Design pattern State

#1 Consente di gestire il comportamento di un oggetto in base al suo stato INTERNO. L'oggetto cambia il suo comportamento in base allo stato in cui si trova.

#2 Utile quando si hanno oggetti che devono eseguire comportamenti diversi a seconda del loro stato interno, e permette di semplificare la gestione dei comportamenti complessi dell'oggetto e di aggiungere nuovi stati in modo semplice e MODULARE

Il pattern State è un design pattern comportamentale che consente di gestire il comportamento di un oggetto a seconda del suo stato interno. In questo pattern, l'oggetto cambia il suo comportamento a seconda dello stato in cui si trova, senza che il client debba conoscere l'implementazione specifica dello stato.

Il pattern State è utile in situazioni in cui un oggetto può avere diversi comportamenti a seconda del suo stato interno e questi comportamenti devono essere gestiti in modo pulito e mantenibile. Ad esempio, il pattern State può essere utilizzato per gestire gli stati di una macchina a stati finiti, la gestione dei diversi tipi di utenti in un'applicazione web o la gestione dei diversi tipi di file in un gestore di file.

Intento → lo STATE consente all'oggetto di MODIFICARE il proprio COMPORTAMENTO quando il suo stato interno cambia.

Lo stato è un modello di progettazione comportamentale che consente a un oggetto di modificare il proprio comportamento quando il suo stato interno cambia. Sembra che l'oggetto abbia cambiato classe.

Strettamente correlato al CONCETTO di macchina a STATI FINITI. Il nostro programma in un certo istante di tempo si può trovare in un certo NUMERO di STATI FINITI.

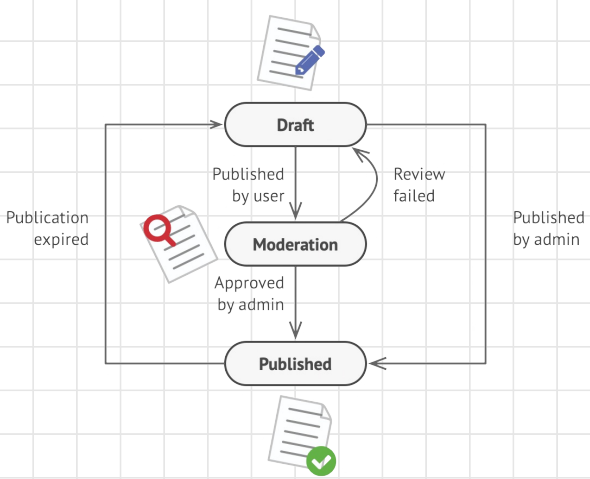
Problema All'interno di ogni STATO il programma si comporta in MODO DIVERSO. Il programma può passare da uno stato all'altro istantaneamente.

Il modello di stato è strettamente correlato al concetto di macchina a stati finiti . L'idea principale è che, in un dato momento, c'è un numero finito di stati in cui un programma può trovarsi. All'interno di ogni stato univoco, il programma si comporta in modo diverso e il programma può passare da uno stato all'altro istantaneamente. Tuttavia, a seconda dello stato corrente, il programma può passare o meno a determinati altri stati. Anche queste regole di commutazione, chiamate transizioni , sono finite e predeterminate.

Puoi anche applicare questo approccio agli oggetti. Immagina di avere una classe Document. Un documento può trovarsi in uno dei seguenti tre stati : Draft(bozza), Moderation(moderazione) e Published(publicato).

Il metodo publish() del documento funziona in modo differente in ogni stato :

- In Draft sposta il documento alla moderazione
- In Moderation, rende publico il documento, ma solo se l'utente corrente è un amministratore
- In Published non fa assolutamente niente

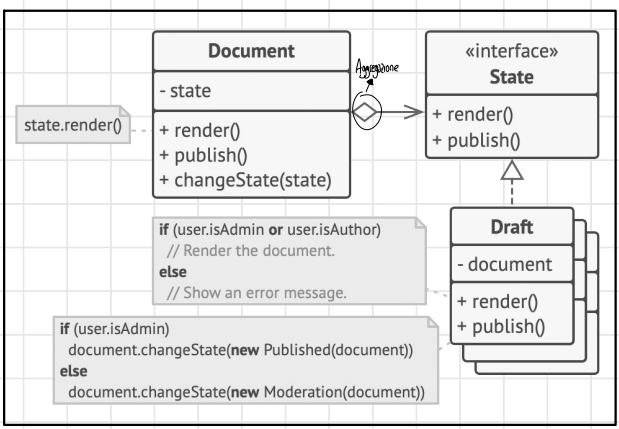


Le macchine a stati finiti sono generalmente implementate con molte istruzioni condizionali del tipo if, switch ecc.. che selezionano il comportamento appropriato in base allo stato corrente dell'oggetto. Di solito, questo "stato" è solo un insieme di valori nei campi dell'oggetto. La più grande debolezza di una macchina a stati basata sui condizionali si rivela quando iniziamo ad aggiungere sempre più stati e comportamenti dipendenti dallo stato alla classe Document. La maggior parte dei metodi conterrà condizionali mostruosi che selezionano il comportamento corretto di un metodo in base allo stato corrente. Il codice come questo è molto difficile da mantenere perché qualsiasi modifica alla logica di transizione può richiedere la modifica dei condizionali di stato in ogni metodo.
Il problema tende a diventare più grande man mano che un progetto si evolve. È piuttosto difficile prevedere tutti i possibili stati e le transizioni in fase di progettazione. Quindi, una macchina a stati snella costruita con un insieme limitato di condizionali può trasformarsi in un pasticcio gonfio nel tempo.

- Siccome dobbiamo valutare diversi stati dell'oggetto (anziché appesantire il codice, rendendolo complesso)

Soluzione costruiamo una classe per ogni stato dell'oggetto ed estraiamo i comportamenti dello stato in queste classi. Invece di implementare tutti i comportamenti da solo memorizza un riferimento a uno degli oggetti stato (che rappresentano lo stato corrente dell'oggetto) e tutto il lavoro sarà svolto da loro.

Il modello State suggerisce di creare nuove classi per tutti i possibili stati di un oggetto ed estrarre tutti i comportamenti specifici dello stato in queste classi. Invece di implementare tutti i comportamenti da solo, l'oggetto originale, chiamato context , memorizza un riferimento a uno degli oggetti stato che rappresenta il suo stato corrente e delega tutto il lavoro relativo allo stato a quell'oggetto.

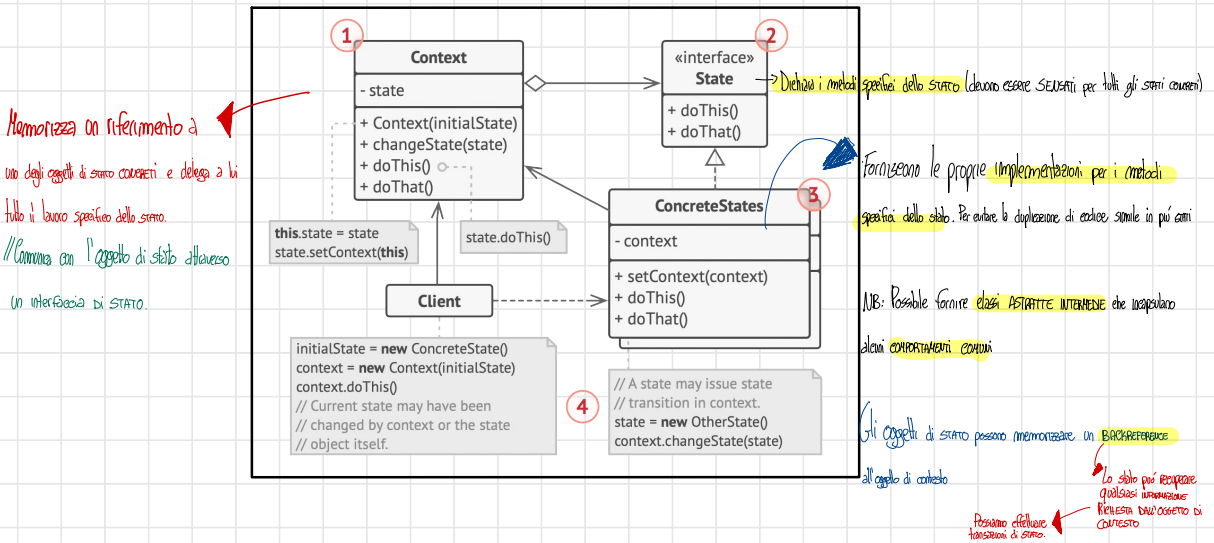


Per far passare il contesto in un altro stato, sostituisci l'oggetto stato attivo con un altro oggetto che rappresenta quel nuovo stato. Questo è possibile solo se tutte le classi di stato seguono la stessa interfaccia e il contesto stesso funziona con questi oggetti attraverso quell'interfaccia.

Esempio pratico

Un esempio semplice del pattern State potrebbe essere un'applicazione per la gestione di un ordine online. L'ordine può trovarsi in diversi stati, come "in elaborazione", "in attesa di pagamento", "in spedizione" o "consegnato".
Per gestire questi diversi stati, potremmo utilizzare il pattern State. In particolare, potremmo creare un oggetto di stato per ogni stato dell'ordine e definire le azioni che possono essere eseguite in quel particolare stato. Ad esempio, l'oggetto di stato "in elaborazione" potrebbe consentire solo l'aggiunta di nuovi prodotti all'ordine, mentre l'oggetto di stato "in attesa di pagamento" potrebbe consentire solo il pagamento dell'ordine.
Il contesto sarebbe rappresentato dall'oggetto dell'ordine in sé, che cambierebbe il suo comportamento a seconda dello stato in cui si trova. L'interfaccia di stato definirebbe il contratto tra il contesto (l'ordine) e l'oggetto di stato, consentendo al contesto di interagire con l'oggetto di stato senza conoscere l'implementazione specifica dello stato.
In questo modo, il pattern State ci consente di gestire in modo efficiente gli stati di un ordine online, migliorando la flessibilità e la manutenibilità del codice.

Struttura



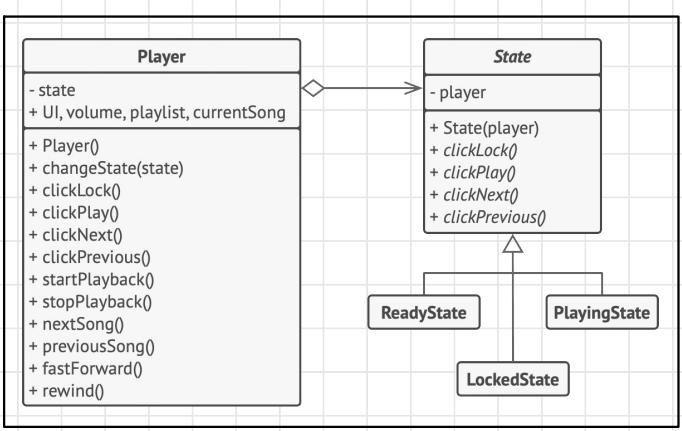
1 Il **Context** memorizza un riferimento a uno degli oggetti di stato concreti e delega ad esso tutto il lavoro specifico dello stato. Il contesto comunica con l'oggetto di stato tramite l'interfaccia di stato. Il contesto espone un setter per passargli un nuovo oggetto di stato.

2 L' interfaccia **State** dichiara i metodi specifici dello stato. Questi metodi dovrebbero avere senso per tutti gli stati concreti perché non vuoi che alcuni dei tuoi stati abbiano metodi inutili che non verranno mai chiamati.

3 I **concrete states** forniscono le proprie implementazioni per i metodi specifici dello stato. Per evitare la duplicazione di codice simile in più stati, è possibile fornire classi astratte intermedie che incapsulano alcuni comportamenti comuni.
Gli oggetti di stato possono memorizzare un backreference all'oggetto di contesto. Attraverso questo riferimento, lo stato può recuperare qualsiasi informazione richiesta dall'oggetto di contesto, nonché avviare transizioni di stato.

Sia il contesto che gli stati concreti possono impostare lo stato successivo del contesto ed eseguire la transizione di stato effettiva sostituendo l'oggetto stato collegato al contesto.

Altro esempio :



Lo utilizziamo quando si dispone di un oggetto che assume diversi COMPORTAMENTI A SECONDA DEL SUO STATO CORRENTE.
Quando abbiamo una classe INQUINATA DA ENORMI CONDIZIONALI.
Quando ABBIAMO CODICE DUPLICATO TRA STATI E TRANSIZIONI

Applicabilità

Utilizzare il modello di State quando si dispone di un oggetto che si comporta in modo diverso a seconda del suo stato corrente, il numero di stati è enorme e il codice specifico dello stato cambia frequentemente.

- Il modello suggerisce di estrarre tutto il codice specifico dello stato in un insieme di classi distinte. Di conseguenza, è possibile aggiungere nuovi stati o modificare quelli esistenti indipendentemente l'uno dall'altro, riducendo i costi di manutenzione.

Usa il modello quando hai una classe inquinata da enormi condizionali che alterano il modo in cui la classe si comporta in base ai valori correnti dei campi della classe.

- Il modello State consente di estrarre i rami di questi condizionali nei metodi delle classi di stato corrispondenti. Mentre lo fai, puoi anche pulire i campi temporanei e i metodi di supporto coinvolti nel codice specifico dello stato fuori dalla tua classe principale.

Usa State quando hai molto codice duplicato tra stati e transizioni simili di una macchina a stati basata sulle condizioni.

- Il modello State consente di comporre gerarchie di classi di stato e ridurre la duplicazione estraendo il codice comune in classi di base astratte.

Conseguenze sul codice :

PRINCIPIO RESPONSABILITÀ UNICA . Open/Close ; Classi che gestiscono determinati STATI
INTRODUCIAMO NUOVI STATI SENZA MODIFICARE IL CODICE.
L'APPLICAZIONE del modello può essere eccessiva se gli STATI sono pochi e CAMBIANO RARAMENTE.

Vantaggi

Principio di responsabilità unica . Organizzare il codice relativo a stati particolari in classi separate.

Principio aperto/chiuso . Introduci nuovi stati senza modificare le classi di stato esistenti o il contesto.

Semplifica il codice del contesto eliminando gli ingombranti condizionali della macchina a stati.

Svantaggi

- L'applicazione del modello può essere eccessiva se una macchina a stati ha solo pochi stati o cambia raramente.

Diagramma UML degli stati

Il diagramma UML degli stati (State Diagram) è un tipo di diagramma utilizzato nella notazione Unified Modeling Language (UML) per rappresentare il comportamento di un sistema o di un componente del sistema attraverso il cambiamento degli stati e le transizioni tra di essi. Questo diagramma mette in evidenza come un oggetto può passare da uno stato all'altro, considerando gli eventi che causano tali transizioni e le condizioni che devono essere soddisfatte per queste transizioni.

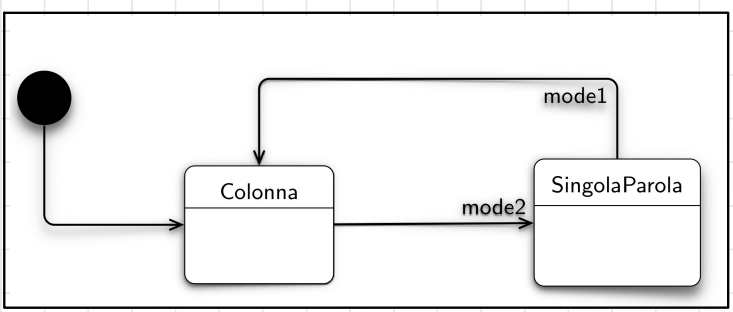
Un diagramma degli stati UML è composto dai seguenti elementi principali:

1. **Stati** : Rappresentano le diverse condizioni in cui può trovarsi un oggetto. Gli stati sono solitamente rappresentati da rettangoli arrotondati con il nome dello stato all'interno (Colonna e SingolaParola).

2. **Transizioni**: Sono le linee con le frecce che collegano gli stati e rappresentano il passaggio da uno stato all'altro. Le transizioni possono avere un'etichetta che indica l'evento o la condizione che provoca il passaggio tra gli stati (mode1 e mode2).

3. **Eventi**: Sono gli stimoli che causano il passaggio da uno stato all'altro. Gli eventi possono essere azioni dell'utente, input esterni o eventi temporizzati.

4. **Azioni**: Sono le attività che vengono eseguite quando un oggetto entra o esce da uno stato. Le azioni possono essere semplici o composte e sono solitamente associate a uno stato o a una transizione.



Nel diagramma UML degli stati, lo spazio bianco all'interno della rappresentazione di uno stato indica l'attività che l'oggetto sta eseguendo mentre si trova nello stato corrispondente. Se lo spazio bianco è vuoto, significa che non ci sono azioni specifiche che l'oggetto deve eseguire durante la permanenza in quel particolare stato, ma lo stato può comunque rappresentare una condizione o una situazione in cui l'oggetto si trova.