|  |  |  |  |
| --- | --- | --- | --- |
| NOMBRE | Programacion Avanzada | | |
| AREA | Software | REGIMEN | Semestral |

**Tema: Patrones de diseño**

**Estado**

Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno. Busca que un objeto pueda reaccionar según su estado interno. Si bien muchas veces esto se puede solucionar con un boolean o utilizando constantes, esto suele terminar con una gran cantidad de if-else, código ilegible y dificultad en el mantenimiento. La intención del State es desacoplar el estado de la clase en cuestión.

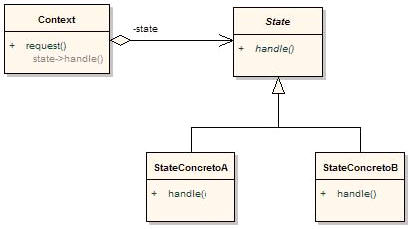
En determinadas ocasiones se requiere que un objeto tenga diferentes comportamientos según el estado en que se encuentra. Esto resulta complicado de manejar, sobre todo cuando se debe tener en cuenta el cambio de comportamientos y estados de dicho objeto, todos dentro del mismo bloque de código. El patrón State propone una solución a esta complicación, creando un objeto por cada estado posible.

Este patrón debe ser utilizado cuando:

* El comportamiento de un objeto depende de un estado, y debe cambiar en tiempo de ejecución según el comportamiento del estado.
* Cuando las operaciones tienen largas sentencias con múltiples ramas que depende del estado del objeto.
* Este patrón se utiliza cuando un determinado objeto tiene diferentes estados y también distintas responsabilidades según el estado en que se encuentre en determinado instante. También puede utilizarse para simplificar casos en los que se tiene un complicado y extenso código de decisión que depende del estado del objeto.

**Diagrama UML**

**Vista estática: diagrama de clases**



Context: mantiene una instancia con el estado actual

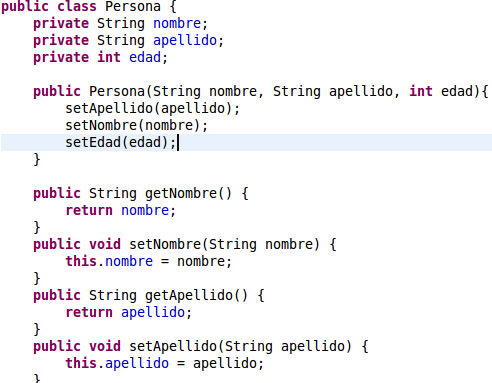
State: define interfaz para el comportamiento asociado a un determinado estado del Contexto.

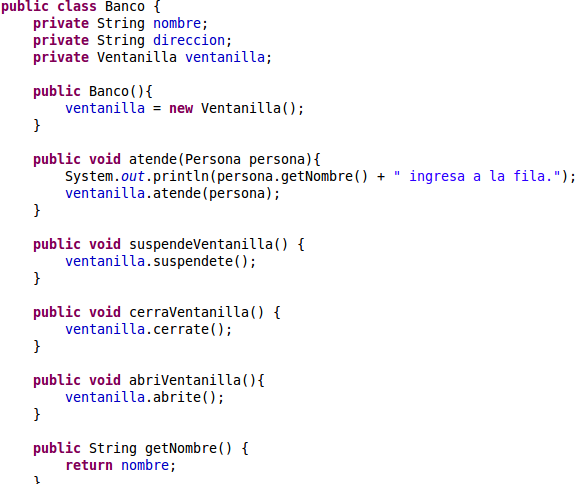
StateConcreto: cada subclase implementa el comportamiento asociado con un estado del contexto.

El Context delega el estado específico al objeto StateConcreto actual. Un objeto Context puede pasarse a sí mismo como parámetro hacia un objeto State. De esta manera la clase State puede acceder al contexto si fuese necesario. Context es la interfaz principal para el cliente. El cliente puede configurar un contexto con los objetos State. Una vez hecho esto estos clientes no tendrán que tratar con los objetos State directamente. Tanto el objeto Context como los objetos de StateConcreto pueden decidir el cambio de estado.

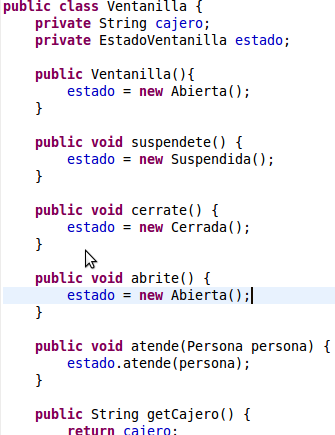
***Caso práctico a desarrollar en clase***

Imaginemos que vamos a un banco y cuando llegamos nos colocamos en la fila de mostrador: si la misma está abierta, seguiremos en la fila. En cambio, si está cerrada nos colocaremos en otra fila o tomaremos alguna decisión acorde. Por otro lado, si vemos un cartel que dice "enseguida vuelvo" quizás tenemos que contemplar el tiempo disponible que tenemos. Es decir, para nosotros, el comportamiento de un banco cambia radicalmente según el estado en el que se encuentre. Para estas ocasiones, es ideal el uso de un patrón de estados.

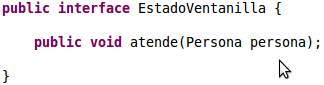


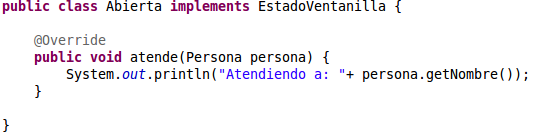


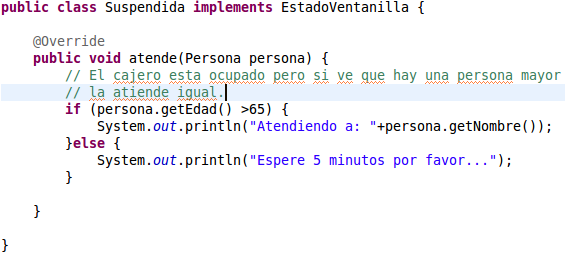
El banco publica un método llamado atende () pero en realidad la atención la realiza la ventanilla.

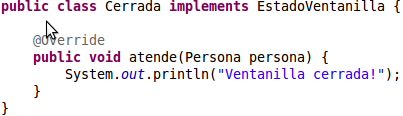


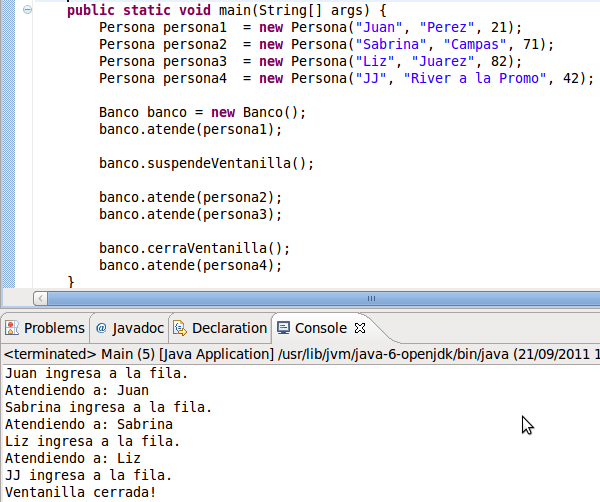
La ventanilla cambia su comportamiento según el estado en que se encuentre. Por ejemplo, si está cerrada, no hay atención directamente. Por eso mismo, delega el método de atención a su estado y es este mismo estado quién toma la decisión de atender o no.











**Consecuencias**

Se localizan fácilmente las responsabilidades de los estados concretos, dado que se encuentran en las clases que corresponden a cada estado. Esto brinda una mayor claridad en el desarrollo y el mantenimiento posterior. Esta facilidad la brinda el hecho que los diferentes estados están representados por un único atributo (state) y no envueltos en diferentes variables y grandes condicionales.

Hace los cambios de estado explícitos puesto que en otros tipos de implementación los estados se cambian modificando valores en variables, mientras que aquí al estar representado cada estado.

Facilita la ampliación de estados mediante una simple herencia, sin afectar al Context.

Permite a un objeto cambiar de estado en tiempo de ejecución.

Los estados pueden reutilizarse: varios Context pueden utilizar los mismos estados.

Se incrementa el número de subclases.