**What does it do?**

Allow an object to alter its behavior when its internal state changes.The object will appear to change its class. It is also known as Objects for States.

Consider a class TCPConnection that represents a network connection. A TCPConnection object can be in one of several different states:Established, Listening, Closed. When a TCPConnection object receivesrequests from other objects, it responds differently depending on itscurrent state. For example, the effect of an Open request depends onwhether the connection is in its Closed state or its Establishedstate. The State pattern describes how TCPConnection can exhibitdifferent behavior in each state.

The key idea in this pattern is to introduce an abstract class calledTCPState to represent the states of the network connection. TheTCPState class declares an interface common to all classes thatrepresent different operational states. Subclasses of TCPStateimplement state-specific behavior. For example, the classesTCPEstablished and TCPClosed implement behavior particular to theEstablished and Closed states of TCPConnection.

The class TCPConnection maintains a state object (an instance of asubclass of

TCPState) that represents the current state of the TCPconnection. The class

TCPConnection delegates all state-specificrequests to this state object.

TCPConnection uses its TCPStatesubclass instance to perform operations particular

to the state of theconnection.

Whenever the connection changes state, the TCPConnection object changes the state object it uses. When the connection goes fromestablished to closed, for example, TCPConnection will replace itsTCPEstablished instance with a TCPClosed instance.

**Consider the following example:**

interface MobileAlertState

{

public void alert(AlertStateContext ctx);

}

class AlertStateContext

{

private MobileAlertState currentState;

public AlertStateContext()

{

currentState = new Vibration();

}

public void setState(MobileAlertState state)

{

currentState = state;

}

public void alert()

{

currentState.alert(this);

}

}

class Vibration implements MobileAlertState

{

@Override

public void alert(AlertStateContext ctx)

{

System.out.println("vibration...");

}

}

class Silent implements MobileAlertState

{

@Override

public void alert(AlertStateContext ctx)

{

System.out.println("silent...");

}

}

class StatePattern

{

public static void main(String[] args)

{

AlertStateContext stateContext = new AlertStateContext();

stateContext.alert();

stateContext.alert();

stateContext.setState(new Silent());

stateContext.alert();

stateContext.alert();

stateContext.alert();

}

}

Here, the alertStateContext maintains a state object. (We pass the interface reference to the alertStateContext’s method. Now, that interface can hold any objrct which implements it. Those objects will be state objext.