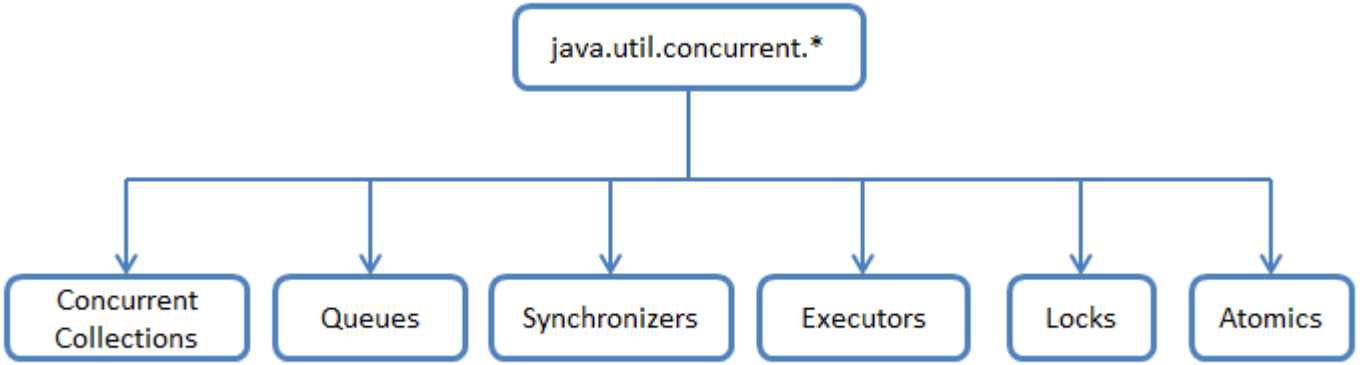
Параллельное программирование: Лекция 6

Учебный курс

Кензин Максим Юрьевич
Кафедра Информационных технологий
ИМИТ ИГУ / Институт динамики
систем и теории управления СО РАН

ИМИТ ИГУ, 2022

Java.util.concurrent.*



Потокобезопасные коллекции

Потокобезопасные коллекции

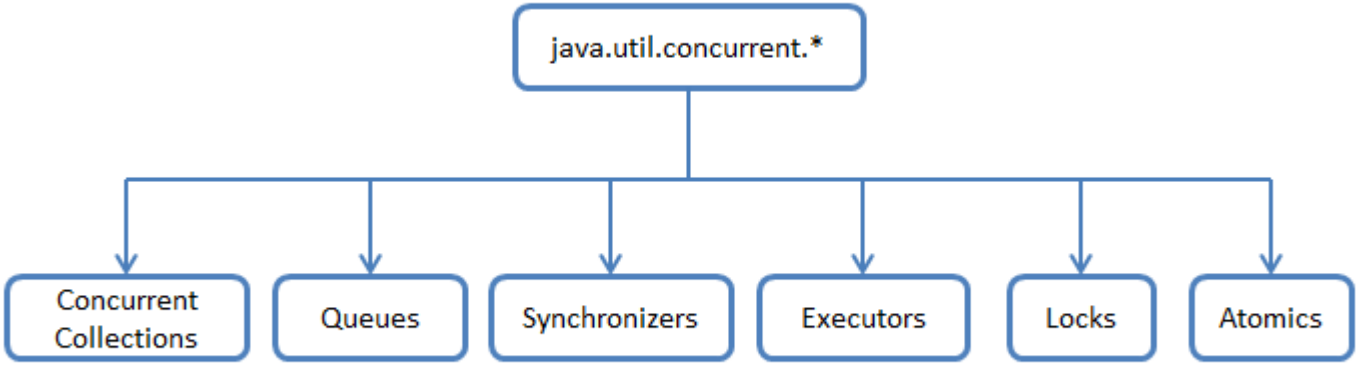
- CopyOnWriteArrayList – аналог ArrayList для тех коллекций, где редко используются команды изменения (add, set, remove, clear).

  При каждом изменении создает копию всего массива в памяти. Зато (!) можно обходить.

- ConcurrentHashMap – аналог HashMap с потокобезопасной реализацией изменений.

ConcurrentHashMap<K,V>

Java.util.concurrent.*



Потокобезопасные очереди

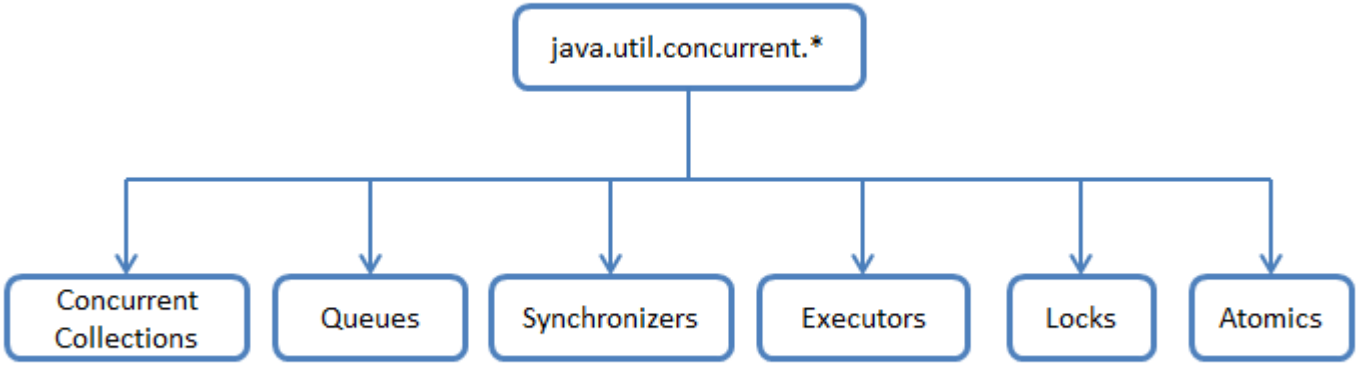
Потокобезопасные коллекции

Потокобезопасные очереди

- BlockingQueue – очереди с реализацией блокировки (по таймеру/до возможности выполнения задачи) в двух случаях:
  - при попытке получения элемента из пустой очереди;
  - при попытке размещения элемента в полной очереди.

- Виды BlockingQueue:
  - ArrayBlockingQueue — очередь, реализующая классический кольцевой буфер;
  - LinkedBlockingQueue — односторонняя очередь на связанных узлах;
  - LinkedBlockingDeque — двунаправленная очередь на связанных узлах;
  - И др….

Java.util.concurrent.*



Шаблоны синхронизации

Локи

Потокобезопасные очереди

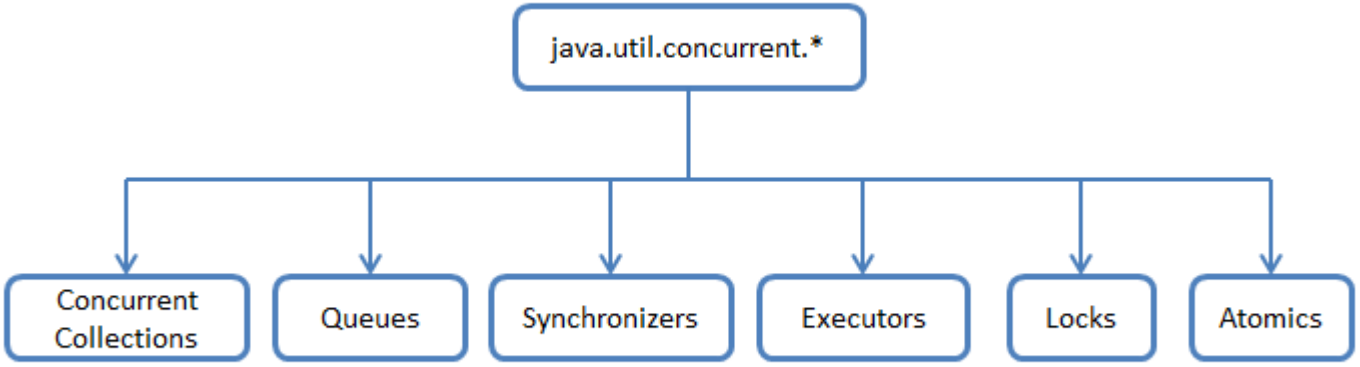
Потокобезопасные коллекции

Шаблоны синхронизации

- Semaphore;
- Phaser;
- CountDownLatch;
- CyclicBarrier;
- Exchanger<V>.

- Lock;
- ReentrantLock;
- Condition.

Java.util.concurrent.*



Атомарные операции

Шаблоны синхронизации

Локи

Потокобезопасные очереди

Потокобезопасные коллекции

Атомарные операции

- AtomicBoolean;
- AtomicLong;
- AtomicLongArray.

AtomicInteger;

AtomicIntegerArray;

```
class synchronized Counter {
  private int c = 0;

  public synchronized void increment()
         {c++;}

  public synchronized void decrement()
         {c--;}

  public synchronized int value()
         { return c; }
}
```

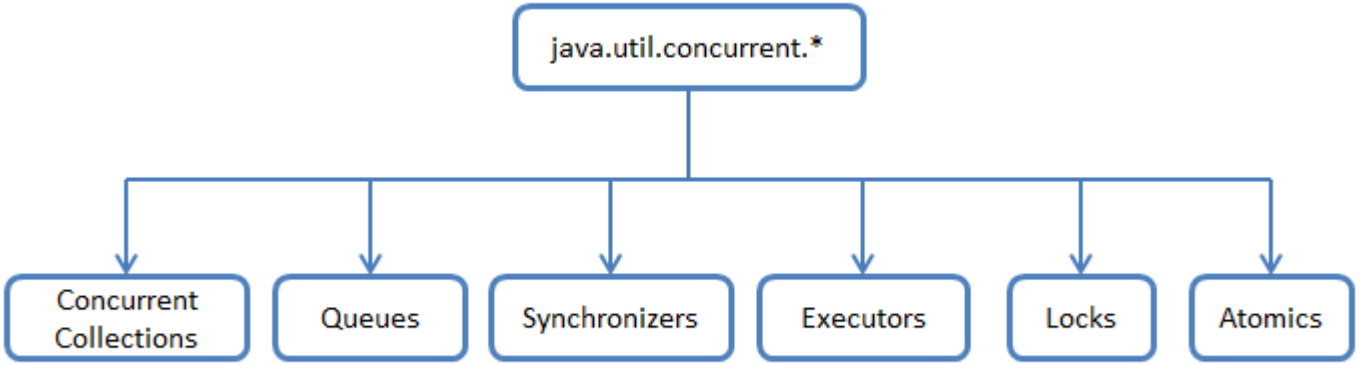
```
class AtomicCounter {
  private AtomicInteger c = new
AtomicInteger(0);

  public void increment()
         {c.incrementAndGet();}

  public void decrement()
         {c.decrementAndGet();}

  public int value()
         {return c.get();}
  }
```

Java.util.concurrent.*



Исполнители

Атомарные операции

Шаблоны синхронизации

Локи

Потокобезопасные очереди

Потокобезопасные коллекции

Шаблоны синхронизации

- Semaphore;
- Phaser;
- CountDownLatch;
- CyclicBarrier;
- Exchanger<V>.

- Lock;
- ReentrantLock;
- Condition.

Semaphore

- Semaphore(**int** permits, **boolean** fair): конструктор объекта-семафора.
  - **permits –** максимальное количество объектов с доступом к данным;
  - **fair** – порядок доступа к данным (true – соответствует порядку запроса, false – автоматически).

Управление объектом-семафором с помощью методов:

- **acquire()** – запрос на разрешение пользования данными;
- **release()** – отказ от использования данными.

Задача 1: База данных (Starvation).

Есть база данных, представленная в виде объекта.

Генерируется очередь пользователей, пытающихся получить доступ к базе длительностью 100-1000мс (через sleep). Каждый пользователь относится либо к классу читателей, либо к классу писателей. Обеспечить управление доступом к базе данных таким образом, чтобы одновременно пользоваться базой могли:

A) Ровно один писатель и ноль читателей;

B) Ноль писателей и сколько угодно читателей.

Задание на дом (улучшить):

Если база занята читателями, а следующий в очереди находится писатель, он может пропускать вперед тех читателей, доступ которых к базе не приведет к увеличению его времени ожидания.