Gestion de données spécifique

Comment faciliter la programmation parallèle avec les données

1

Objets immuables

- Ils sont par essence « read only »
- Ils limitent de ce fait les conflits d'accès lecture/écriture
- On peut les considérer comme une version stable de données

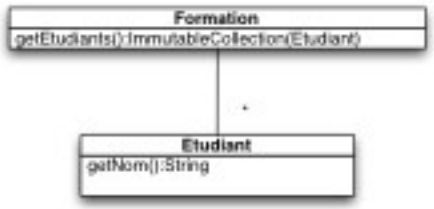
2

Support chez Sun

- Pas de support particulier, seulement une préconisation
- Il existe un moyen d'avoir des collections immuables
  - Pas de type spécifique !

3

Exemple d'emploi



4

Configuration lecteur/rédacteur

- Les copies immuables participent à un algorithme lecteur/rédacteur spécifique
  - pas d'écriture en cours : on lit la valeur à jour
  - une écriture en cours : on lit la valeur précédente

5

Configuration lecteur/rédacteur (2)

- Associé à un modèle transactionnel pour l'écriture
  - début de transaction d'écriture indiquée spécifiquement
  - exécution séquentielle des demandes de transaction
- Pas d'attente pour les lecteurs
  - si une transaction est en cours, ils obtiennent la dernière valeur validée par une transaction

6

Début de transaction

- Création d'une copie en lecture seule pour les autres accès (lecteurs)
- Les lectures faites pendant la transaction retournent l'état de la copie en lecture seule
- À la fin de la transaction, on supprime la copie
  - les objets qui gardent une référence conservent la copie antérieure
  - attention aux alias liés à l'application de Proxy

7

Collections concurrentes

- La bibliothèque Sun contient des structure de donnée composites adaptées au parallélisme
  - BlockingQueue
  - ConcurrentMap
- L'objectif est d'éviter de devoir gérer explicitement des verrous

8

Présentation de BlockingQueue

- L'interface BlockingQueue<T> offre plusieurs catégories d'accès
  - bloquant (put, take)
  - non bloquant (offer, poll, peek) avec timeout éventuel
  - à exceptions (add, remove, element)

9

Accès bloquant

- void put(E e) throws InterruptedException
- E take() throws InterruptedException

10

Exemple classique producteur/consommateur

- class Producteur implements Runnable {
- private BlockingQueue<T> bqueue = ...; // Obtenu de l'extérieur
- void run() { try {
  - while(true) { ...; bq.put(v); }
  - catch(InterruptedException e) { //gasp }

11

Côté consommateur

- class Consommateur implements Runnable {
- private BlockingQueue<T> bqueue = ...; // Obtenu de l'extérieur
- void run() { try {
  - while(true) {T v = bq.take(); ...}
  - catch(InterruptedException e) { //argh }

12

Usage de BlockingQueue

- Moyen de communication entre tâches lancée par Executor
- Minimum de synchronisation
- Mais bien sûr le risque d'interblocage est présent

13

Mise en œuvres concretes de BlockingQueue

- ArrayBlockingQueue
  - simple
- DelayQueue
  - accepte des éléments qui mettent en œuvre Delayed
  - interface Delayed<T> { long getDelay(TimeUnit unit) }
  - seuls les éléments à delai 0 peuvent être retirés

14