State & ParallelStream

Alessandro Midolo, Ph.D. Student
alessandro.midolo@phd.unict.it

Tutorato Ingegneria del Software

A.A. 2021/2022

State

Intento

- Permettere ad un oggetto di modificare il suo comportamento quando il suo stato interno cambia

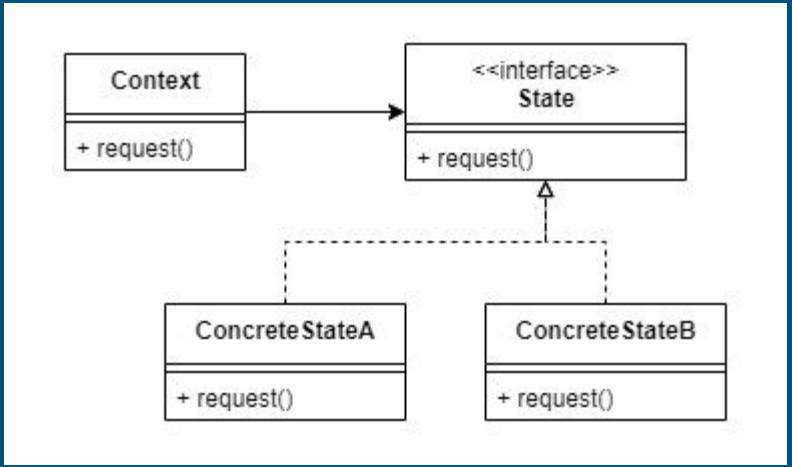
Problema

- Il comportamento di un oggetto deve cambiare a run-time in base al suo stato attuale
- Le operazioni da svolgere hanno vari grandi rami condizionali che dipendono dallo stato
- Lo stato può essere rappresentato da una o più variabili enumerative costanti

Soluzione

- **Context**: l'interfaccia visibile dal client, mantiene un'istanza del *ConcreteState* per definire lo stato corrente
- **State**: definisce l'interfaccia per incapsulare il comportamento di uno specifico stato del *Context*
- **ConcreteState**: sono le sottoclassi che implementato ciascun comportamento associato ad uno stato del *Context*

Diagramma UML State



Conseguenze State

- Il comportamento associato ad uno stato è gestito da una singola classe **ConcreteState**. E’ quindi possibile aggiungere nuovi stati e transizioni facilmente, basta creare delle nuove sottoclassi. Questo rende il codice più strutturato e rende più chiaro lo scopo del codice
- La gestione delle transizioni tra stati è gestita separatamente nel *Context*, invece di utilizzare costrutti vari (if, switch) sulla classe che implementa i comportamenti. Questo rende la gestione dei cambiamenti di stato più consistente
- Il numero di classi è maggiore, ma queste risultano essere più semplici

Programmazione Parallela

La programmazione parallela permette di eseguire codice in modo parallelo, incrementando le performance dell'applicativo → E' più difficile di quella sequenziale

In Java la classe ***Thread*** mette a disposizione delle API per l'esecuzione e la gestione di nuovi Thread. Bisogna però attenzionare i problemi di "corsa critica" (Race condition), bisogna infatti saper utilizzare *synchronized*, *wait* e *notify*

Da Java 5 è possibile utilizzare diverse librerie come Lock, Esecutori etc.

ParallelStream

Le operazioni ***filter*** e ***map*** sono operazioni stateless, ovvero non tengono uno stato durante l'esecuzione → lo stream non viene modificato da queste operazioni, poiché viene generato un nuovo stream

Queste proprietà facilitano la parallelizzazione di queste operazioni

Il metodo ***parallelStream()*** permette di generare uno stream le cui operazioni verranno eseguite in modo parallelo (spesso questo non necessita di thread e sincronizzazione)

Generazione di uno Stream

iterate() restituisce uno stream infinito e ordinato a partire da una funzione f applicata ad un *seme* → *seme, f(seme), f(f(seme)) etc…*

Tramite il metodo *limit()* è possibile troncare lo stream ad una lunghezza definita

generate() produce uno stream infinito di valori utilizzando una funzione di tipo *Supplier* → non applica una funzione ad ogni nuovo valore