# Chapter 9: State Design Pattern

## Identification

* Behavioural
* Delegation
* Allow an object to alter its behaviour when its internal state has changed. The object will appear to change its class.

## Structure

* Diagram

  Description automatically generatedState: define an interface to encapsulate the behaviour of a particular state of the context
* Concrete State: implements the behaviour
* Context: maintains and keeps reference to a concrete state

## Problem: where the state pattern can work

* When an object becomes too large and if it has extensive number of state changes it can go through

## Explained

* If the system is required to change its behaviour depending on the state.
* Changes to the concrete states will not affect the rest of the system if they are changes
* Context makes use of polymorphism to alter the behaviours
* Delegation is used by the Context to carry out the request of the concrete states

## Improvements achieved

* Increased maintainability
  + All the behaviour associated with an object is under the state hierarchy. This allows for easier maintenance.
* Eliminate Large conditional statements
  + State changes are done in the subclasses (concrete state classes) instead of in if or switch statements.
* Make state transitions explicit
  + Allows for internal states to remain somewhat similar between state transitions. Used to have to change the whole internal state constantly, but concrete classes allow for a more explicit change.

## Disadvantages

* Higher Coupling:
  + Increase number of classes, and is less compact
  + Also requires more commination between classes

## Implementation Issues

* Changing of the states: who will do it
  + The Context:
    - The state criteria is fixed and the context must know all of it
    - Downside: not flexible – context must change if there are more or less concretes states created
    - It’d be better if the concrete states knew of their successor
  + Context applying variable criteria:
    - Allows for the context to not explicitly guess which state it is currently in for it to change state
    - Coupling is lowered
    - State is unaware of the Context
    - But not so memory efficient as each time a request is made to separate objects

Diagram

Description automatically generated

* + Concrete States applying variable criteria
    - Context and State must know of each other
    - Higher coupling

Diagram

Description automatically generated

## Related Patterns

* Strategy
  + Strategy is about having different implementations that accomplish the same result
  + State is about doing different things based on your state
* Singleton or Prototype
  + States are often singletons
* Flyweight
  + States object shared by applying Flyweight

## Diagram Description automatically generatedExample:

* State: Disposition
* Concrete State: BadMood, OkMood, GoodMood
  + - * Context: Boss
      * Request(): helpMe(), directMe()
      * Hanlde: helpMe(), directMe()

# Composite: