Наборы данных для потоков

Классы коллекций

- До Java 2
  - Vector<E> is synchronized
  - Hashtable<K,V> is synchronized
  - Properties is synchronized
- Новые коллекции Java 2
  - . . . is not synchronized
  - synchronization wrapper
- Начиная с Java 5
  - Наборы для безопасной работы с потоками

Старые коллекции

- Vector
- Hashtable

Предназначены для работы из нескольких потоков

Все методы синхронизованы

Коллекции Java 2

- ArrayList <>
- HashMap <>
- . . .

Не обеспечивают безопасную работу с потоками

Необходима явная синхронизация

Особенности использования итераторов:

Если после создания итератора, набор данных изменился в другом потоке, генерируется исключение ConcurrentModificationException

```
List myList = new ArrayList<…>();

synchronized(myList){
    Iterator iter =myList.iterator();
    while (iter.hasNext()){ ….}
}
```

Синхронизующие оболочки

Позволяют обеспечить синхронизацию обычной коллекции там, где она необходима

```
ArrayList<…> usynch = new ArrayList<…>();

 List<. . .> synchAL =
    Collections.synchronizedList (usynch);
```

Синхронизующие оболочки

Получаем двойной доступ – с синхронизацией и без

usynch.add(. . .);

synchAL.clear();

Синхронизующие оболочки

Если нельзя допустить использование потоками несинхронизированных методов, то не сохраняем ссылку на базовый объект

```
Map<…>  synchM =
   Collections.synchronizedMap(
                        new HashMap<…>());
```

!!!

Синхронизующие оболочки управляют лишь методами коллекции

Их действие не распространяется на итераторы коллекций

При использовании итераторов и цикла for each необходима явная блокировка

Наборы для безопасной работы с потоками

Блокирующие очереди

Эффективные очереди

Эффективные хеш-таблицы

Массивы, копируемые при записи

Блокирующие очереди

Приостанавливает работу потока, если он пытается добавить элемент в заполненную очередь или извлечь элемент из пустой очереди

Предоставляют операции трех типов

- Генерирующие исключение при ошибочном использовании
- Возвращающие значение, свидетельствующее об ошибке
- Блокирующие

Операции

Summary of BlockingQueue methods

	Throws exception	*Special value*	*Blocks*	*Times out*
Insert	`add(e)`	`offer(e)`	`put(e)`	`offer(e, time, unit)`
Remove	`remove()`	`poll()`	`take()`	`poll(time, unit)`
Examine	`element()`	`peek()`	*not applicable*	*not applicable*

Операции

- Генерирующие исключение

  add() - при переполнении очереди IllegalStateException

  remove() - если очередь пуста NoSuchElementException

  element() если очередь пуста NoSuchElementException

- Возвращающие признак ошибки

  offer() -false, если очередь заполнилась

  poll()- null, если очередь пуста

  peek() - null, если очередь пуста

- Блокирующие

  put()

  take()

Классы блокирующих очередей

- LinkedBlockingQueue <E>
- ArrayBlockingQueue <E>
- DelayQueue <E extends Delayed>
- PriorityBlockingQueue <E >

SynchronousQueue<E>

- Реализует объединение потока-поставщика и потока-потребителя
- Данные передаются только в одном направлении
- Очередь не используется
- Если поток вызвал метод **`put()`**, его выполнение приостанавливается, пока другой поток не вызовет метод **`take()`**

Java 6 и Java 7

- 1.6 Добавлены блокирующие деки
  - **LinkedBlockingDeque<E>**
- 1.7 интерфейс *TransferQueue*
  - когда производитель отправляет сообщение потребителю с помощью метода *transfer ()* , производитель остается заблокированным, пока сообщение не будет использовано

Эффективные коллекции

- Эффективные наборы поддерживают большое количество читающих потоков и фиксированное число записывающих.
- Наборы данных используют специальные алгоритмы, позволяющие избежать блокирования всей структуры и минимизировать конфликты, допуская одновременный доступ к различным частям структуры
- Эффективные наборы данных представляют «слабо согласованные итераторы»
- Это означает, что итератор не обязательно отражает все изменения, произошедшие после их создания, но никогда не возвращают значение дважды и не вызывают исключение

Классы эффективных коллекций

- ConcurrentLinkedQueue <E>
- ConcurrentHashMap <E>
- CopyOnWriteArrayList <E>
- CopyOnWriteArraySet <E>

ConcurrentHashMap<K,V>

```
// изменена хеш-функция и способ организации карты

// добавить, если нет объекта value по ключу
   V putIfAbsent(K key, V value);
 // удалить, если имеется объект value с ключом
   boolean remove(K key, V value);
// заменить oldValue новым newValue объекта с ключом
   boolean replace(K key, V oldValue, V newValue);
// заменить новым значением newValue объект с ключом
    V replace(K key, V newValue);
```

Коллекции, копируемые при записи

- CopyOnWriteArrayList <E>
- CopyOnWriteArraySet <E>

Перед каждой модификацией коллекция копирует свое содержимое в новую, чтобы операции чтения содержимого коллекции выполнялись без синхронизации (так как они никогда не работают с изменяемыми данными)