UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI **MATEMATICA E INFORMATICA**

Interfacce Grafiche e Programmazione ad Eventi

Carmine Dodaro

Anno Accademico 2019/2020

Concorrenza con le API ad alto livello

Concurrences utilities

Java offre delle API ad alto livello per la gestione ad alto livello delle concorrenza. Queste API consentono:

- di aumentare l'affidabilità del codice perché sono state scritte da esperti e testate per evitare i problemi di deadlock, starvation, ecc;
- di migliorare la velocità del codice, in quanto il codice è ottimizzato per le performance;
- di garantire una manutenubilità del codice in quanto il singolo programmatore non si deve occupare della gestione della sincronizzazione.

API ad alto livello

Il package java.util.concurrent

Il package mette a disposizione diversi componenti:

- l'executor framework
- un fork/join framework
- le concurrent collections
- i syncronizers

Executor Framework

Cos'è

L'Executor Framework, che permette di astrarre la creazione e la gestione dei thread attraverso l'uso di altre classi, chiamate executor, e di classi che implementano il concetto di thread pool. Nell'ambito di questo framework si trovano tre interfacce:

- Executor, che consente di creare ed eseguire un nuovo thread rappresentato da un oggetto che estende Runnable;
- ExecutorService, che permette di gestire i thread in modo più versatile rispetto ad Executor;
- ScheduledExecutorService, che permette l'esecuzione dei thread dopo un certo intervallo di tempo, oppure in modo ciclico.

Come si usa

Uso di Executor Framework

- Executors.newCachedThreadPool(), si occupa di creare un nuovo thread quando serve, ed usa la cache nel caso in cui sia necessario riutlizzarli per velocizzare
- Executors.newFixedThreadPool(int NUM), si usano al massimo NUM thread
- Executors.newSingleThreadExecutor(), viene creato un unico thread che esegue le operazioni

```
ExecutorService es = Executors.newFixedThreadPool(3);
es.submit(new AnObjectImplementingRunnable());
es.shutdown();
```

Fork/Join Framework

Cos'è

Consente di sviluppare dei programmi che agiscono in modo parallelo (fork), i cui risultati vengono poi uniti (join). La gestione dei programmi è fatta in modo leggero ed efficiente.

Concurrent Collections

Cosa sono

Sono collezioni che sono particolarmente adatte al contesto della programmazione concorrente. Un esempio è la classe ConcurrentHashMap, che è analoga alla classe HashMap, che rende atomiche le operazioni di interazione con la mappa.

I synchronizers

Cosa sono

Sono classi che implementano in un modo efficiente e pulito la gestione della sincronizzazione tra i vari thread.

Semafori

Un semaforo è usato per controllare il numero di thread che può avere accesso ad una determinata risorsa. Come dice il nome, un semaforo dà accesso ad alcuni thread bloccandone altri.

Programmazione concorrente e JavaFX

Lo Scene Graph di JavaFX non è thread-safe e può essere letto e modificato solo dal thread dell'interfaccia grafica, detto JavaFX Application thread. Quindi, nelle applicazioni JavaFX non è possibile usare dei thread che modificano l'interfaccia grafica come nel caso delle Swing.

Nel caso in cui si voglia implementare una porzione di codice che funziona in parallelo rispetto all'interfaccia grafica, si possono utilizzare le API definite nel package javafx.concurrent. Queste API si occupano di interagire con l'interfaccia grafica di JavaFX, assicurandosi che l'interazione avvenga nel thread corretto.

Il package javafx.concurrent

Struttura

Il package javafx.concurrent consiste dell'interfaccia Worker e due classi, Task e Service, che implementano l'interfaccia Worker.
La classe Task permette agli sviluppatori di sviluppare task asincroni in applicazioni JavaFX. La classe Service esegue i task. Inoltre, le classi Task e Service implementano l'interfaccia EventTarget e questo permette di ricevere notifiche quando lo stato di un Worker cambia.

L'interfaccia Worker

Funzionamento

Definisce un oggetto che esegue codice su uno o più thread in background. Lo stato dell'oggetto Worker è osservabile e usabile dal JavaFX Application thread. Gli stati di un Worker sono i seguenti:

- READY è lo stato che assume quando è creato
- SCHEDULED è lo stato che assume non appena l'oggetto viene schedulato per eseguire il lavoro
- RUNNING è lo stato che assume non appena il codice del Worker viene eseguito
- SUCCEEDED è lo stato che assume quando il codice del Worker termina con successo.
- FAILED è lo stato che assume quando il codice del Worker solleva qualche eccezione
- CANCELLED è lo stato che assume quando il Worker è stato interrotto

La classe Task

Funzionamento

La classe Task può essere usata per implementare il codice che deve essere eseguito su un thread in background.
Le operazioni da eseguire sono: estendere la classe Task ed effettuare l'override del metodo call. Il metodo call è successivamento invocato dal thread in background, quindi l'implementazione può manipolare stati la cui lettura e scrittura è sicura. Per esempio, NON può manipolare uno scene graph attivo (avreste un'eccezione).
Un task può essere eseguito in modo semplice come un oggetto di tipo Runnable:

```
Thread t = new Thread(task);
t.setDaemon(true);
t.start();
```

oppure usando l'ExecutorService API:

```
ExecutorService.submit(task);
```

IMPORTANTE: La classe Task definisce un oggetto che può essere utilizzato una sola volta. Per gli oggetti che vogliono essere usati più volte bisogna usare la classe Service.

Esempio di utilizzo della classe Task

Modificare una ProgressBar

```
Task task = new Task<Void>() {
  @Override public Void call() {
    static final int max = 1000000;
    for (int i=1; i<=max; i++) {
      if (isCancelled()) {
        break;
      }
      updateProgress(i, max);
    }
    return null;
  }
};
ProgressBar bar = new ProgressBar();
bar.progressProperty().bind(task.progressProperty());
new Thread(task).start();
```

La classe Service

Funzionamento

La classe Service è progettata per eseguire un oggetto di tipo Task su uno o più thread in background. Lo scopo di questa classe è di aiutare lo sviluppatore ad implementare la corretta interazione tra i thread in background e il JavaFX Application thread.
Un oggetto di tipo Service può essere avviato, cancellato e riavviato. Per avviare un oggetto di tipo Service si può usare il metodo start().
Di default il Service usa un Executor con un numero fissato o un numero massimo di thread.