# STATE pattern

El estado de un objeto se puede definir como su condición exacta en cualquier momento, dependiendo de los valores de sus propiedades o atributos (un cambio en un atributo es un cambio de estado del objeto). El conjunto de métodos implementados por una clase constituye el comportamiento de sus ejemplares.

El patrón de estado es útil para diseñar una estructura eficiente para una clase, un ejemplar que puede existir en muchos estados diferentes y exhibir un comportamiento diferente según el estado en el que se encuentra.

* En otras palabras, en el caso de un objeto de tal clase, parte o todo su comportamiento está completamente influenciado por su estado actual. En la terminología de patrones de diseño de estado, dicha clase se denomina una clase de contexto.

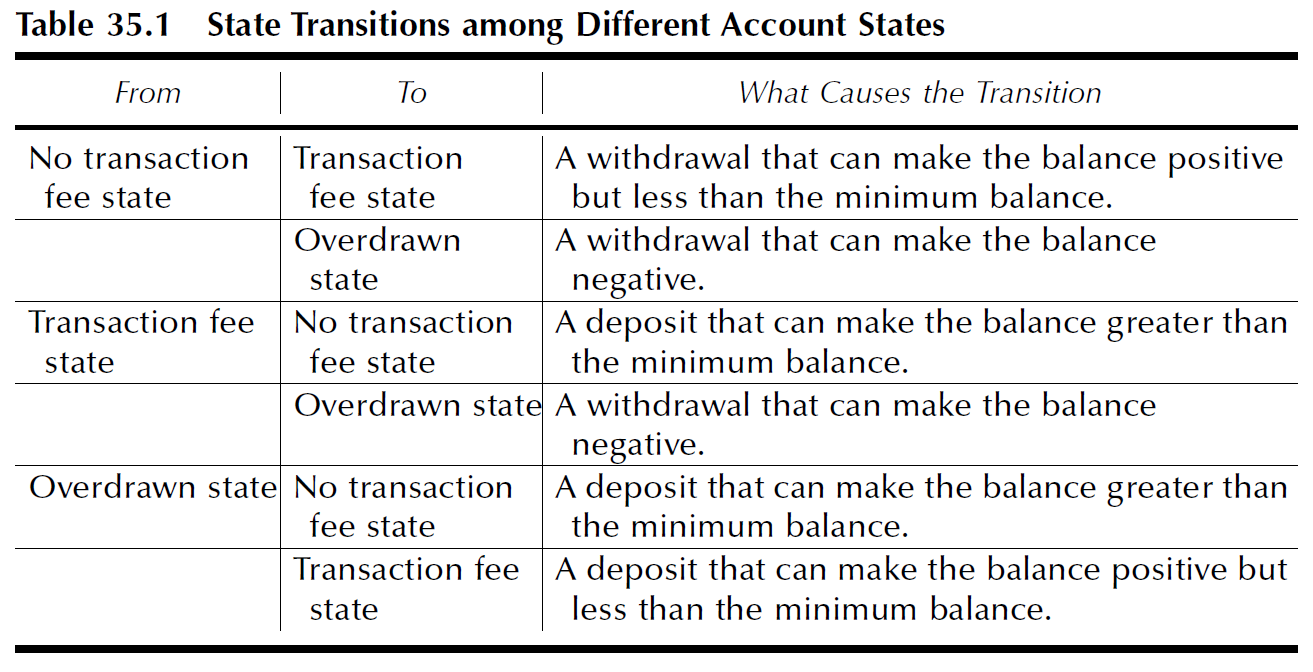
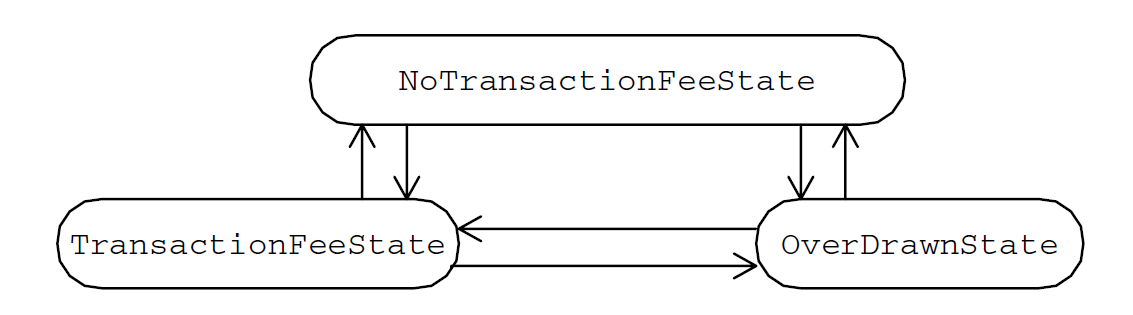
Un objeto de contexto puede cambiar su comportamiento cuando hay un cambio en su estado interno y también se le conoce como **objeto Stateful**.

## Objeto Stateful

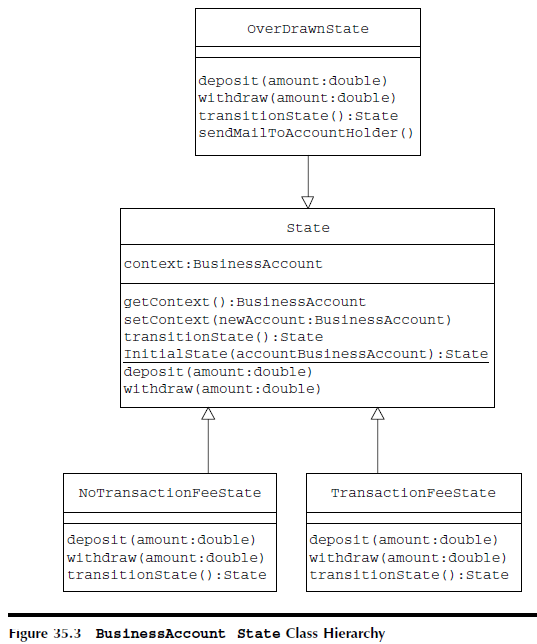
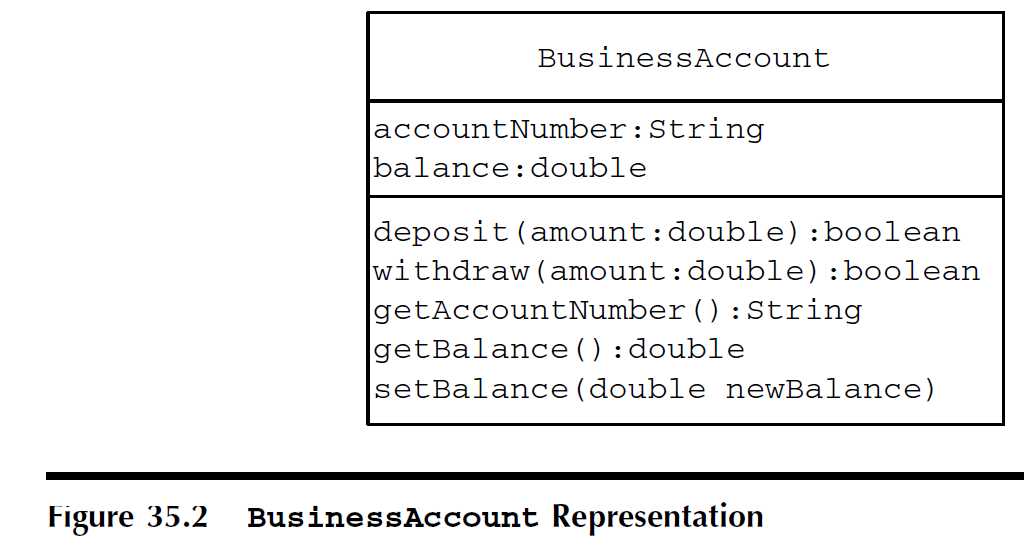
La mayoría de los editores HTML disponibles en la actualidad ofrecen diferentes vistas de una página HTML en el momento de su creación. Consideremos uno de esos editores que ofrece tres vistas de una página web determinada de la siguiente manera:

1. Vista de diseño: en esta vista, un usuario puede crear visualmente una página web sin tener que conocer los comandos HTML internos.
2. Vista HTML: esta vista ofrece al usuario la estructura básica de la página web en términos de las etiquetas HTML y le permite personalizar la página web con código HTML adicional.
3. Vista de página rápida: esta vista proporciona una vista previa de la página web que se está creando.

En los tres estados, no se permite una transacción de retiro que exceda el límite de sobregiro. La figura 35.1 muestra las posibles transiciones de estado para una cuenta y la tabla 35.1 muestra cómo pueden ocurrir estas transiciones.



Diseñemos una representación para la cuenta comercial en forma de la clase BusinessAccount como en la Figura 35.2 y el Listado 35.1 La clase BusinessAccount ofrece la funcionalidad básica en forma de métodos para permitir que un objeto cliente realice operaciones de depósito y retiro. La clase BusinessAccount define algunos de los límites de transacciones y ofrece métodos de acceso para leer su estado.

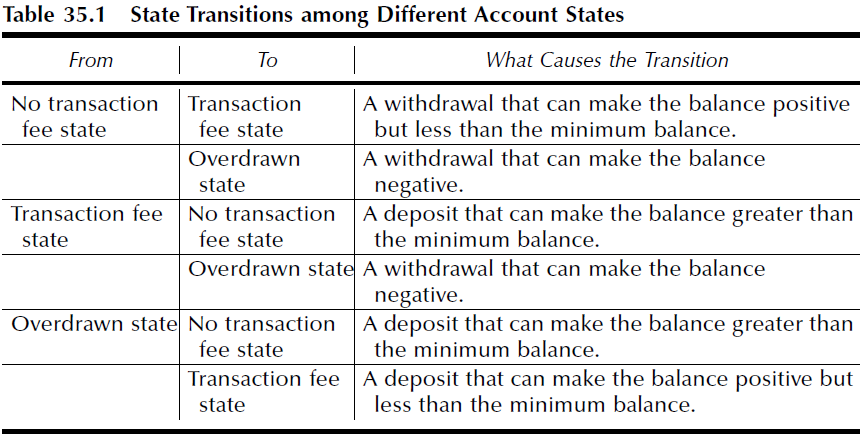


Definamos una clase State común (Listado 35.2) que contiene el comportamiento de la cuenta de negocio que es común en todos los estados. En lugar de mantener el comportamiento específico del estado dentro de la clase BusinessAccount, aplicando el patrón de estado, el comportamiento específico de cada uno de los tres estados puede implementarse en forma de tres subclases separadas:

* NoTransactionFeeState, TransactionFeeState y OverDrawnState

La figura 35.3 muestra la jerarquía de clases resultante. La clase State padre común declara que la interfaz será utilizada por diferentes objetos de cliente para acceder a los servicios de los objetos en la jerarquía de clases de estado. Si un objeto cliente está diseñado para utilizar los servicios de un objeto del tipo de clase State padre común, puede acceder a los servicios ofrecidos por sus subclases de manera transparente.

Además, estas subclases conocen el estado al que debe hacer la transición y cuándo hacer esa transición.



Cada una de estas subclases de estado implementa esta funcionalidad de transición de estado sobreescribiendo el método transitionState de la clase principal según las reglas de transición de estado detalladas en la Tabla 35.1. Si bien el comportamiento específico del estado se separa de BusinessAccount, el estado (es decir, el saldo de la cuenta) aún se mantiene dentro de la clase BusinessAccount. Debido a que el comportamiento contenido en cada uno de los Stateobjects es específico de un estado de la cuenta comercial representada por BusinessAccountclass, un objeto State debería poder leer el estado de BusinessAccountobject. Para facilitar esto, cada uno de los objetos de estado está diseñado para contener una referencia de objeto del tipo BusinessAccount.

Cuando se crea un objeto de estado, se configura con una instancia de BusinessAccount. Con este objeto BusinessAccount, un objeto de estado puede verificar o alterar el estado de la cuenta comercial que representa. Debido a que el comportamiento específico del estado de una cuenta comercial está contenido en la jerarquía de clases State, BusinessAccount necesita una forma de acceder al comportamiento específico de su estado actual. Este requisito se puede abordar mejorando el diseño de la clase BusinessAccount para que BusinessAccountobject mantenga una variable de referencia al objeto de tipo State para almacenar su objeto de estado actual.

Cuando se crea un objeto BusinessAccount por primera vez, establece una instancia de la clase NoTransactionFeeState (el estado predeterminado) como su objeto currentState. Siempre que un objeto cliente invoca un método como depositar o retirar en el objeto BusinessAccount, reenvía la llamada al método a su objeto de estado actual. La figura 35.4 y el Listado 35.6 muestran la representación revisada de la clase Business-Account. La clase BusinessAccount representa la cuenta comercial y actúa como contexto en este ejemplo. La figura 35.5 muestra la asociación de clases general. Diseñemos un AccountManager cliente de prueba para permitir que un usuario realice diferentes transacciones en una cuenta comercial. Cuando se ejecuta, AccountManager:

* Crea un objeto BusinessAccount que representa una cuenta comercial.
* Muestra la interfaz de usuario necesaria como en la Figura 35.6 para permitir que un usuario realice transacciones de depósito y retiro que pueden hacer que la cuenta comercial pase por diferentes estados.

Cada transacción de depósito o retiro iniciada a través de la interfaz de usuario se traduce en una llamada al método de depósito (doble) o retiro (doble) en el objeto BusinessAccount que se crea cuando se ejecuta el AccountManager.

* El objeto BusinessAccount a su vez reenvía esta llamada a su objeto State actual interno.
* El objeto State actual ejecuta el comportamiento que contiene y establece un objeto State apropiado como el objeto State actual del objeto BusinessAccount. De esta manera, la clase Context (BusinessAccount) y su comportamiento específico de estado (jerarquía de clases State) están completamente separados entre sí.
* Cuando se agrega un nuevo comportamiento específico de un estado o se modifica el comportamiento específico de un estado, la clase Context actual BusinessAccount no se ve afectada.