There are two structures:

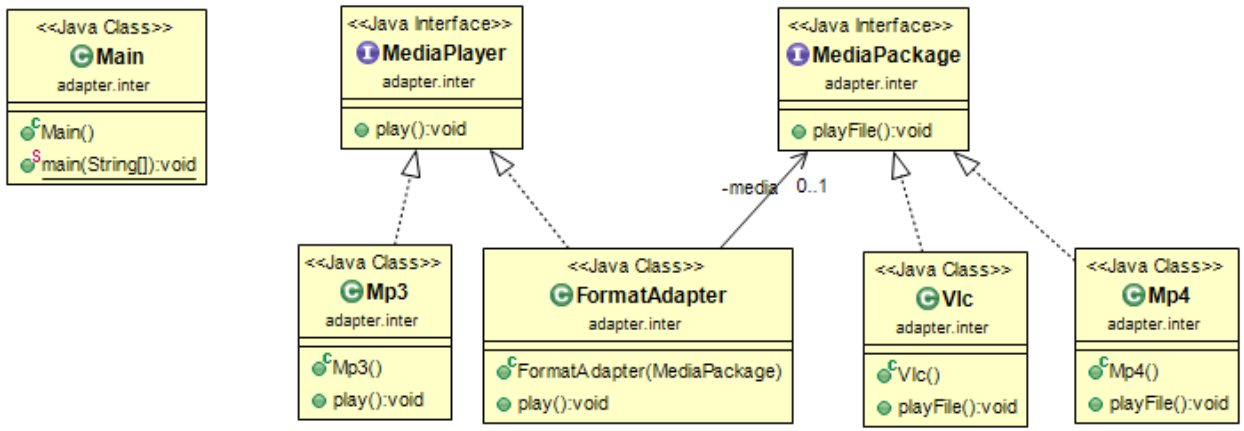
Object

The Adapter has a reference to the incompatible object.



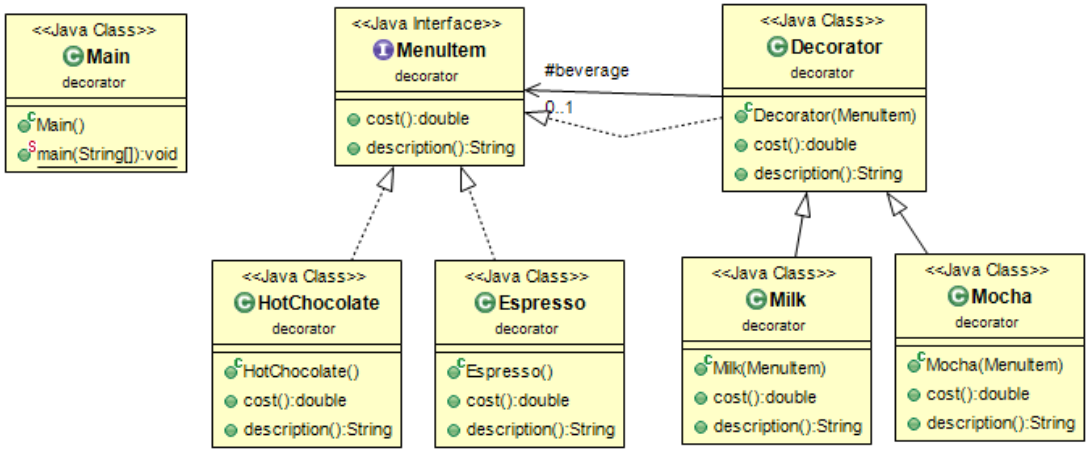
Interface

The Adapter has a reference to the incompatible interface.



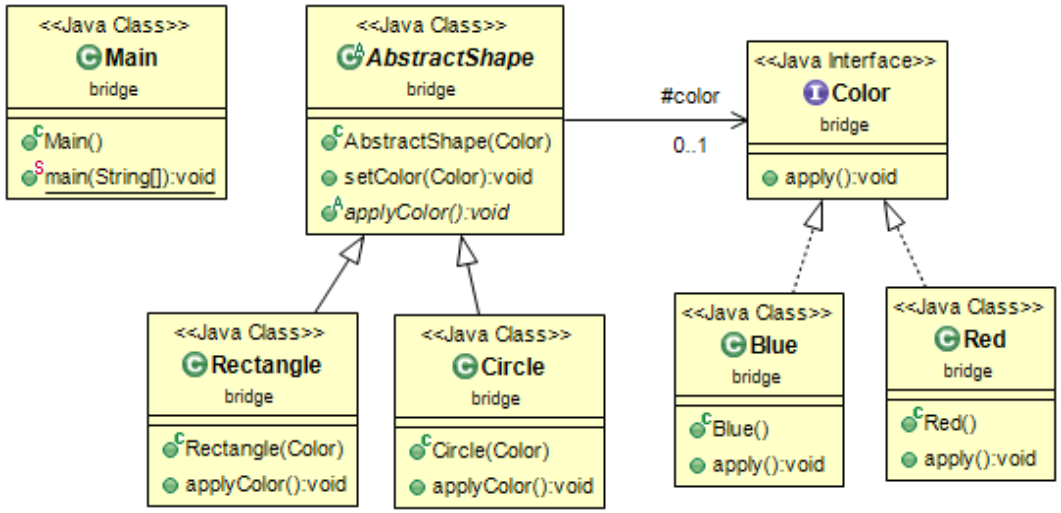
DECORATOR

The decorator pattern extends the functionality of an object dynamically.



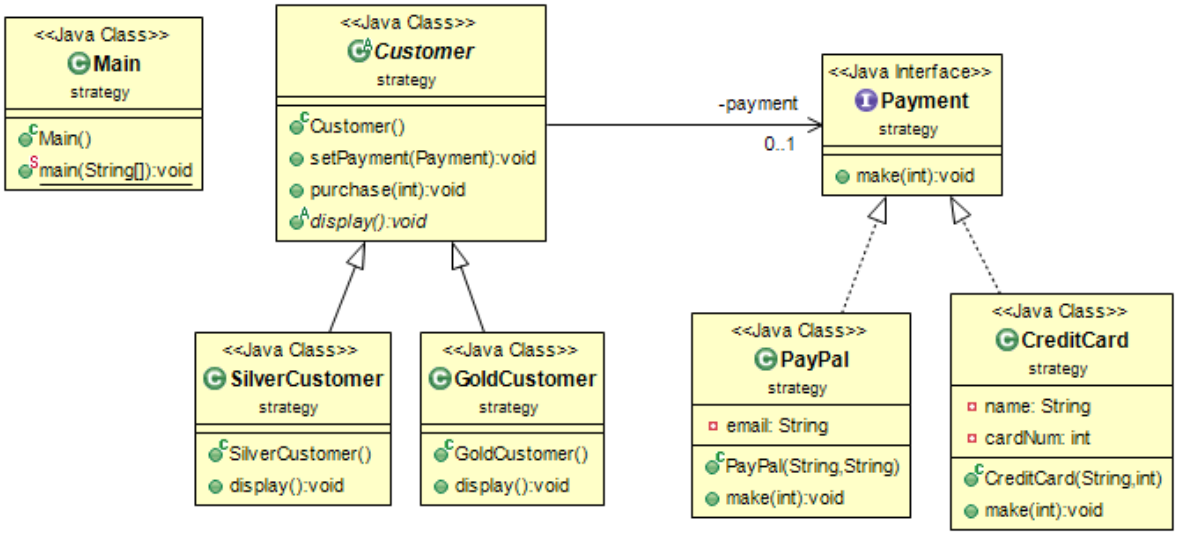
BRIDGE

Decouples an abstraction from its implementation so that the two can vary independently. As an example, If you have a class called Rectangle. This class could have two different implementations, Red Rectangle and Blue one. Instead of Inheriting from Rectangle class, one for blue rectangle and another for red, We could instead pull out these implementations and use Composition over Inheritance.



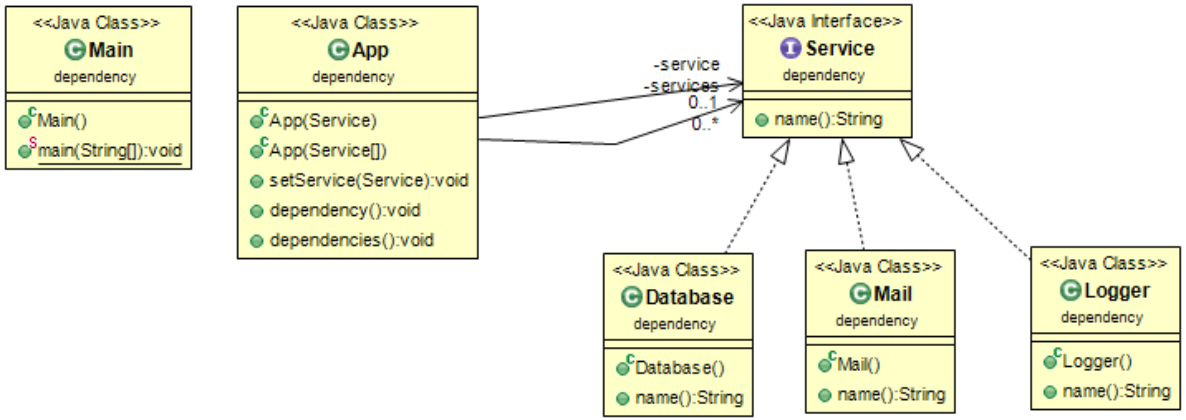
STRATEGY

Strategy is used when you want to extend the behavior of an Object, where this behavior could vary during the run time. If multiple objects need to use the same behavior(algorithm), we get the benefit of code reuse too.



DEPENDENCY INJECTION

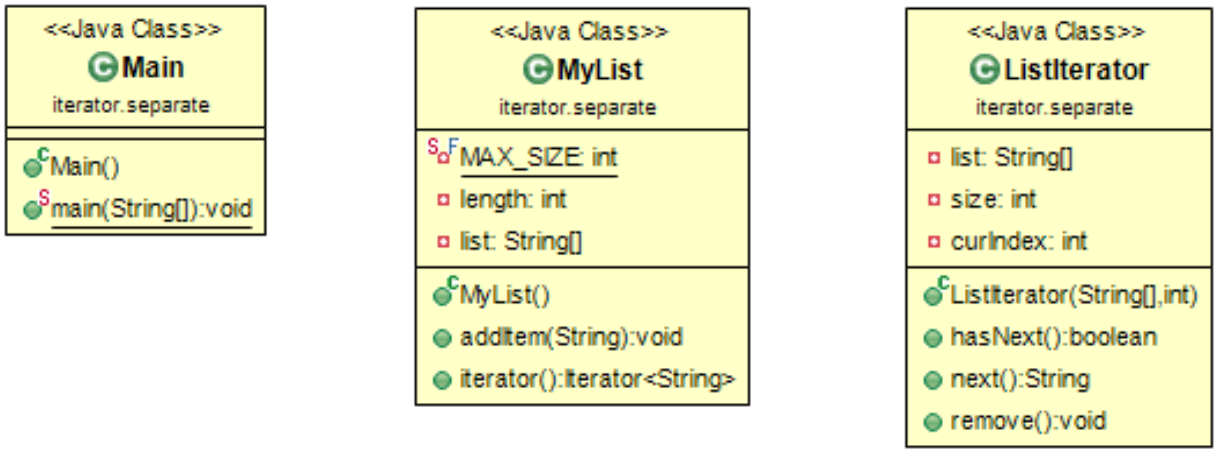
Dependency is used when you want to separate the dependencies of an Object, and pass them to dependent object during run time. The



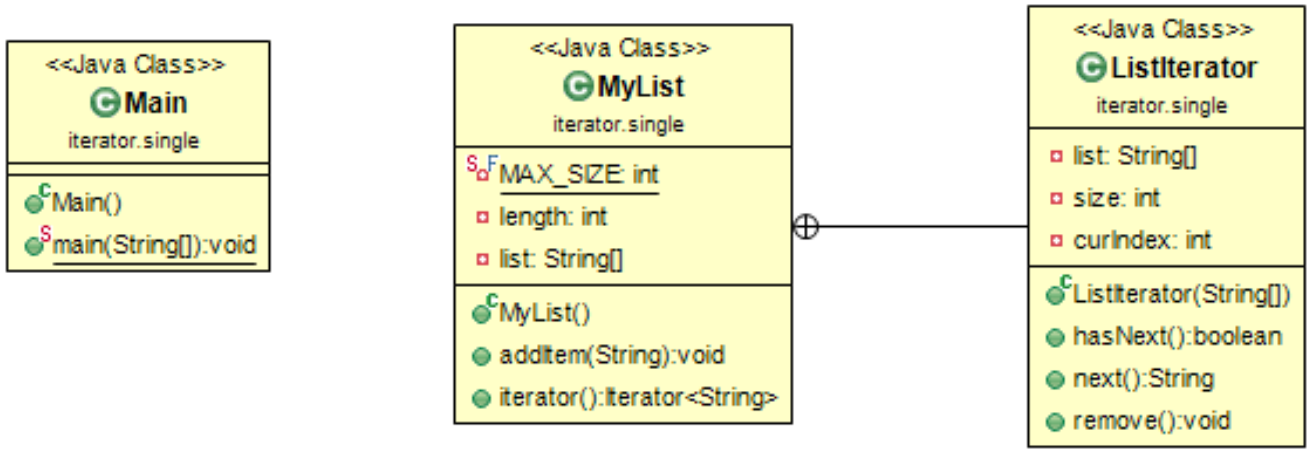
ITERATOR

This pattern is used to get a way to access the elements of a collection object in sequential manner without exposing its underlying representation.
In this snippet, I am using Java's built-in Iterable & Iterator classes.

Separate Class

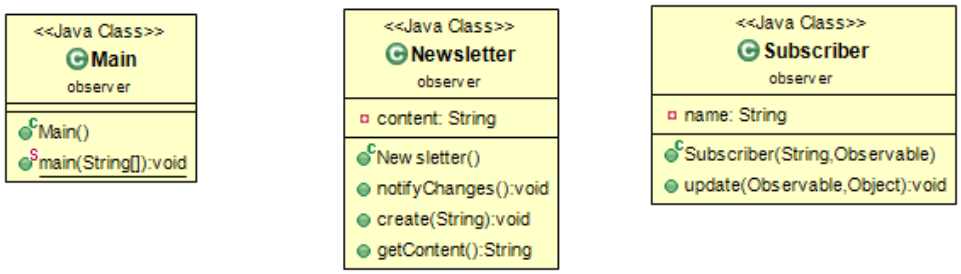


Singles Class



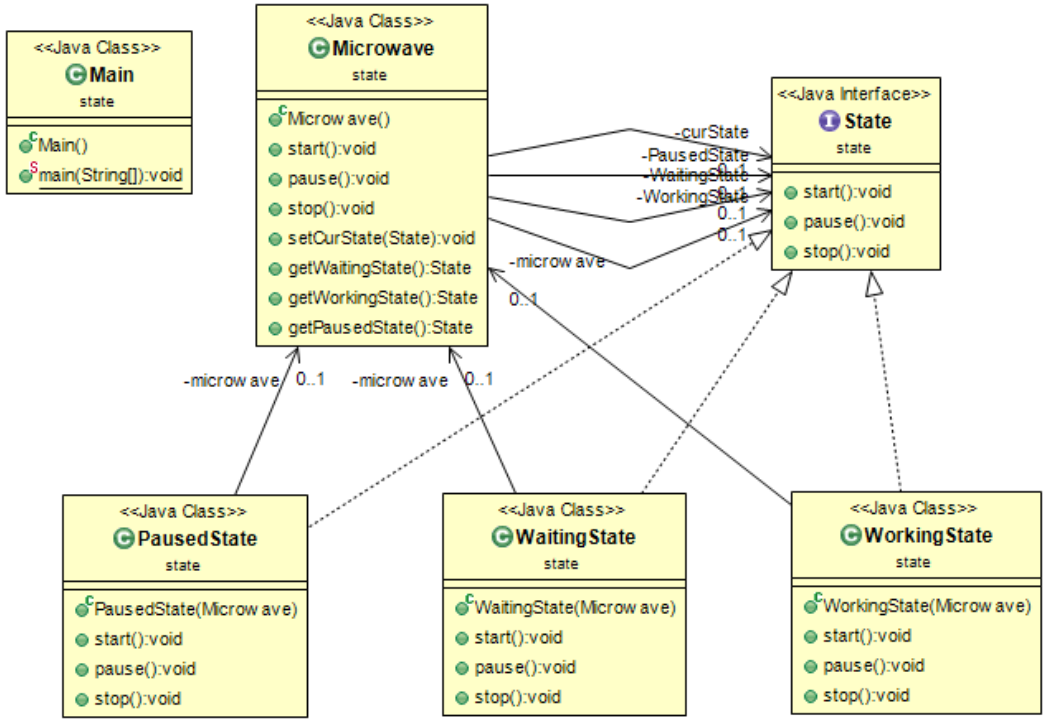
OBSERVER

Observer pattern is used such that if an object is changed, its dependents objects get notified of that change, Thus, there is 1:M Relationship. As an example, having a Publisher that publish news to the Subscribers, Whenever any new updates or data added, the Subscribers get notified. In this snippet, I am using Java's Observer and Observable classes.



STATE

A class behavior may change based on set of states either made by user, or internally by the system. In this pattern, We encapsulate each state. The user doesn't need to know about each state, the user only performs some actions which in turn may change the state of the object.

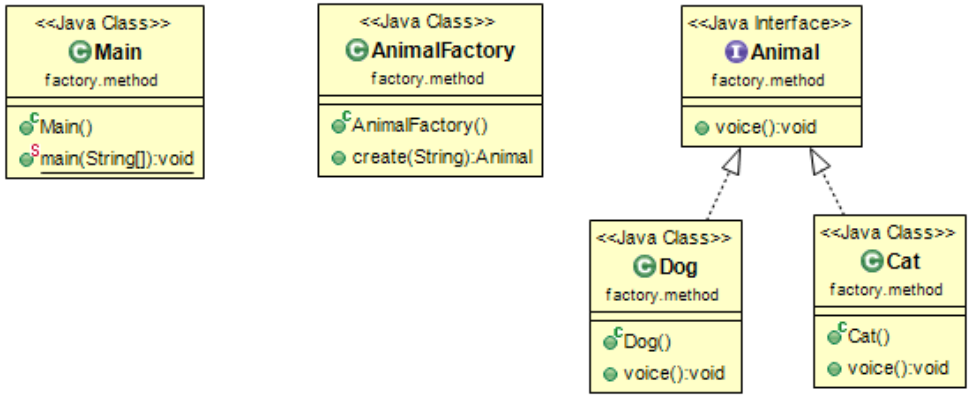


FACTORY

This pattern defines a way for creating object(s) during run time.

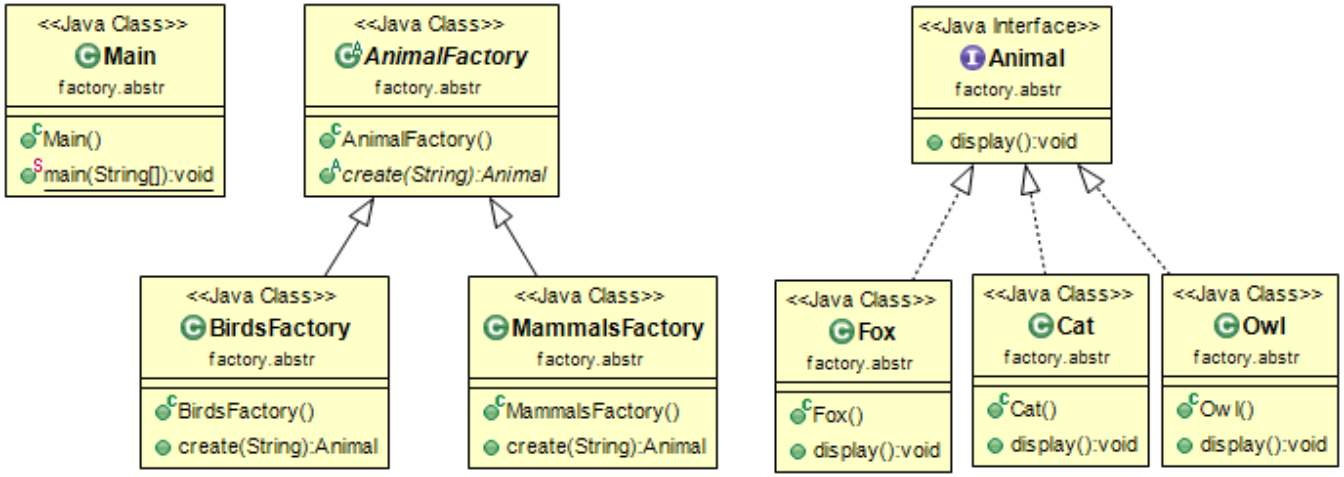
Factory Method

Factory Method is a method used to create object(s) of a certain type(interface) during run time.



Abstract Factory

Factory Method is an object used to create a set of related objects during run time.

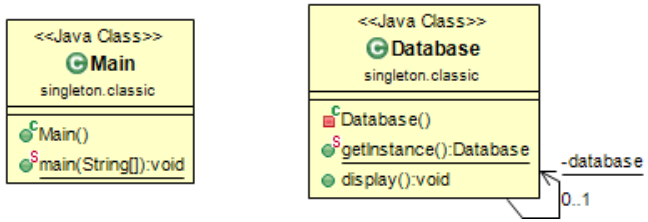


SINGLETON

The Singleton Pattern is a pattern that ensures that there is only ever one single instance of a class, And it provides a global way to get to that instance.

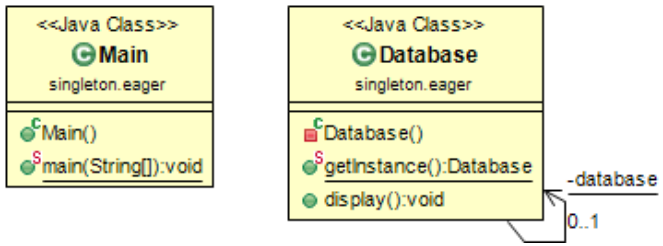
Classic

This is the basic implementation



Eager Instantiation

If you are concerned about synchronization, eager intantiation could be useful as long as you know you'll always need to instantiate the object, and the object doesn't take a lot of time to load.



Synchronized

Another solution for synchronization using synchronized method. But, you will pay for it's pitfall; Synchronized code takes a lot longer to run.