State Pattern – Behavioural

- Intent
  - » **Alter behaviour of an object when its internal state changes**
  - » **Object appears to change its class**

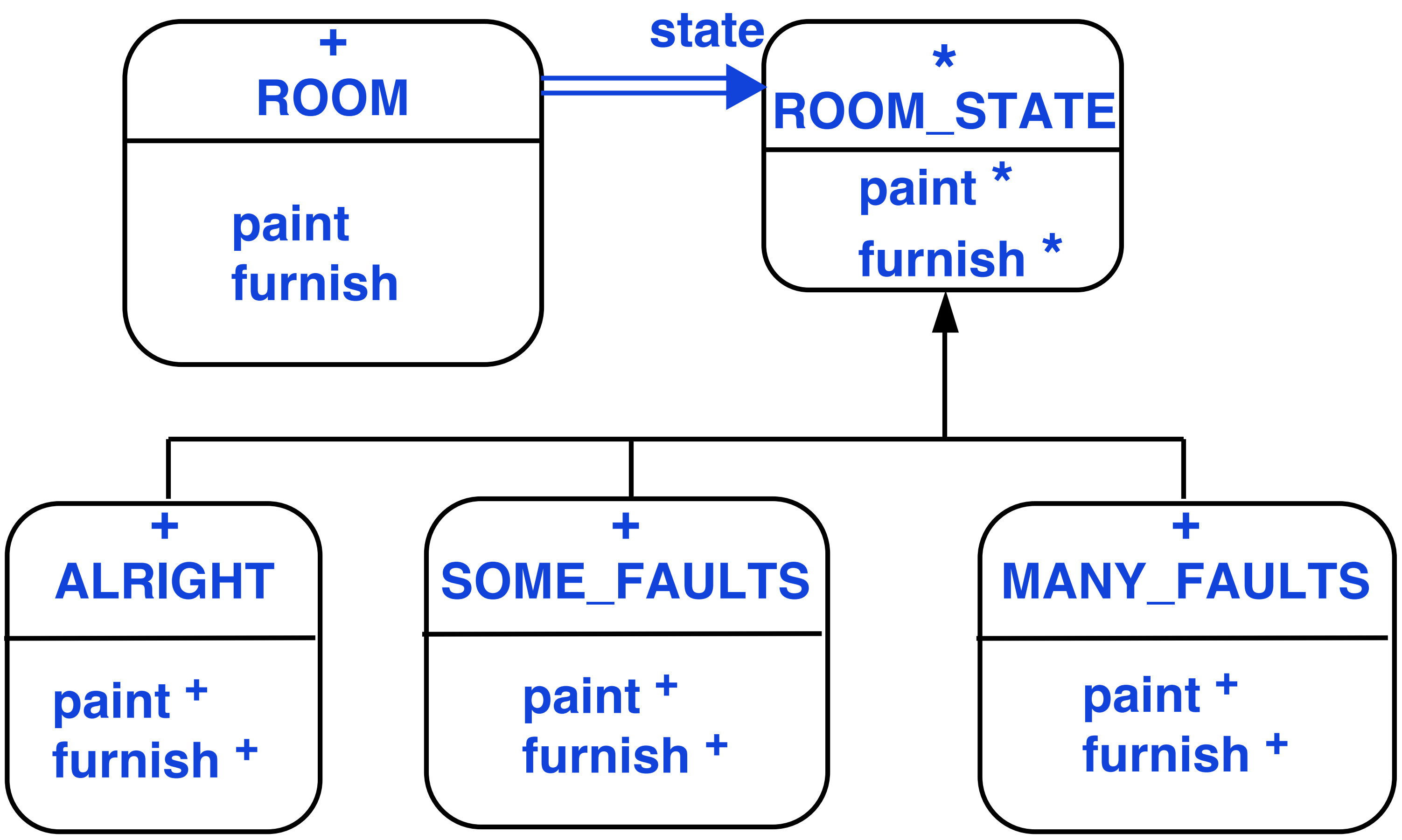
- Alternate names

  Objects for States

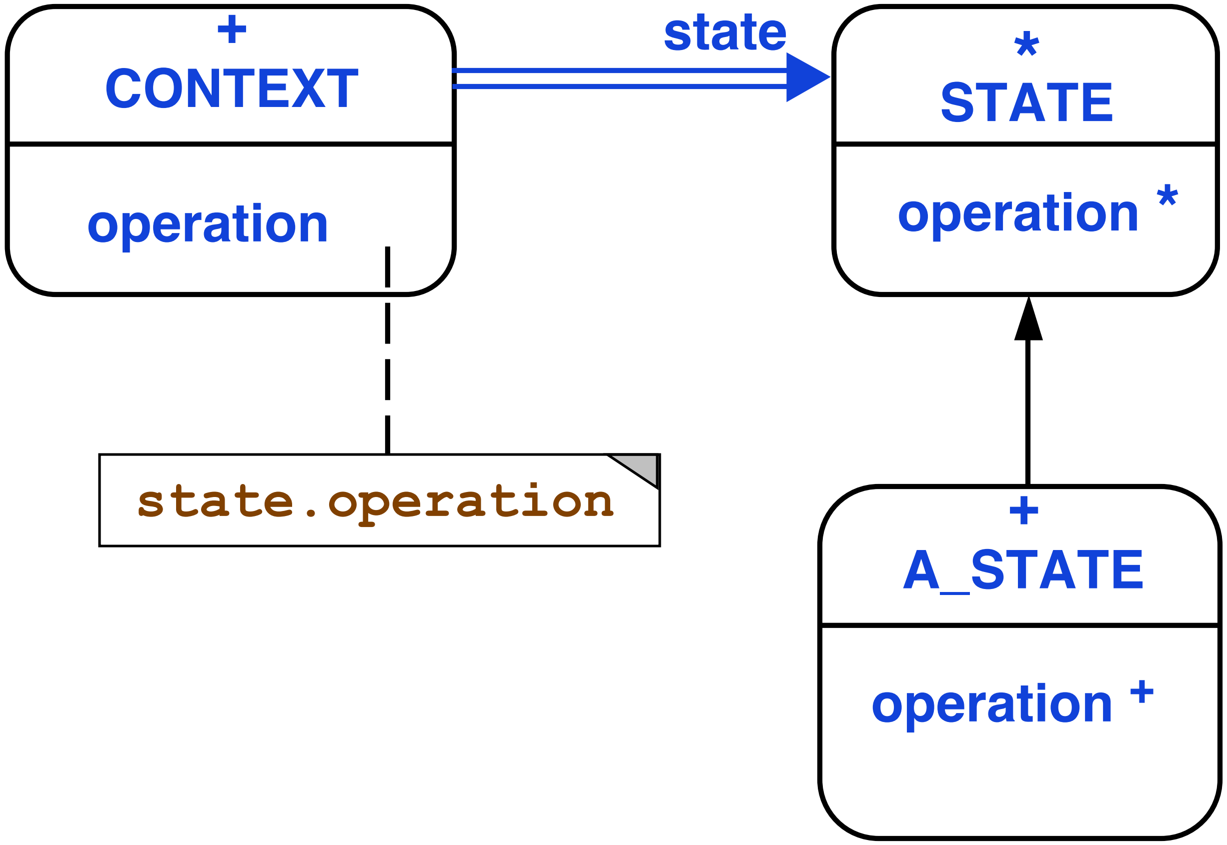
Motivation

- An object may be in one of many states. It responds differently depending upon its current state
  - » Example
    - > A Room can be in one of the states
      - – **Alright, SomeFaults, ManyFaults**
    - > A request to paint the room is made
      - – **Alright state – clean and paint room**
      - – **SomeFaults– repair yourself and paint room**
      - – **ManyFaults– hire contractor to repair and paint room**

Example Architecture



Abstract Architecture



Participants

- Context

  Defines client interface

- Deferred State

  Defines interface for common behaviour for different states

- Effective State

  Implements behaviour of that state in context

Applicability

- Object has different behaviour depending on state
- Operations have multipart conditional statement dependent upon state
  - » **State is represented by an enumerated constant**
  - » **Several operations have same conditional structure**
- Pattern puts each branch of the conditional into a separate class
  - » **Object's state becomes an object that can vary independently of other objects**

Collaborations

- Context delegates state specific behaviour to a concrete state object
- Context may pass itself as an argument so that state can access context features
- Context is the primary interface with clients
    - » **Clients configure context with state objects**
    - » **Clients do not deal directly with state objects**
- Context or concrete state can decide which state follows another state

Related Patterns

- Flyweight explains when and how State objects can be shared

- State objects are often Singletons

State in Java API

- The class **Container** represents an aggregation of **Component** objects.
  - **It has a LayoutManager that describes how the Container will display the Components**
  - **The LayoutManager object reflects the state of the Component object**
    - **If layout is a FlowLayout, then add() places the new component at the end of the list**
    - **If layout is a BorderLayout, then add() has a different implementation**
  - **Not a real state pattern use because the LayoutManager rarely changes for a given Container but could**