Лекция 1

Расширенные возможности многопоточного программирования

Высокоуровневые средства многопоточного программирования

- Объекты блокировки (Locks)
- Исполнители (Executors)
- Коллекции (Concurrent collections)
- Атомарные переменные (Atomic variables)

С версии 1.5

Пакет java.util.concurrent

Reentrant lock

- public void **lock**()
- public boolean **tryLock**()
- public boolean **tryLock**(long timeout, TimeUnit unit) throws InterruptedException
- public void **lockInterruptibly**() throws InterruptedException
- public void **unlock**()
- public int **getHoldCount**()
- public boolean **isHeldByCurrentThread**()

```
public class Safelock {
    static class Friend {
        private final String name;
        private final Lock lock = new ReentrantLock();

        public Friend(String name) {
            this.name = name;
        }

        public String getName() {
            return this.name;
        }

        public boolean impendingBow(Friend bower) {
            Boolean myLock = false;
            Boolean yourLock = false;
            try {
                myLock = lock.tryLock();
                yourLock = bower.lock.tryLock();
            } finally {
                if (! (myLock && yourLock)) {
                    if (myLock) {
                        lock.unlock();
                    }
                    if (yourLock) {
                        bower.lock.unlock();
                    }
                }
            }
            return myLock && yourLock;
        }
```

```
public void bow(Friend bower) {
            if (impendingBow(bower)) {
                try {
                    System.out.format("%s: %s has bowed to me!%n",
                            this.name, bower.getName());
                    bower.bowBack(this);
                } finally {
                    lock.unlock();
                    bower.lock.unlock();
                }
            } else {
                System.out.format("%s: %s started to bow to me, but" +
                        " saw that I was already bowing to him.%n",
                        this.name, bower.getName());
            }
        }

        public void bowBack(Friend bower) {
            System.out.format("%s: %s has bowed back to me!%n",
                    this.name, bower.getName());
        }
    }
```

```
static class BowLoop implements Runnable {
        private Friend bower;
        private Friend bowee;

        public BowLoop(Friend bower, Friend bowee) {
            this.bower = bower;
            this.bowee = bowee;
        }

        public void run() {
            Random random = new Random();
            for (;;) {
                try {
                    Thread.sleep(random.nextInt(10));
                } catch (InterruptedException e) {}
                bowee.bow(bower);
            }
        }
    }
    public static void main(String[] args) {
        final Friend alphonse = new Friend("Alphonse");
        final Friend gaston = new Friend("Gaston");
        new Thread(new BowLoop(alphonse, gaston)).start();
        new Thread(new BowLoop(gaston, alphonse)).start();
    }
}
```

Затвор

CountDownLatch – класс, позволяющий потокам ожидать окончания операций, выполняемые другими потоками. Счетчик задается в конструкторе.

- countDown() – уменьшает счетчик на 1.
- await() – блокирует поток, пока счетчик не 0.

Барьер

CyclicBarrier – барьер. Потоки «подходят» к барьеру и ожидают последнего (количество задается в конструкторе)

- await() – блокирует поток до наступления одного из событий:
  - подошел последний поток
  - Текущий поток был прерван
  - Один из ожидающих у барьера потоков был прерван
  - Один из ожидающий у барьера потоков закончил ожидать по таймауту
  - Кто-то вызвал метод reset() у барьера

Семафор

Semaphore позволяет ограничить доступ к ресурсу до нескольких потоков.

- Конструктор Semaphore(int permits)
- acquire()
- release()
- tryAcquire()

ReadWriteLock

ReadWriteLock позволяет синхронизовать потоки, желающие получить доступ на чтение или на запись.

Методы ReadWriteLock:

- `Lock readLock()`
- `Lock writeLock()`

Механизм представлен в виде класса **ReentrantReadWriteLock**

Исполнители

- Интерфейс Executor

```
new Thread(r)).start();
```

Заменяется на:

```
e.execute(r);
```

r – объект класса, имплементирующий Runnable

- Интерфейс ExecutorService

`execute(Runnable r) и Future Submit(Callable c)`

- Интерфейс ScheduledExecutorService

Позволяет выполнять задачи с определенной задержкой

Runnable и Callable

- Runnable – выполяемая задача
- Callable<T> – выполняемая задача, имеющая результат.

```
public class Factorial implements Callable<Long> {
    public Factorial(int n) {
       this.n = n;
    }
    public Long call() {
       int res = 1;
       for (int i = 1; i < n; i++)
          res *= n;
       return n;
    }
}
```

Класс Future

Метод `Future<T> submit(Callable<T> c)` интерфейса `ExecutorService` возвращает объект Future<T>.

Методы Future:

- `boolean` **`isDone`**`()`
- `V` **`get`**`() throws InterruptedException() throws InterruptedException, ExecutionException`
- `V` **`get`**`(long timeout, TimeUnit(long timeout, TimeUnit unit) throws InterruptedException(long timeout, TimeUnit unit) throws InterruptedException, ExecutionException(long timeout, TimeUnit unit) throws InterruptedException, ExecutionException, TimeoutException`
- `boolean` **`cancel`**`(boolean mayInterruptIfRunning)`
- `boolean` **`isCancelled`**`()`

Пулы потоков

Пулы потоков предназначены для параллельного выполнения задач Runnable и Callable.

Класс Executors предоставляет:

- `public static` ExecutorService **`newSingleThreadExecutor`**`()`
- `public static` ExecutorService **`newFixedThreadPool`**`(int nThreads)`
- `public static` ExecutorService **`newCachedThreadPool`**`()`

Интерфейс BlockingQueue

- `boolean` **`add`**`(E e)`
- `boolean` **`remove`**`(Object o)`
- `boolean` **`offer`**`(E e)`
- `boolean` **`offer`**`(E(E e, long timeout, TimeUnit(E e, long timeout, TimeUnit unit) throws InterruptedException`
- `void` **`put`**`(E(E e) throws InterruptedException`
- `E` **`take`**`() throws InterruptedException`
- `E` **`poll`**`(long timeout, TimeUnit(long timeout, TimeUnit unit) throws InterruptedException`
- `E element()`
- `E peek()`

Сводка по методам

	Вызывает исключение	Возвращает специальное значение	Блокируется	Завершение блокировки по тайм-ауту
Добавление	add(e)	offer(e)	put(e)	offer(e, time, unit)
Удаление	remove()	poll()	take()	poll(time, unit)
Просмотр	element()	peek()	-	-

Коллекции

Интерфейс BlockingQueue имеет реализации:

- `ArrayBlockingQueue`
- `DelayQueue`
- `LinkedBlockingQueue`
- `PriorityBlockingQueue`
- `SynchronousQueue`

- `LinkedBlockingDeque`

Атомарные операции

Данный класс не приспособлен для использования из разных потоков

```
class Counter {
    private int c = 0;
    public void increment() {
        c++;
    }
    public void decrement() {
        c--;
    }
    public int value() {
        return c;
    }
}
```

Атомарное действие – действие, которое происходит за один раз. Оно либо производится полностью, либо не производится вообще. Результат атомарной операции становится виден только после ее завершения.

Атомарные операции

Атомарными операциями являются:

- Чтение и запись ссылок на объекты
- Чтение и запись значений переменных примитивных типов (за исключением long и double)
- Чтение и запись значение переменных любых типов, объявленных как **volatile**

Атомарные операции

Объекты данного класса может использоваться из разных потоков, но не эффективно.

```
class SynchronizedCounter {
    private int c = 0;

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

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

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

Атомарные переменные

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

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

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

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

}
```

Классы атомарных переменных

- AtomicBoolean
- AtomicInteger
- AtomicLong
- AtomicReference
- AtomicIntegerArray
- AtomicLongArray
- AtomicReferenceArray

Методы классов атомарных переменных

- get()
- set(value)
- getAndSet(value)
- compareAndSet(expect, newValue)

Для чисел:

- addAndGet(delta)
- getAndAdd(delta)
- getAndDecrement()
- getAndIncrement()
- incrementAndGet()
- decrementAndGet()