State

Propósito

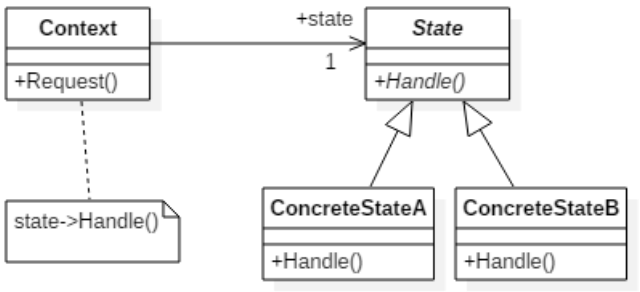
Permite a un objeto alterar su comportamiento cuando cambia su estado interno. El objeto parecerá cambiar de clase.

Aplicabilidad

El patrón State se utiliza cuando:

- El comportamiento de un objeto depende de su estado y debe cambiar en tiempo de ejecución
- Tenemos operaciones con muchas sentencias condicionales que dependen del estado del objeto. Este estado se representa normalmente como una o más valores constantes enumerados. Con frecuencia, varias operaciones contienen la misma estructura condicional. El patrón State coloca cada ramal de la estructura condicional en una clase separada. Esto nos permite trata el estado del objeto como un objeto, de modo que puede variar de forma independiente.

Estructura



Participantes

- Context: define la interface para los clientes
    - Mantiene una instancia a un ConcreteState que define el estado actual
- State: define una interface que encapsula el comportamiento asociado con un estado particular de Context
- ConcreteState: cada subclase implementa un comportamiento asociado con un estado de Context

Colaboraciones

- Context delega las peticiones específicas del estado al objeto ConcreteState actual
- Un contexto puede pasarse a sí mismo como argumento al objeto State que se encarga de la petición. Esto permite que el objeto State acceda al contexto si le hace falta
- Context es la interface principal para los clientes. Los clientes pueden configurar un contexto con un objeto State. Una vez se ha configurado un contexto, los clientes no tratan directamente con los objetos State

- Tanto Context como las subclases ConcreteState pueden decidir qué estado sucede a otro y bajo qué circunstancias

Consecuencias

El patrón State tiene las siguientes consecuencias:

1. Localiza el comportamiento específico del estado y lo divide en diferentes estados. El patrón State coloca todo el comportamiento asociado con un estado particular en un objeto. Como todo el código específico de un estado vive en una subclase State, se pueden agregar fácilmente nuevas subclases y transiciones definiendo nuevas subclases.
   Una alternativa es utilizar valores para definir los estados internos y tener operaciones en Context que comprueben los datos explícitamente. Pero entonces tendremos sentencias condicionales repartidas allí donde se implemente Context. Agregar un nuevo estado puede suponer cambiar varias sentencias, lo que complica el mantenimiento.
   El patrón State evita este problema, pero puede introducir otro, ya que distribuye el comportamiento en diferentes objetos State. Esto incrementa el número de clases y resulta menos compacto que una sola clase. Esta distribución es buena si hay muchos estados, ya que de otro modo tendríamos sentencias condicionales grandes.
   Como los procedimientos grandes, las sentencias condicionales grandes son indeseables. Son monolíticas y tienden a hacer el código menos explícito, lo cual resulta en un código difícil de mantener y extender. El patrón State ofrece un modo mejor de estructurar el código específico del estado. La lógica que determina las transiciones no reside en sentencias monolíticas if o switch sino que se particiona entre subclases State. Encapsular cada transición de estado y acción en una clase eleva la idea de estado de ejecución a estado del objeto. Esto impone estructura en el código y hace más clara la intención.
2. Hace explícitas las transiciones entre estados. Cuando un objeto define su estado actual únicamente en términos de valores internos, las transiciones de estado no se representan de forma explícita; sólo se muestran como asignaciones a algunas variables. Introducir objetos separados para los diferentes estados hace las transiciones más explícitas. Además, los objetos State pueden proteger a Context de estados inconsistentes ya que las transiciones de estados son atómicas desde la perspectiva de Context, ya que ocurren con la asignación de una variable en vez de varias.
3. Los objetos State se pueden compartir. Si los objetos State no tienen variables de instancia -esto es, los estados que representan están codificados por su tipo-, entonces los contextos pueden compartir los objetos State. Cuando se comparten los State de esta forma, realmente se comportan según el patrón Flyweight sin estado intrínseco.