Gestori combinati di deadline missed

Ho pensato di realizzare dei gestori in grado di poter variare tra le tre strategie base fin qui create (asap, skip, stop).

Quando un job sfora consecutivamente la deadline per un certo numero di volte si deve poter cambiare strategia.Si vogliono quindi modellare una serie di gestori che consento di utilizzare questa strategia “a soglia”

Threshold PolicyHandler

E' la classe base di tutti i gestori a soglia ed estende direttamente la classe madre DeadlineMissedHandler.

Ha una proprietà pubblica threshold, un intero che indica dopo quante violazioni consecutive di deadline si deve cambiare strategia.

Ha un'altra proprietà intera: deadlineCount, ossia il contatore che tiene traccia di quante volte la deadline è stata violata consecutivamente, se questo valore raggiunge o supera la soglia, si deve usare la seconda politica

ASAPStopThresholdPolicyHandler

Questa classe estende ThresholdPolicyhandler e modella un gestore che, finchè un job non sfora consecutivamente la deadline più di un certo valore adotta la politica asap, schedulando i job successivi immediatamente dopo la fine del primo. Tuttavia, qualora il job sfori almeno tante deadline quante la soglia, allora si decide di interromperlo.

Dal momento che il sistema adotta nativamente una politica asap, bisogna assicurarsi che, una volta interrotto il job, non si eseguano i job di recupero, ma li si salti, in maniera analoga a quanto avviene con la skip.

La classe ha quindi come attributi i due gestori di deadline missed.I cui getter e setter sono protected.

I due gestori vengono creati nel cotruttore. Per evitare che il thread controllato richiami direttamente uno di questi due, al termine del costruttore si chiama setIpendigJob mananager del thread controllato.

E' necessario tenere anche una variabile interna che chiamiamo skipcount che contiene il numero di job da saltare una volta che si esegue in politica Stop.

E' necessario che il thread controllato sia un InterrumpiblePeriodicThread, di conseguenza, è necessario adeguare i parametri del costruttore e del getter e del setter di tale proprietà in modo che accettino e restituiscano instanze di questa classe.

E' necessario modificare il metodo setControlledThread, in modo che richiami lo stesso metodo anche per il due handler interni e, analogamente a quanto fatto per il costruttore, sistema il riferimento all'interfaccia IpendingJobManager del thread controllato.

Nel metodo handleAsyncEvent si scrive sul log l'evento di deadline miss e si mette il flag pendingMode del thread controllato a true, poi si deve incrementare deadlineCount e controllare se questo valore è inferiore alla soglia. In questo caso si richama il metodo handleAsyncEvent dell'handler asap. In caso contrario si memorizza il valore dei job da skippare (pari a deadlineCount-1, visto che bisogna schedulare correttamente il job del prossimo periodo) nell'attributo skipCount, si azzera il valore di missedCount e la proprietà pending releases dell'asapHandler e si chiama il metodo HandleAsyncEvent dello stop handler

Nel metodo doPendingJob si guarda se skipCount è unguale a zero, in tal caso dobbiamo usare la politica asap invocando l'omonimo metodo del getore asap. In caso contrario, significa che abbiamo eseguito la stop e il sistema sta schedulando le release di recupero: dobbiamo saltarle tutte. Quindi, si scrive la skip sul log e si decrementa lo skipCount.Se lo skipCount torna a zero si setta il flag pendingMode del thread controllato a false.

SkipStopThresholdPolicyHandler

Questa classe estende ThresholdPolicyhandler e modella un gestore che, finchèhè un job non sfora consecutivamente la deadline più di un certo valore adotta la politica skip, non eseguendo alcun job nel periodo in cui termina quello che aveva sforato la deadline . Tuttavia, qualora il job sfori almeno tante deadline quante la soglia, allora si decide di interromperlo.

Dal momento che il sistema adotta nativamente una politica asap, bisogna assicurarsi che, una volta interrotto il job, non si eseguano i job di recupero, ma li si salti, in maniera analoga a quanto avviene con la skip.

La classe ha quindi come attributi i due gestori di deadline missed.I cui getter e setter sono protected.

I due gestori vengono creati nel cotruttore. Per evitare che il thread controllato richiami direttamente uno di questi due, al termine del costruttore si chiama setIpendigJob mananager del thread controllato.

E' necessario che il thread controllato sia un InterrumpiblePeriodicThread, di conseguenza, è necessario adeguare i parametri del costruttore e del getter e del setter di tale proprietà in modo che accettino e restituiscano instanze di questa classe.

E' necessario modificare il metodo setControlledThread, in modo che richiami lo stesso metodo anche per il due handler interni e, analogamente a quanto fatto per il costruttore, sistema il riferimento all'interfaccia IpendingJobManager del thread controllato.

Nel metodo handleAsyncEvent si scrive sul log l'evento di deadline miss e si mette il flag pendingMode del thread controllato a true, poi si deve incrementare deadlineCount e controllare se questo valore è inferiore alla soglia. In questo caso si richama il metodo handleAsyncEvent dell'handler skip. In caso contrario si azzera il valore di deadlineCount, , si decrementa di uno il valore skipCount dello skiphandler(visto che bisogna schedulare correttamente il job del prossimo periodo) e si chiama il metodo HandleAsyncEvent dello stop handler

Dal momento che la stop di per se non prevede release pendenti e cha la gestione è comunque affidata secondo una tecnica skip, nel metodo doPendingJob si scrive sul log che il job è stato saltato e si invoca il metodo doPendingJob dello skipHandler.

ASAPSkipThresholdPolicyHandler

Questo gestore a sogli, che estende la classe ThresholdPolicyHandler modella una strategia che prevede di recuperare al più un certo numero di job. Nello specifico, se un job sfora un numero di deadline pari o puperiore alla soglia, viene recuperato solo un numero di job pari alla soglia stessa.

La classe ha quindi come attributi i due gestori di deadline missed.I cui getter e setter sono protected.

I due gestori vengono creati nel cotruttore. Per evitare che il thread controllato richiami direttamente uno di questi due, al termine del costruttore si chiama setIpendigJob mananager del thread controllato.

E' necessario che il thread controllato sia un InterrumpiblePeriodicThread, di conseguenza, è necessario adeguare i parametri del costruttore e del getter e del setter di tale proprietà in modo che accettino e restituiscano instanze di questa classe.

E' necessario modificare il metodo setControlledThread, in modo che richiami lo stesso metodo anche per il due handler interni e, analogamente a quanto fatto per il costruttore, sistema il riferimento all'interfaccia IpendingJobManager del thread controllato.

Nel metodo handleAsyncEvent si scrive sul log l'evento di deadline miss e si mette il flag pendingMode del thread controllato a true, poi si deve incrementare deadlineCount e controllare se questo valore è inferiore alla soglia. In questo caso si richama il metodo handleAsyncEvent dell'handler asap. In caso contrario si chiama il metodo HandleAsyncEvent dello stop handler

Il metodo doPendingJob deve controllare se il valore skipCount dell'handler di skip è maggiore di zero, in tal caso significa occorre saltare il job corrente: si scrive quindi l'evento sul log e si chiama il metodo doPendigjob dello skip handler; se dopo questa invocazione lo skipCount ha raggiunto il valore zero bisogna premurarsi di rimettere il flag pendingMode del thread controllato a true dal momento che il metodo doPendingJob lo aveva messo a false non sapendo che bisogna eseguire anche le release secondo la modalità asap.Se, al contrario, il valore di skipCount è zero occorre chiamare il metodo doPendingJob dell'handler asap circondato dalle scritture sul log relative all'inizio ed alla fine del job di recupero.