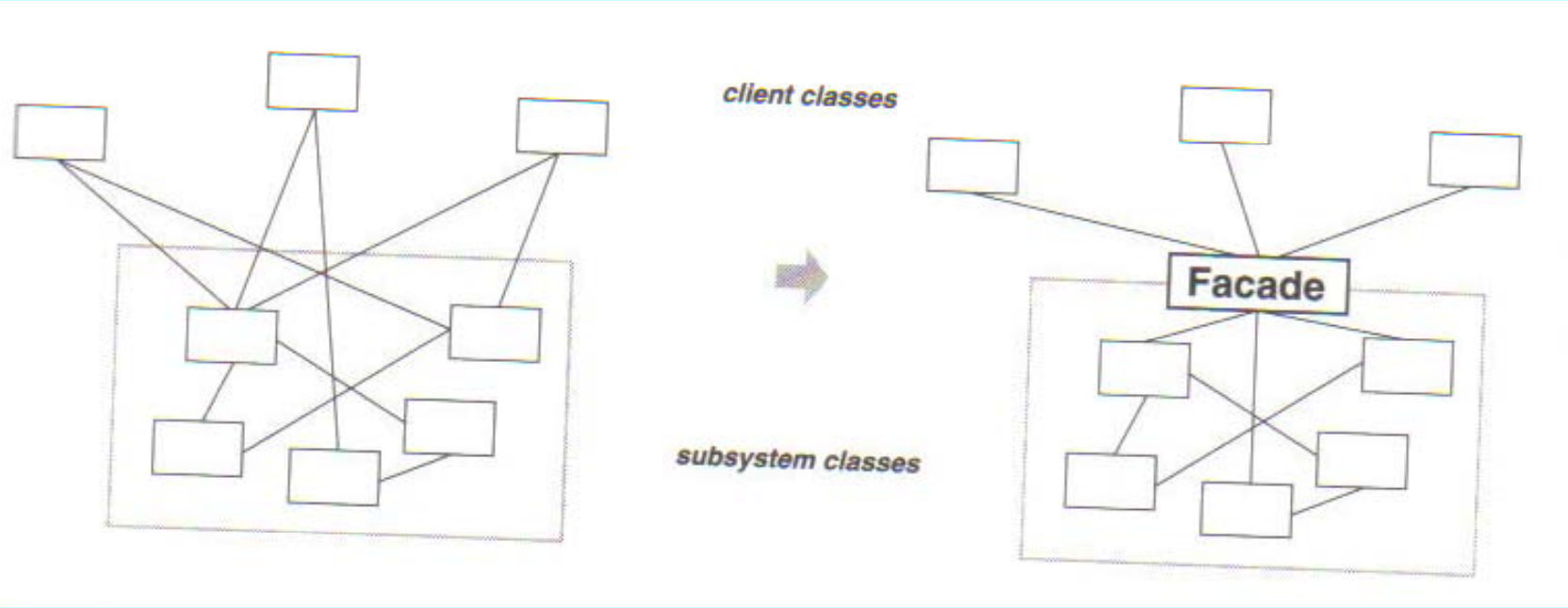
Façade Pattern

Façade Pattern (Structural Pattern):

- **Intent (GOF):**
  - **Provide a unified interface to a set of interfaces in a subsystem.**
  - **Façade defines a higher-level interface that makes the subsystem easier to use.**

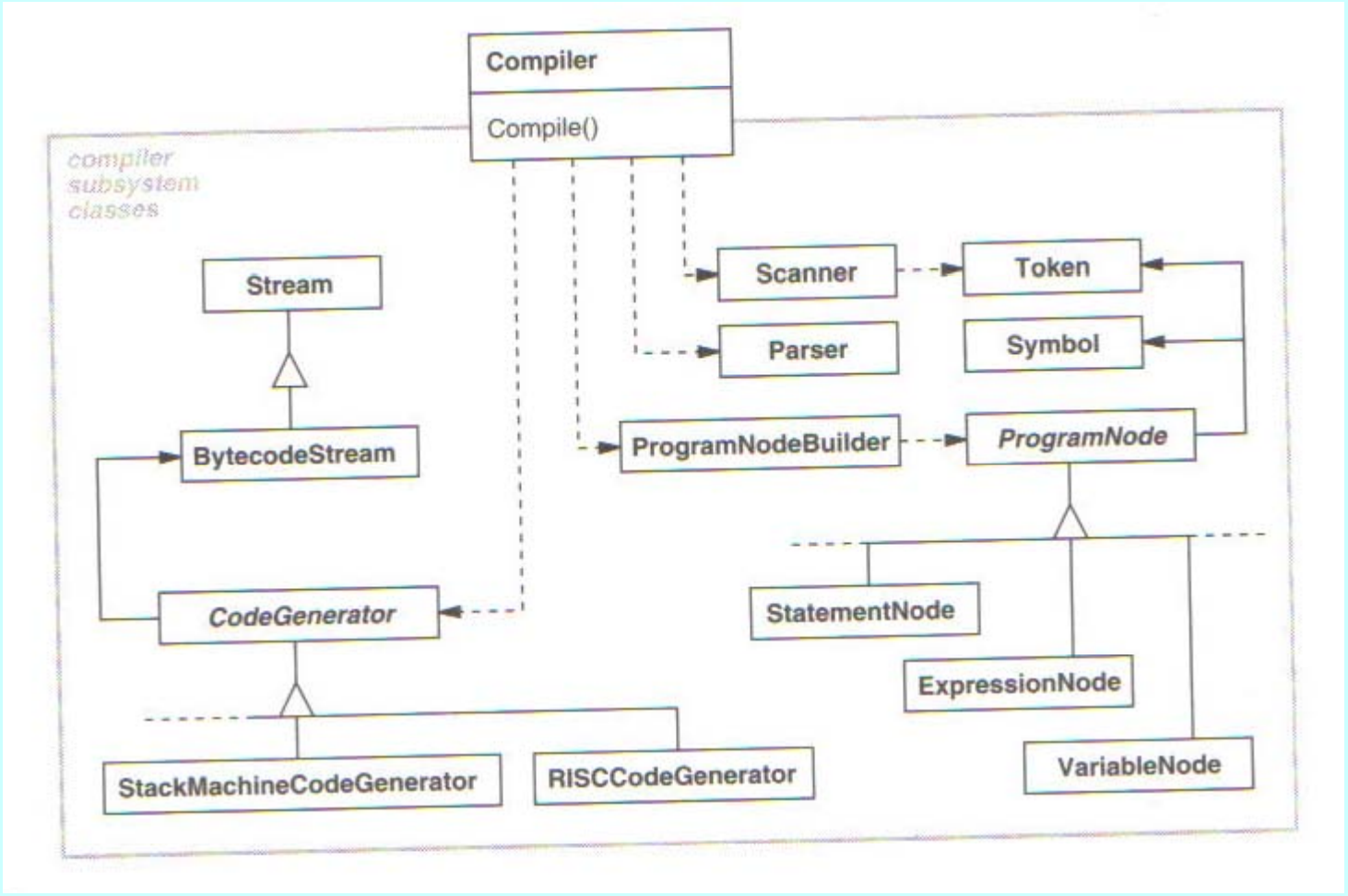
Façade Pattern (Structural Pattern):

- **Motivation:** we want to simplify how to use an existing system..



Example:

Consider a programming environment that gives applications access to its compiler sussystem.

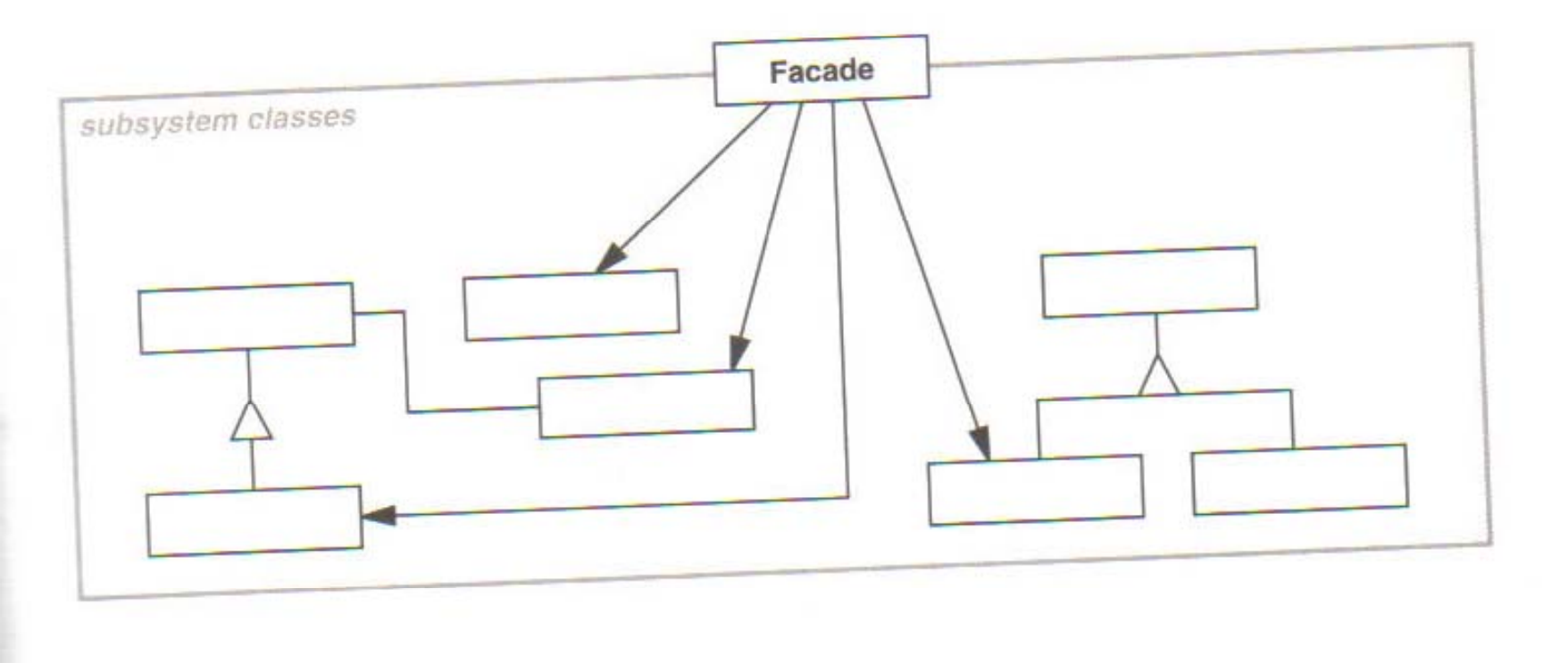


Façade Pattern (contd.):

- **Applicability: Use it when:**
  - **you want to provide a simple interface to a complex subsystem.**
  - **you want to layer your subsystems. Use a façade to define an entry point to each subsystem level.**
  - **you want to decouple classes of a subsystem from its clients and other subsystems, thereby promoting subsystem independence and portability.**

Façade Pattern (contd.):

- **Generic Structure:**



Façade Pattern (contd.):

- Participants and Collaborations: Façade, subsystem classes.

Clients communicate with the subsystem by sending requests to façade, which forwards them to the appropriate subsystem objects. Although the subsystem objects perform the actual work, the façade may have to do work of its own to translate its interface to subsystem interfaces.

Clients that use the façade do not have to access the subsystem objects directly..

Façade Pattern (contd.):

- **Consequences:**
  - it shields clients from subsystem components, thereby reducing the number of objects that clients deal with and making the subsystem easier to use.
  - it promotes weak coupling between the subsystem and its clients. This allows you to change the classes the comprise the subsystem without affecting the clients.
  - it does not prevent sophisticated clients form accessing the underlying classes
  - it reduces dependencies in large software systems.

Façade Pattern (contd.):

- Implementation (from Shalloway and Trot book):
  - Define a new class (or classes) that has the required interface
  - Have this new class use the existing system..

State Pattern

State Pattern (Behavioral Pattern):

- **Intent (GOF):** Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.

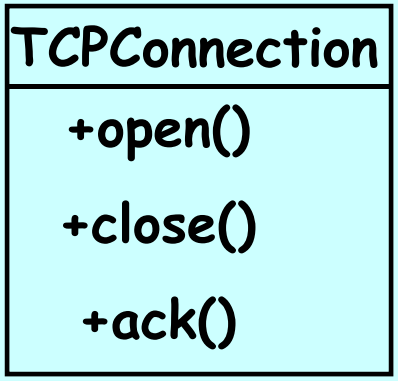
- **Motivation:** sometimes there is an object that can be in one of several states with different behavior in each state..

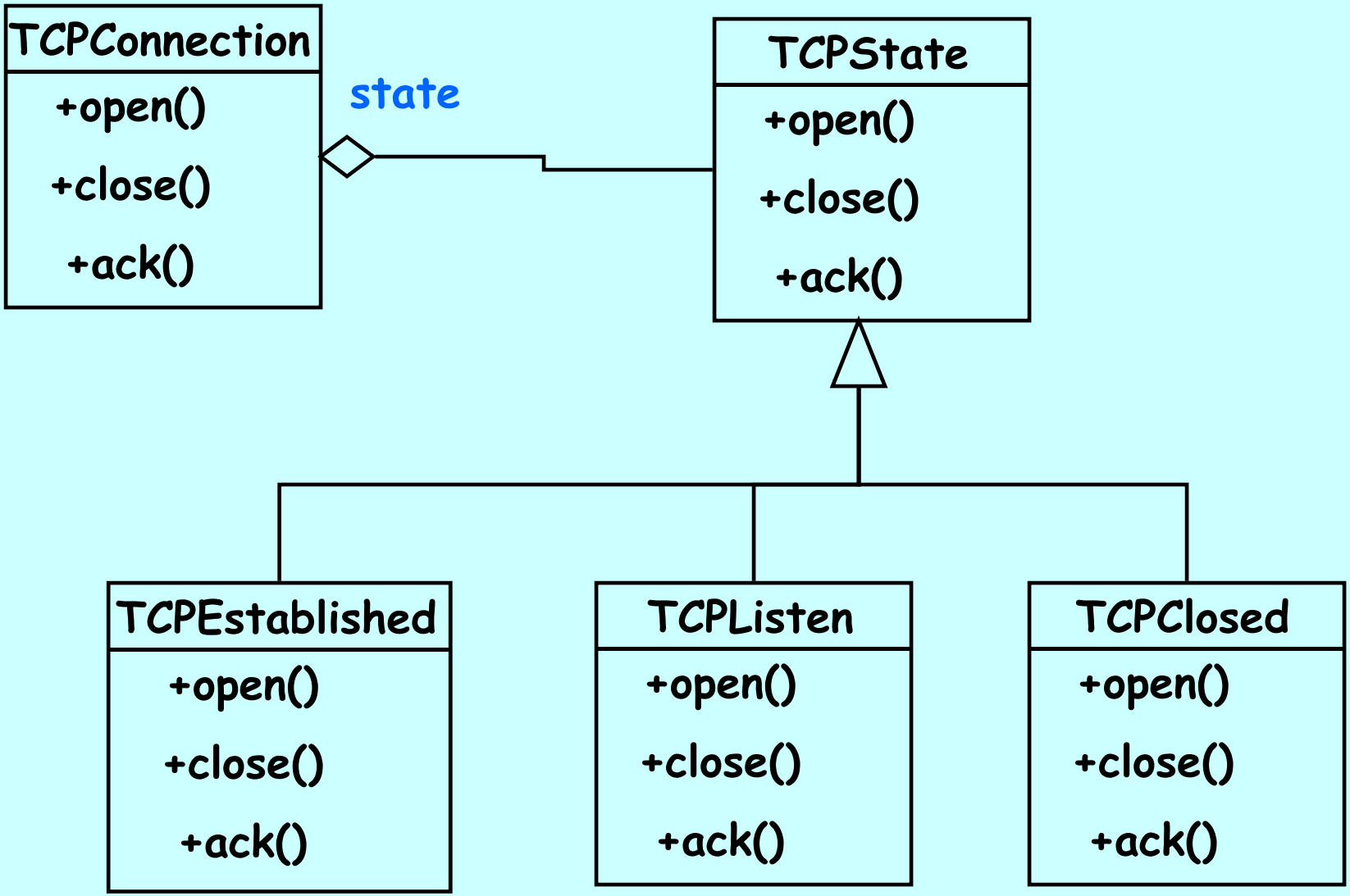
```
if( myState == idle )
    createGames();
else if ( myState == sick )
    sleep();
else if ….
```

State of an object: exact condition of an object in a given time, based on the values of their instance variables

Example (GOF):

Consider a class TCPConnection that represents a network connection. A TCPConnection object can be in one of several states: established, listening, closed. When it receives requests from other objects, it responds differently depending on its current state.




TCPConnection
+open()
+close()
+ack()
state
TCPState
+open()
+close()
+ack()
TCPEstablished
+open()
+close()
+ack()
TCPListen
+open()
+close()
+ack()
TCPClosed
+open()
+close()
+ack()

State Pattern (contd.):

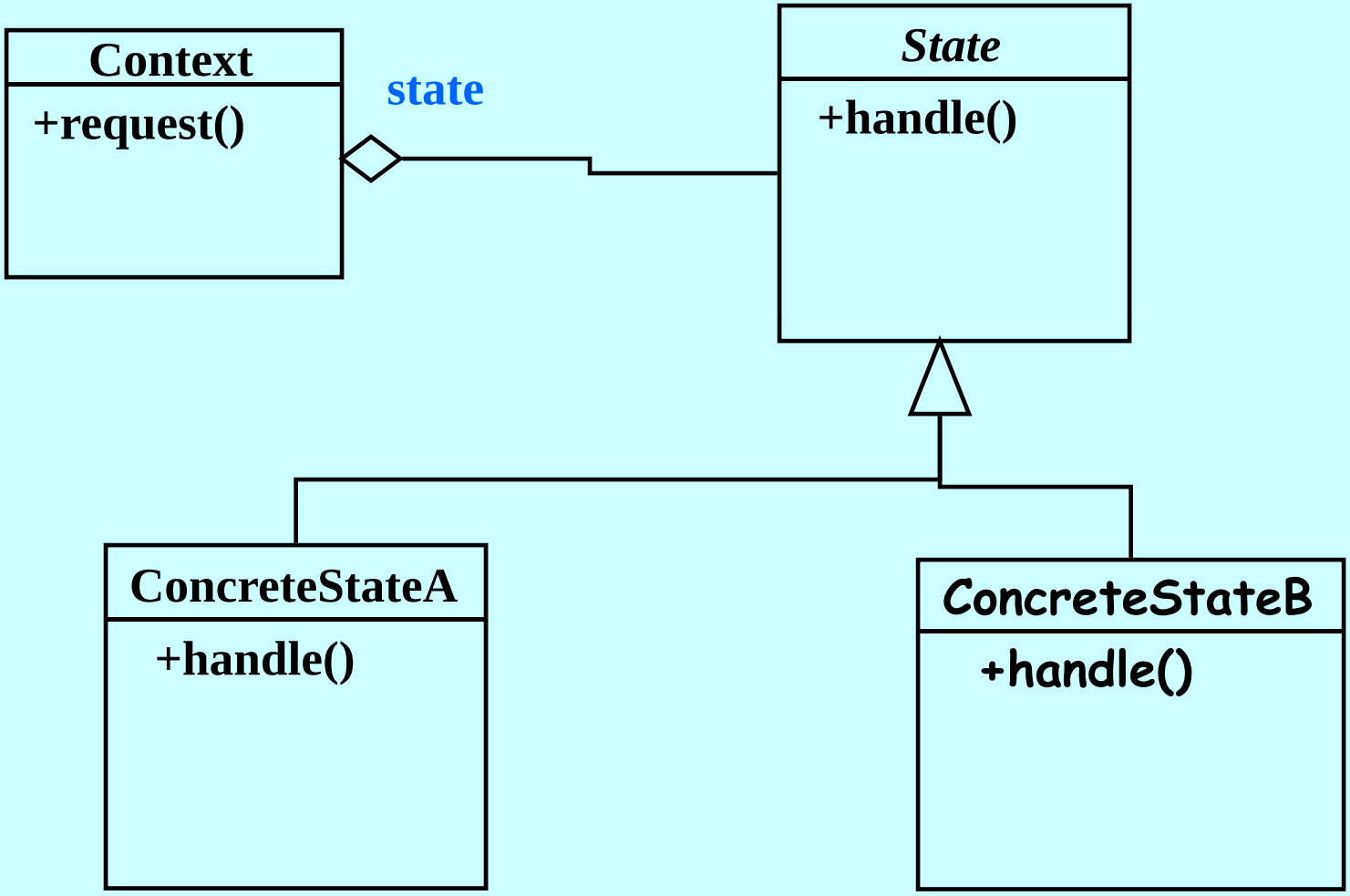
- **Applicability: Use it in following cases:**

  - **an object's behavior depends on its state, and it must change its behavior at run-time depending on that state**

  - **methods have large, multipart conditional statements that depend on the object's state.**

State Pattern (contd.):

- Generic Structure:



State Pattern (contd.):

- Participants and Collaborations: Context, State, ConcreteState subclasses.

- Context delegates state-specific request to the current ConcreteState object
- Context is the primary interface for clients.
- Either Context or ConcreteState subclasses can decide which state succeeds another..

State Pattern (contd.):

- **Consequences:**
  - new states and transitions can be added easily by defining new subclasses.
  - it localizes the behavior associated with a particular state into one object
  - allows state transition logic to be be incorporated into a state object rather than in a monolithic if or switch statement.
  - increased number of objects..

State Pattern (contd.):

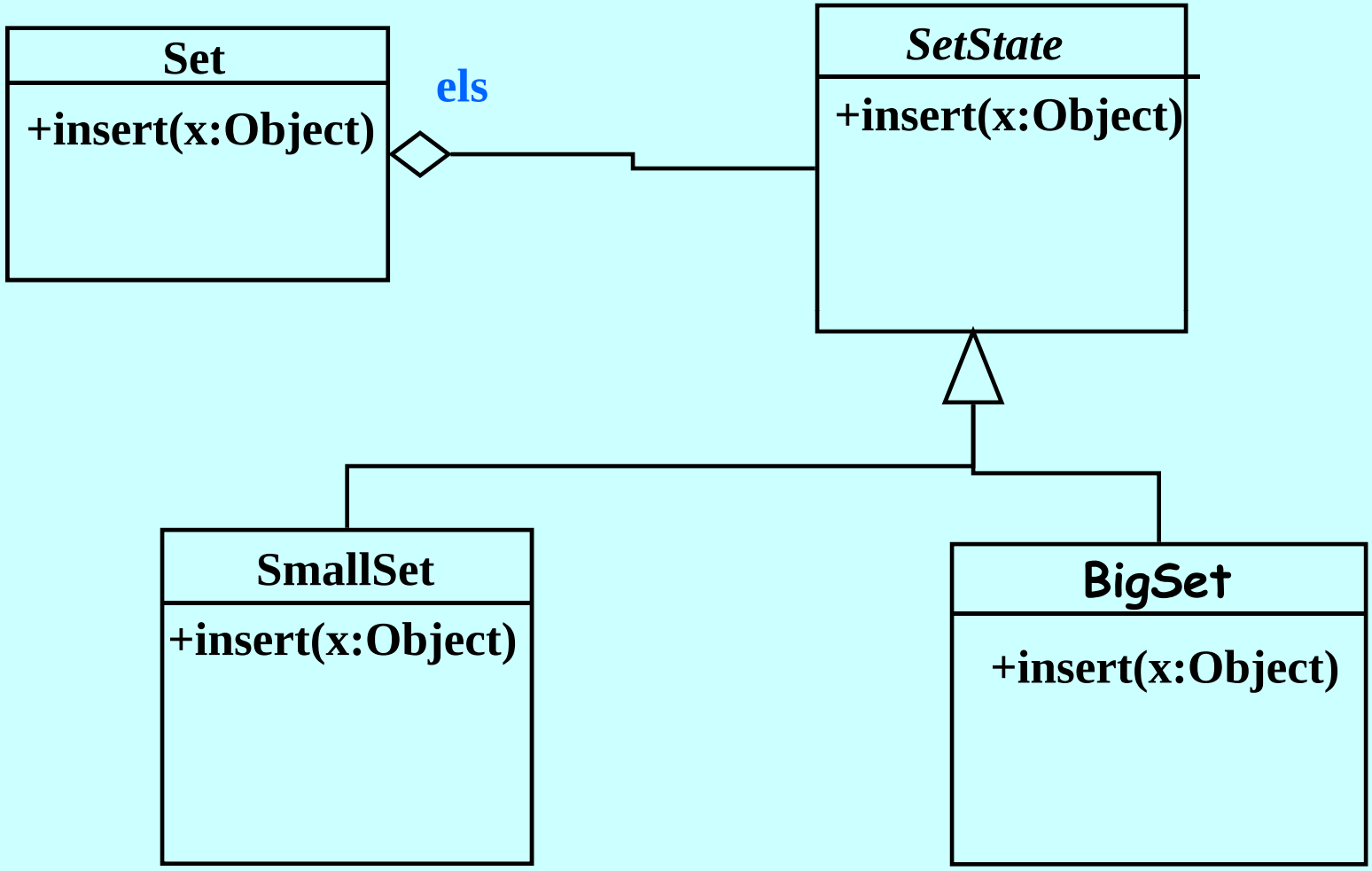
- Implementation: Set implementation using State pattern (Barbara and Liskov book)

```
public class Set {
   private SetState els;
   private int t1; // threshold for growing
   private int t2; //threshold for shrinking
   public Set() { els = new SmallSet(); }
   public void insert(Object x) {
      int sz = els.size(); els.insert(x);
      if(sz == t1 && els.size() > t1)
         els = new BigSet( (Vector) els );
   }
   //other methods go here…
}
```

```
public abstract class SetState {
    …
}

public class SmallSet extends SetState {
    …
}

public class BigSet extends SetState {
    …
}
```


Set
+insert(x:Object)
els
SetState
+insert(x:Object)
SmallSet
+insert(x:Object)
BigSet
+insert(x:Object)