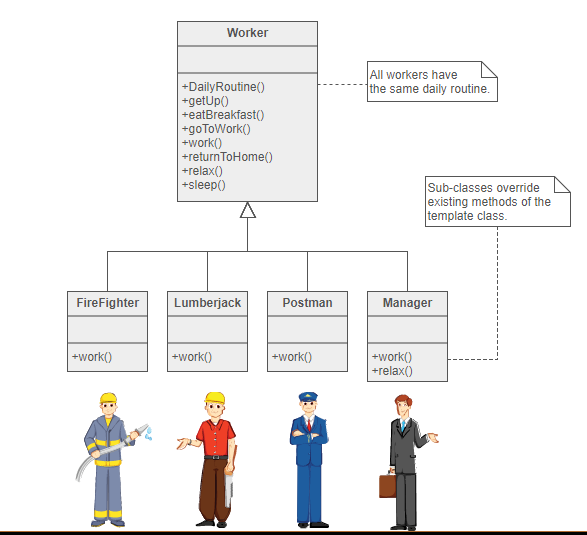
Design pattern pt.2

# Template method

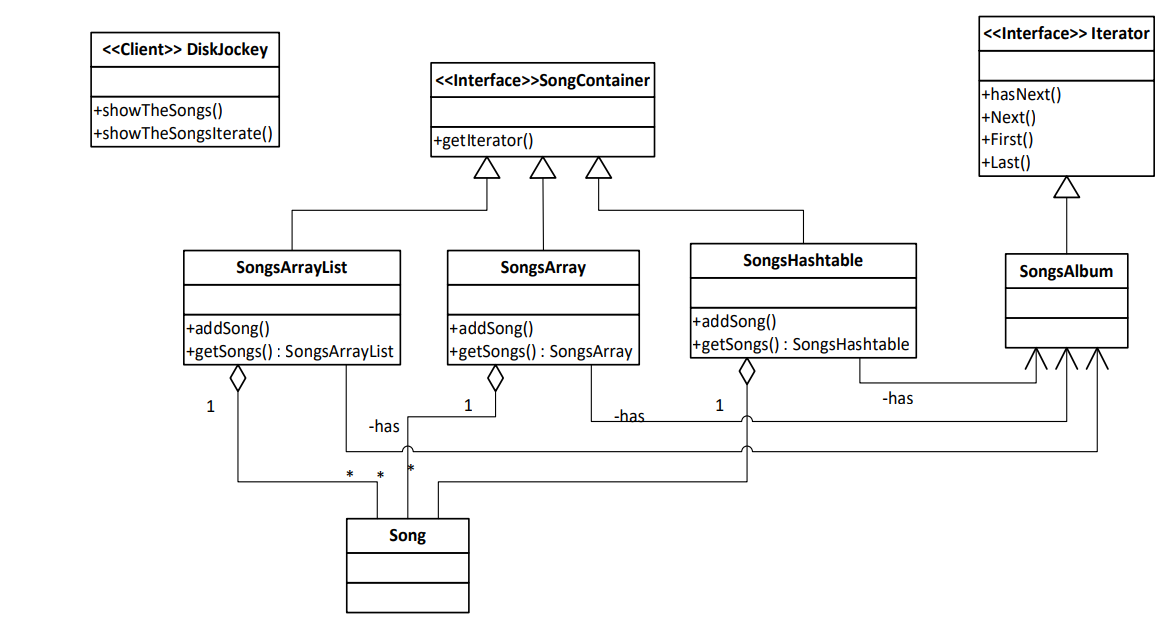
Template method lets us redefine certain steps of an algorithm without changing the algorithm structure.



Strategy design pattern vs template method – strategy changes entire algorithm, while template changes only part of algorithm.

# Iterator

This pattern is provides a way to access the elements of a collection object in sequential manner without knowing its underlying structure.

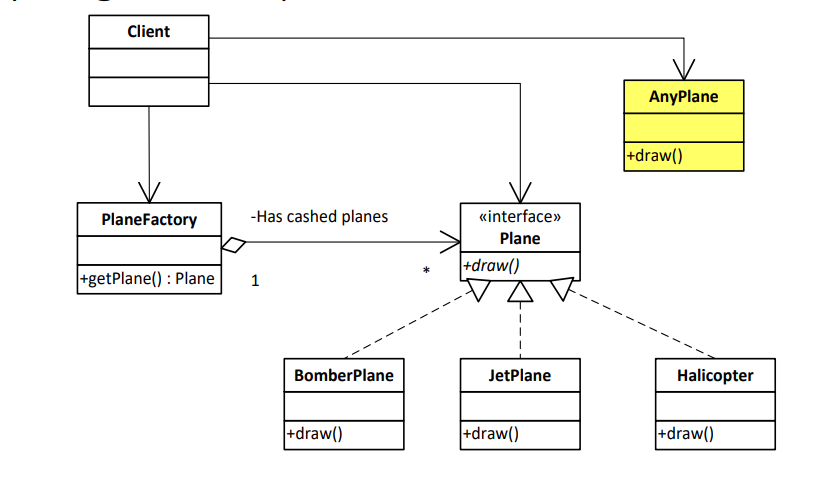


# Flyweight

“Use sharing to support large numbers of fine-grained objects efficiently”

• The key idea is to minimize objects quantity by sharing them.

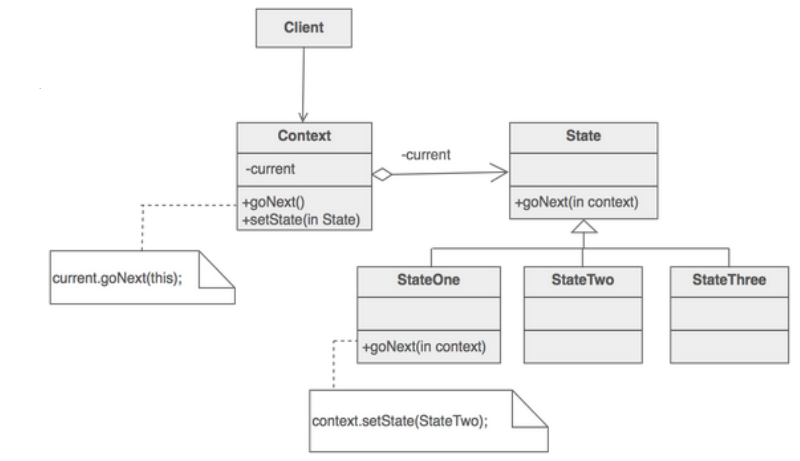
• Objects have part of their internal state in common where the other part of state can vary.



# State

Allow an object to alter its behavior when its internal state changes.

• The State pattern helps avoiding if-else or switch-case conditional logic in this scenario.



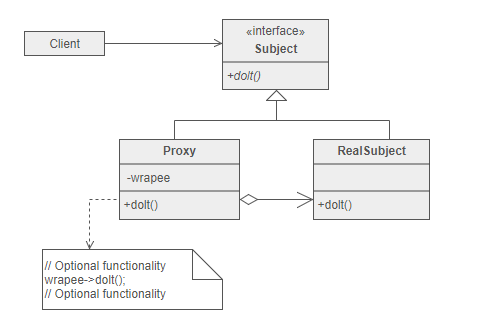
# Proxy

Provide a surrogate or placeholder for another object to control access to it.

1 to defer the full cost of its creation and initialization until we actually need to use it (controlling when a costly object needs to be instantiated and initialized)

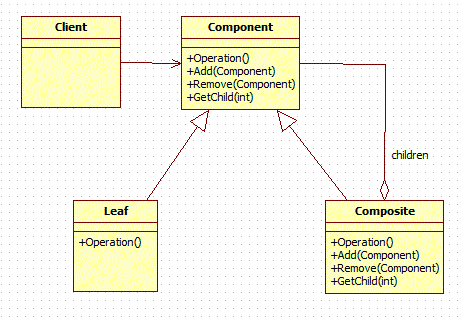
2 giving different access rights to an object.

1. An object, such as a large image, takes a long time to load.
2. The results of a computation take a long time to complete, and you need to display intermediate results while the computation continues.
3. The object is on a remote machine, and loading it over the network may be slow, especially during peak network load periods.
4. The object has limited access rights, and the proxy can validate the access permissions for that user.



# Composite

Composite design patten allows you to have a tree structure and ask each node in the tree structure to perform a task.You can take real life example of a organization.It have general managers and under general managers, there can be managers and under managers there can be developers.Now you can set a tree structure and ask each node to perform common operation like getSalary()



# Chain of Responsability

The chain of responsibility pattern is used to process a list or chain of various types of request and each of them may be handle by a different handler. This pattern decouples sender and receiver of a request based on type of request.

In this pattern, normally each receiver (handler) contains reference to another receiver. If one receiver cannot handle the request then it passes the same to the next receiver and so on.

