НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Многопоточное программирование на Java

Часть 2: Пакет java.util.concurrent

Беркунский Е.Ю., кафедра ИУСТ, НУК
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua

Зачем?

- Класс Thread и интерфейс Runnable
- synchronized блоки и методы
- wait – notify - notifyAll

Если и так есть средства для решения любой задачи многопоточности. Зачем еще что-то?

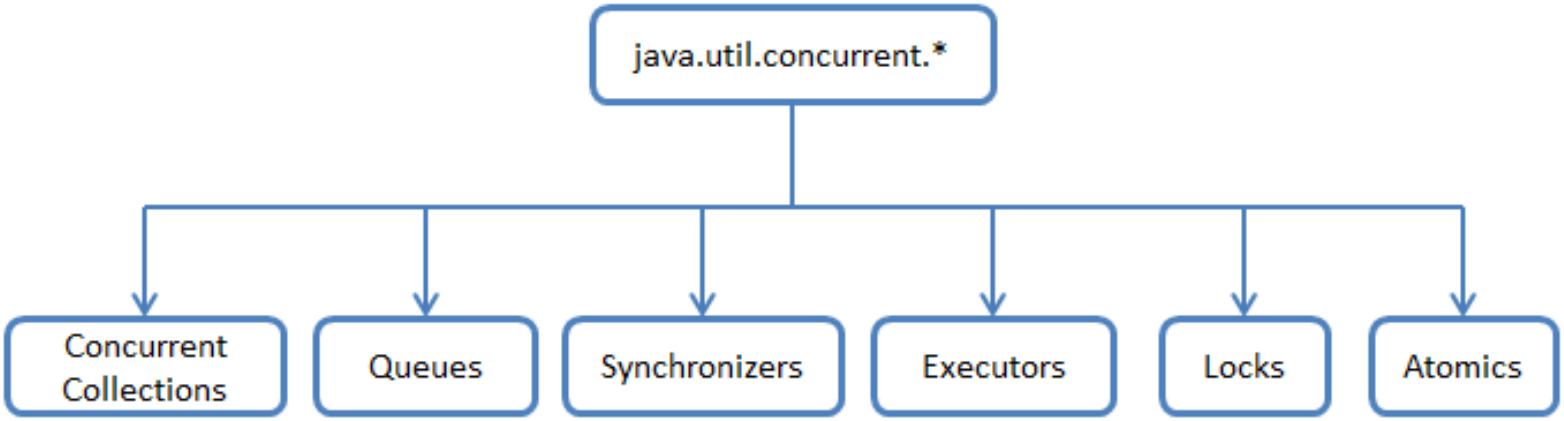
Зачем?

- Класс Thread и интерфейс Runnable
- synchronized блоки и методы
- wait – notify - notifyAll

Если и так есть средства для решения любой задачи многопоточности. Зачем еще что-то?

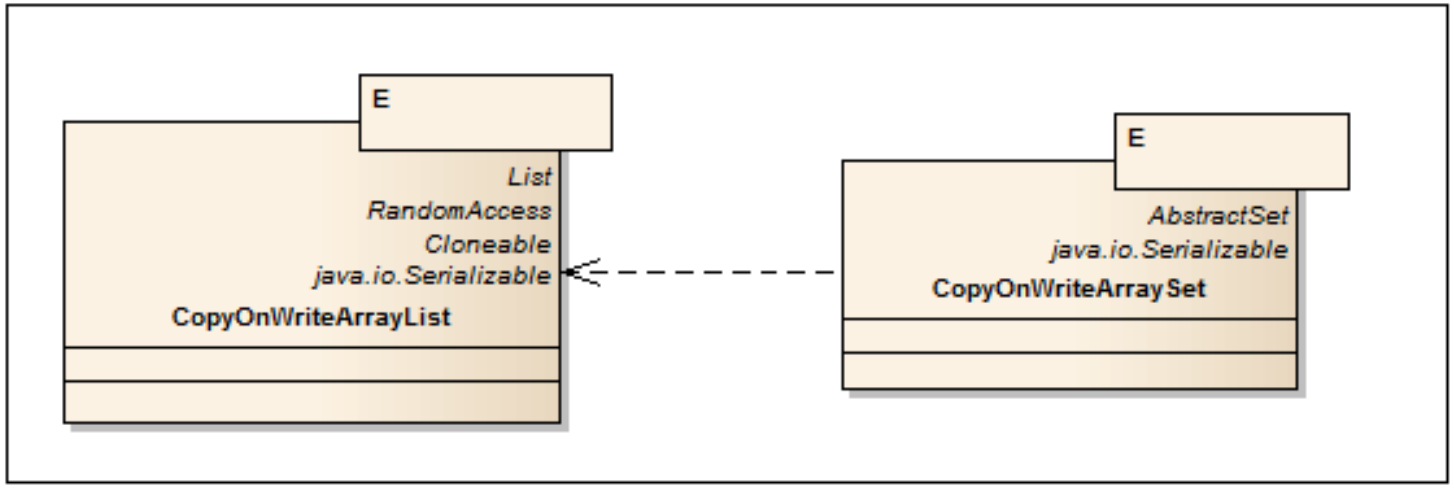
Для удобства!

Состав java.util.concurrent



1. Concurrent Collections

CopyOnWrite коллекции



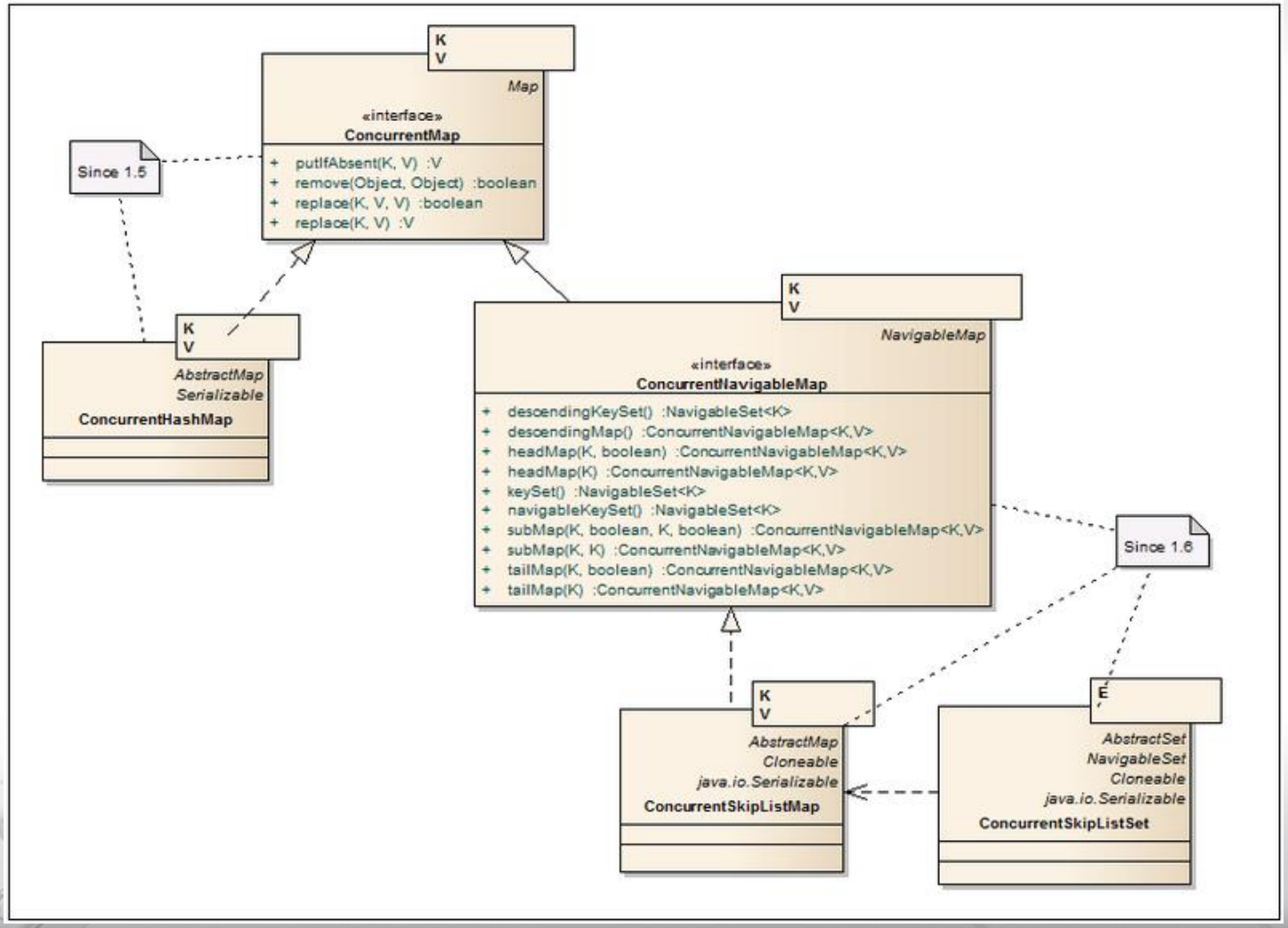
CopyOnWriteArrayList<E> — Потокобезопасный аналог ArrayList, реализованный с CopyOnWrite алгоритмом.

CopyOnWriteArraySet<E> — Имплементация интерфейса Set, использующая за основу CopyOnWriteArrayList.
В отличии от CopyOnWriteArrayList, дополнительных методов нет

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

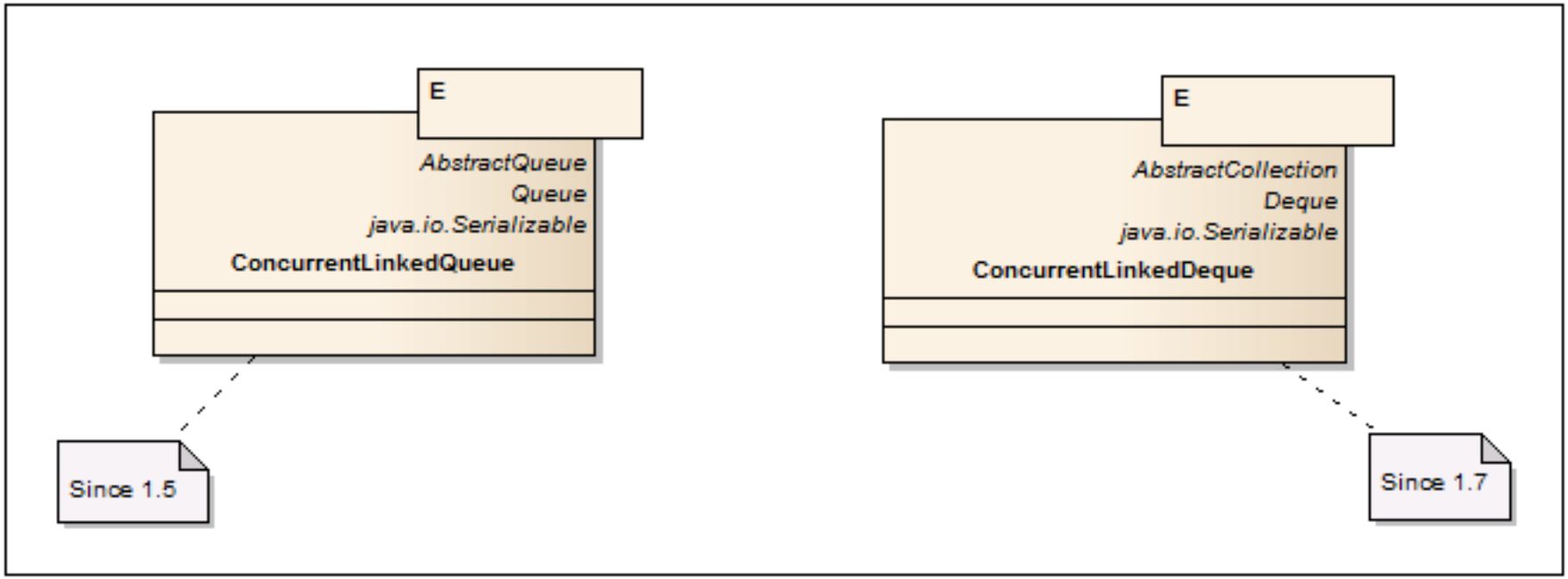
1. Concurrent Collections

Scalable Maps



2. Queues

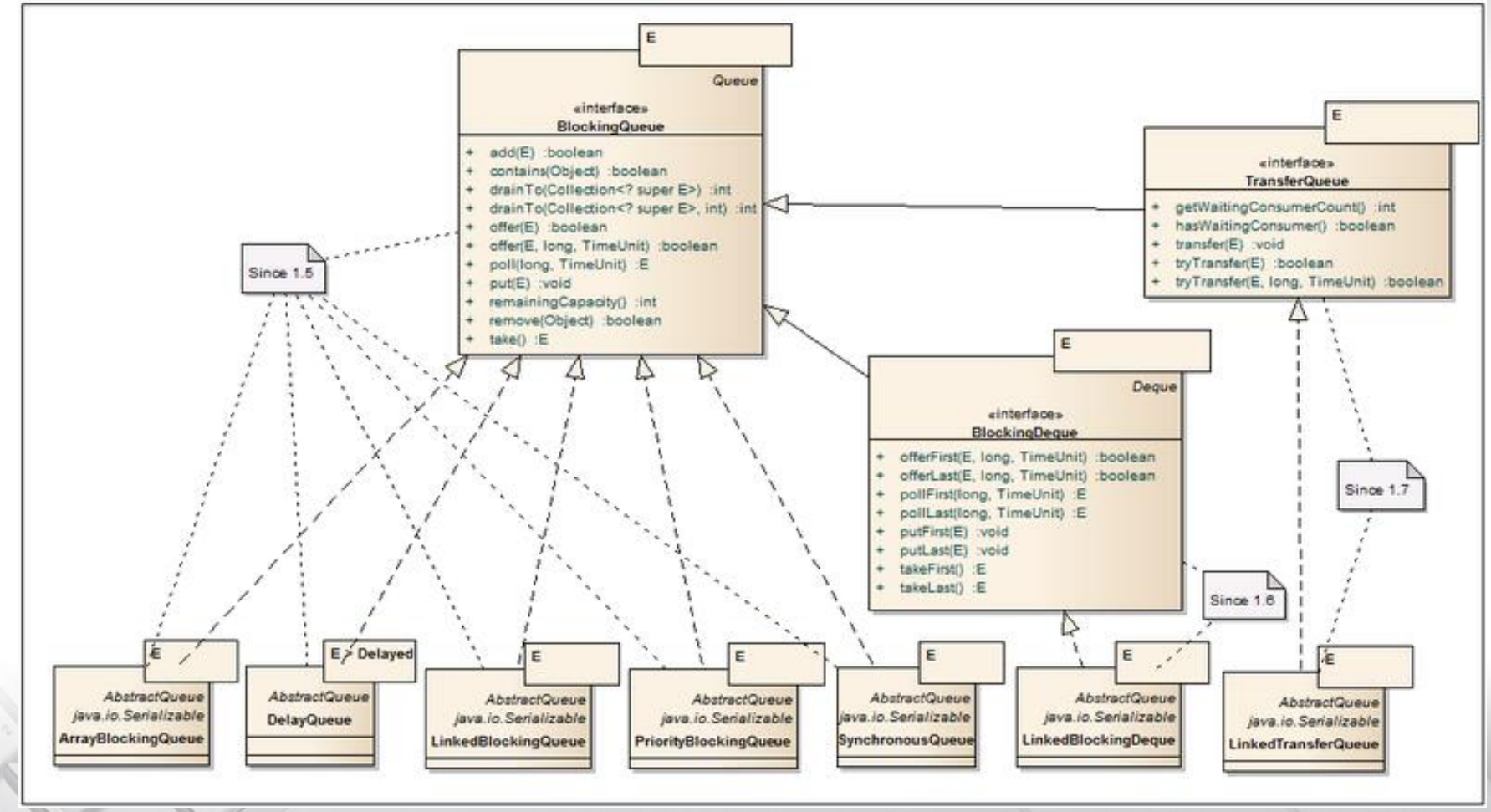
Non-Blocking Queues



НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

2. Queues

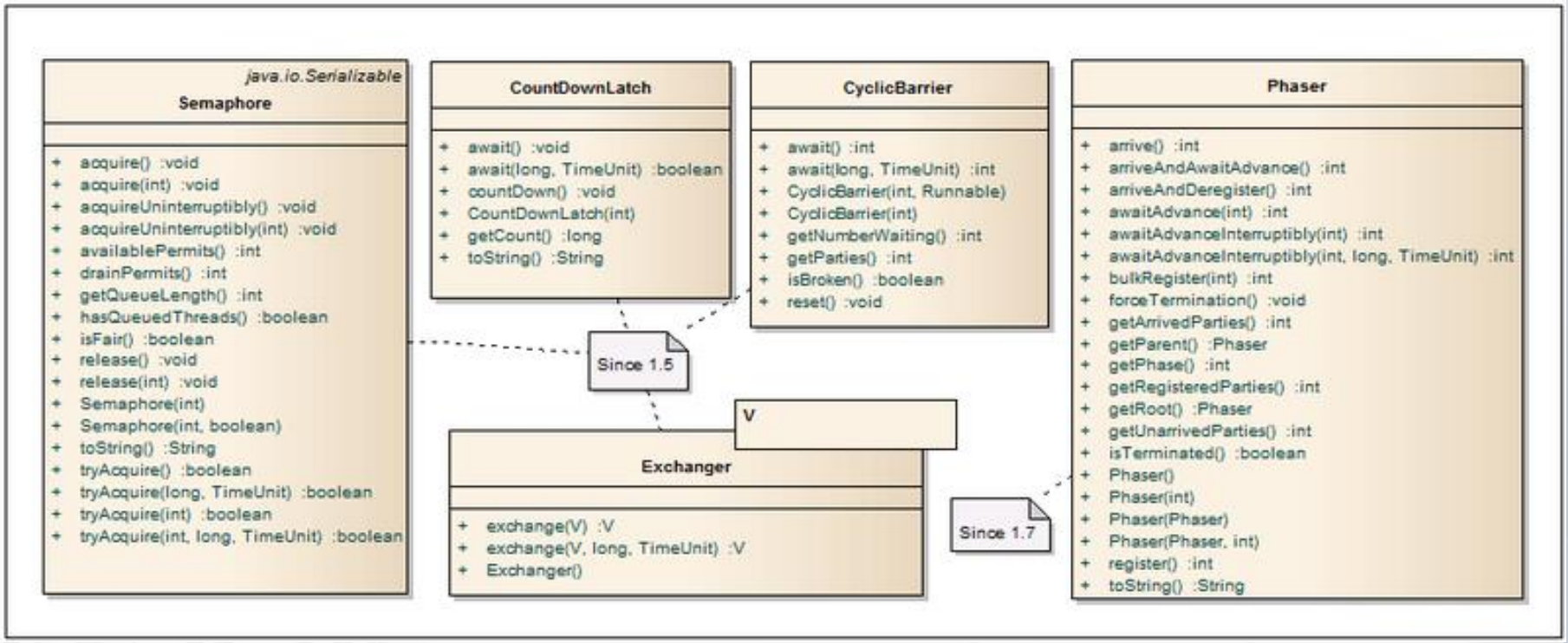
Blocking Queues



НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

3. Synchronizers

Классы для активного управления потоков



Semaphore

Семафоры чаще всего используются для ограничения количества потоков при работе с аппаратными ресурсами или файловой системой.

- Доступ к общему ресурсу управляется с помощью счетчика.
- Если он больше нуля, то доступ разрешается, а значение счетчика уменьшается.
- Если счетчик равен нулю, то текущий поток блокируется, пока другой поток не освободит ресурс.
- Количество разрешений и «честность» освобождения потоков задается через конструктор.

CountDownLatch

Позволяет одному или нескольким потокам ожидать до тех пор, пока не завершится определенное количество операций, выполняющих в других потоках.
Классический пример с драйвером довольно неплохо описывает логику класса:

- Потоки, вызывающие драйвер, будут висеть в методе await (с таймаутом или без), пока поток с драйвером не выполнит инициализацию с последующим вызовом метода countDown.
- Этот метод уменьшает счетчик count down на единицу.
- Как только счетчик становится равным нулю, все ожидающие потоки в await продолжат свою работу, а все последующие вызовы await будут проходить без ожиданий.
- Счетчик countDown одноразовый и не может быть сброшен в первоначальное состояние.

CyclicBarrier

Может использоваться для синхронизации заданного количества потоков в одной точке.

- Барьер достигается когда N-потоков вызовут метод await(...) и заблокируются.
- После чего счетчик сбрасывается в исходное значение, а ожидающие потоки освобождаются.
- Дополнительно, если нужно, существует возможность запуска специального кода до разблокировки потоков и сброса счетчика.
- Для этого через конструктор передается объект с реализацией Runnable интерфейса.

Exchanger<V>

Основное предназначение данного класса — это обмен объектами между двумя потоками.

- При этом, также поддерживаются null значения, что позволяет использовать данный класс для передачи только одного объекта или же просто как синхронизатор двух потоков.
- Первый поток, который вызывает метод exchange(...) заблокируется до тех пор, пока тот же метод не вызовет второй поток.
- Как только это произойдет, потоки обменяются значениями и продолжат свою работу.

Phaser (jdk7)

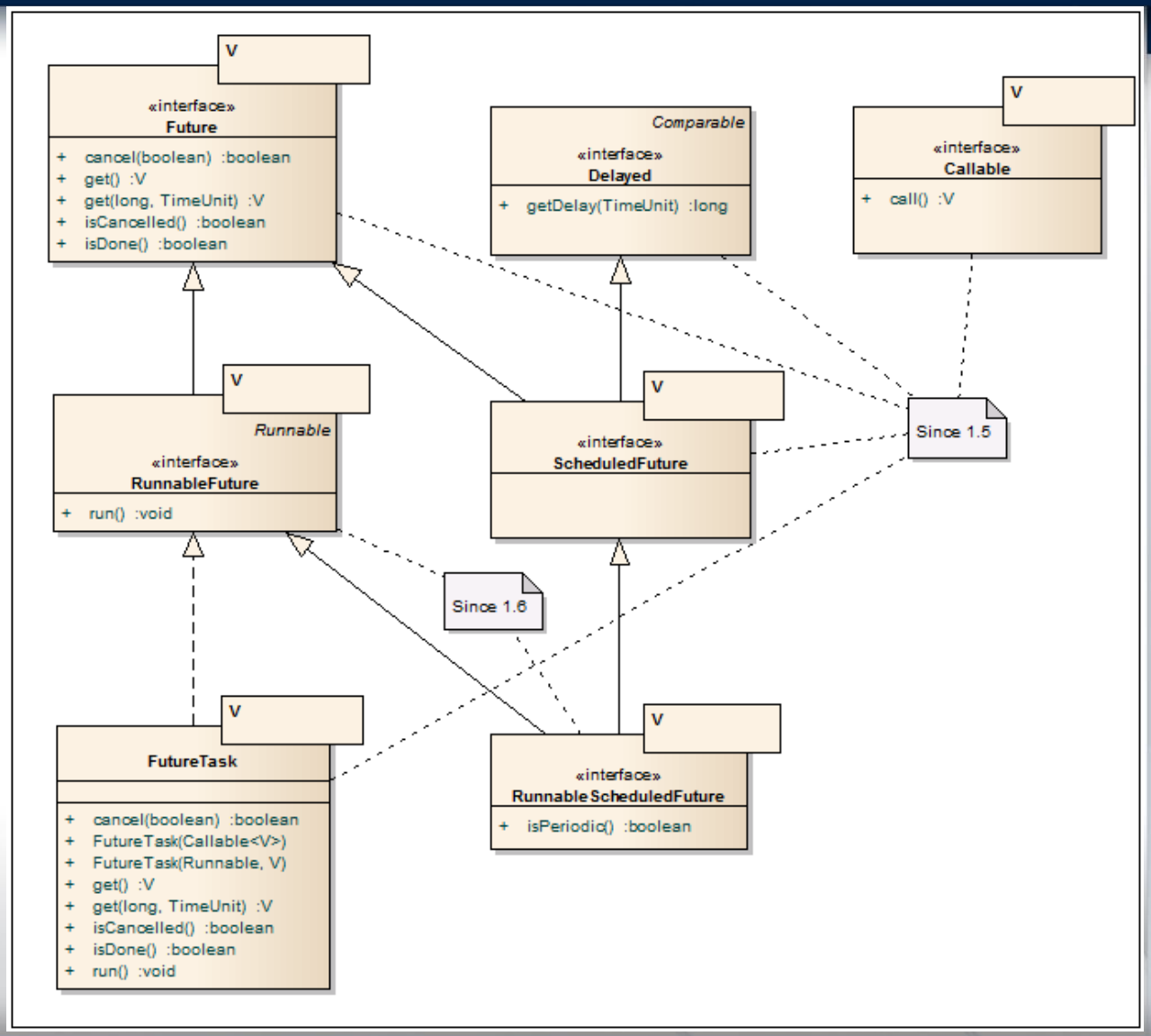
Улучшенная реализация барьера для синхронизации потоков, которая совмещает в себе функционал CyclicBarrier и CountDownLatch, вбирая в себя самое лучшее из них.

- Так, количество потоков жестко не задано и может динамически меняться.
- Класс может повторно переиспользоваться и сообщать о готовности потока без его блокировки.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

4. Executors

Future and Callable



Future<V>

Интерфейс для получения результатов работы асинхронной операции.

- Ключевым методом является метод get, который блокирует текущий поток до завершения работы асинхронной операции в другом потоке.
- Также, дополнительно существуют методы для отмены операции и проверки текущего статуса.
- В качестве имплементации часто используется класс FutureTask.

RunnableFuture<V>

RunnableFuture используется для запуска асинхронной части.

- Успешное завершение метода run() завершает асинхронную операцию и позволяет вытаскивать результаты через метод get.

Callable<V>

Расширенный аналог интерфейса Runnable для асинхронных операций.

- Позволяет возвращать типизированное значение и кидать checked exception.
- Несмотря на то, что в этом интерфейсе отсутсвует метод run(), многие классы java.util.concurrent поддерживают его наряду с Runnable

FutureTask<V>

Имплементация интерфейсов
Future / RunnableFuture.

- Асинхронная операция принимается на вход одного из конструкторов в виде Runnable или Callable объектов.
- Сам класс FutureTask предназначен для запуска в worker потоке, например через new Thread(task).start(), или через ThreadPoolExecutor.
- Результаты работы асинхронной операции вытаскиваются через метод get(...)

Delayed

Используется для асинхронных задач, которые должны начаться в будущем, а также в DelayQueue.

- Позволяет задавать время до начала асинхронной операции

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

ScheduledFuture<V>

Маркерный интерфейс, объединяющий Future и Delayed интерфейсы

RunnableScheduledFuture<V>

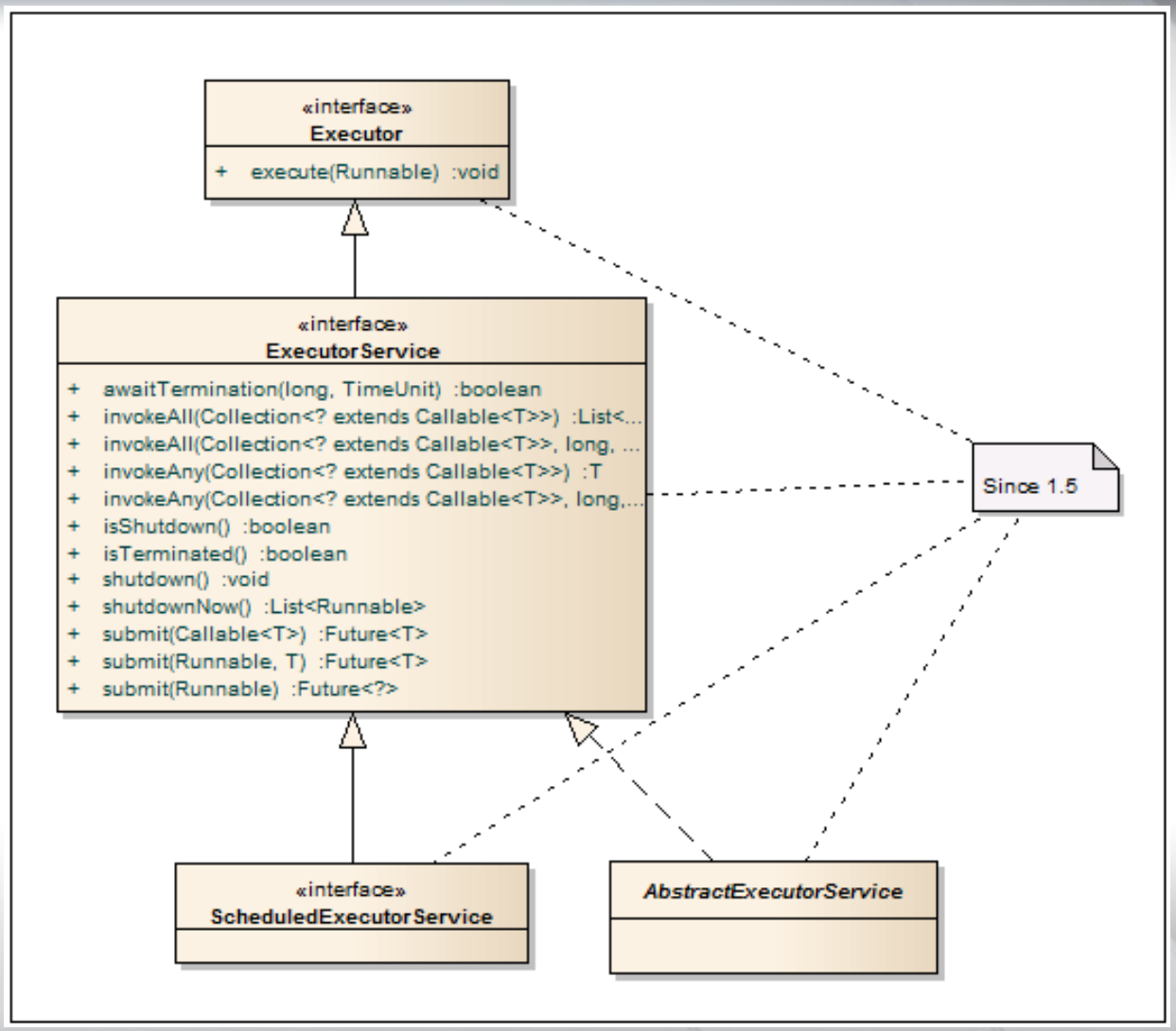
Интерфейс, объединяющий RunnableFuture и ScheduledFuture.

- Дополнительно можно указывать является ли задача одноразовой или же должна запускаться с заданной периодичностью.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

4. Executors

Executor Services



Executor

Базовый интерфейс для классов, реализующих запуск Runnable задач.

Таким оразом обеспечивается разделение между добавлением задачи и способом её запуска.

ExecutorService

Интерфейс, описывающий сервис для запуска Runnable или Callable задач.

- Методы submit на вход принимают задачу в виде Callable или Runnable, а возвращает Future, через который можно получить результат.
- Методы invokeAll работают со списками задач с блокировкой потока до завершения всех задач в переданном списке или до истечения заданного времени
- Методы invokeAny блокируют вызывающий поток до завершения любой из переданных задач.
- После вызова метода shutdown, данный сервис больше не будет принимать задачи, бросая исключение RejectedExecutionException при попытке добавить задачу в сервис.

ScheduledExecutorService

В дополнение к методам ExecutorService,

добавляет возможность запускать отложенные задачи.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

AbstractExecutorService

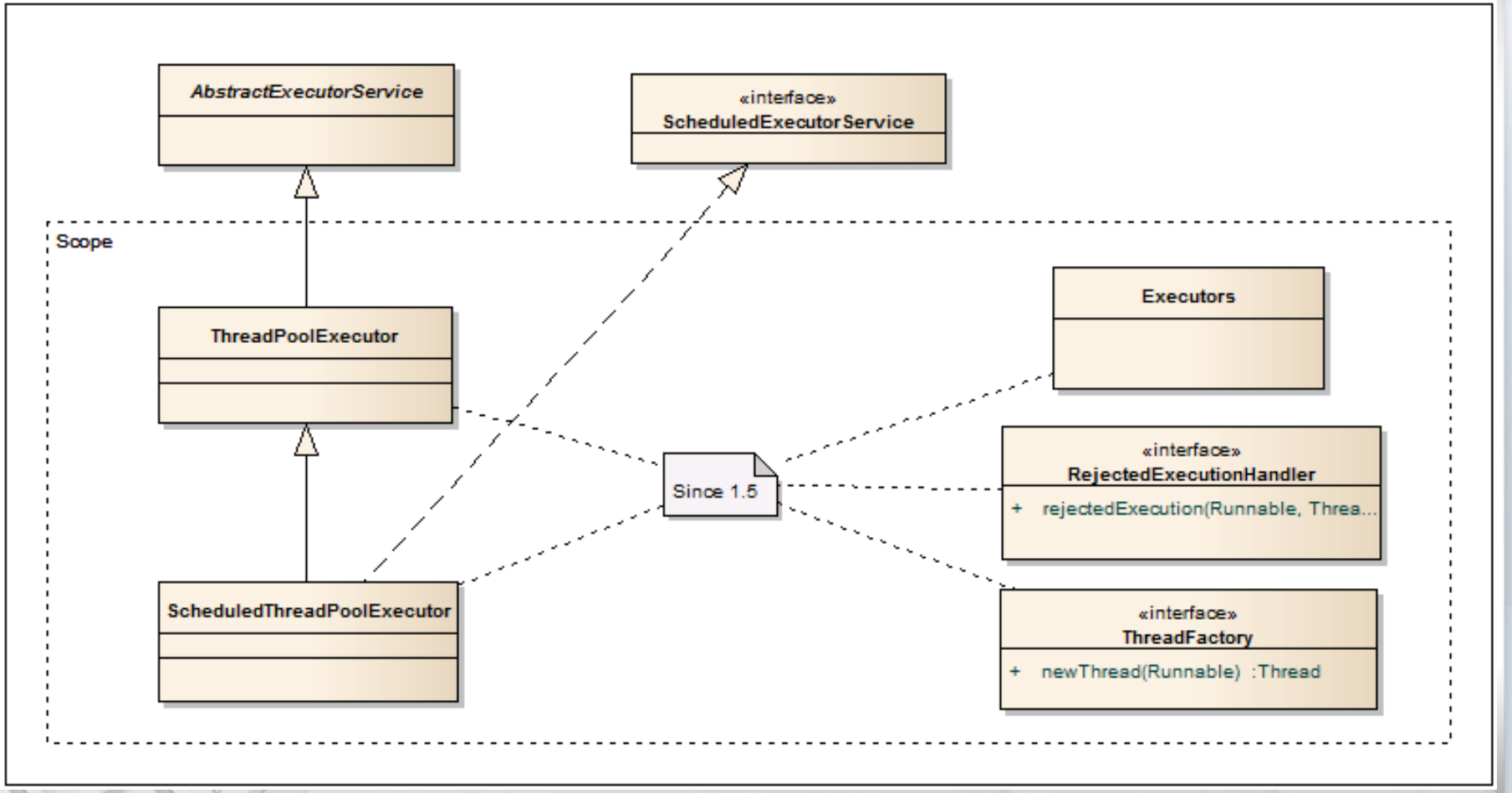
Абстрактный класс для построения ExecutorService'a.

- Содержит базовую реализацию методов submit, invokeAll, invokeAny.
- От этого класса наследуются ThreadPoolExecutor, ScheduledThreadPoolExecutor и ForkJoinPool.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

4. Executors

ThreadPoolExecutor & Factory



Executors

Класс-фабрика для создания ThreadPoolExecutor, ScheduledThreadPoolExecutor.

- Для создания одного из этих пулов используется эта фабрика.
- Кроме того, она содержит разные адаптеры
  - Runnable-Callable
  - PrivilegedAction-Callable
  - PrivilegedExceptionAction-Callable

и другие

ThreadPoolExecutor

- Используется для запуска асинхронных задач в пуле потоков.
- При этом практически полностью отсутствует оверхэд на поднятие и остановку потоков.
- За счет фиксируемого максимума потоков в пуле обеспечивается прогнозируемая производительность приложения.
- Создавать данный пул предпочтительно через один из методов фабрики Executors.
- Если стандартных конфигураций недостаточно, то через конструкторы или сеттеры можно задать все основные параметры пула

ScheduledThreadPoolExecutor

- Позволяет запускать задачи после определенной задержки,
- И с указанной периодичностью, что позволяет реализовать на базе этого класса Timer Service

ThreadFactory

- По умолчанию, ThreadPoolExecutor использует стандартную фабрику потоков, получаемую через Executors.defaultThreadFactory().
- Если нужно что-то больше, например задание приоритета или имени потока, то можно создать класс с реализацией этого интерфейса и передать его в ThreadPoolExecutor

RejectedExecutionHandler

Определяет обработчик для задач, которые по каким-то причинам не могут быть выполнены через ThreadPoolExecutor.

Например, когда нет свободных потоков или сервис выключается или выключен (shutdown).

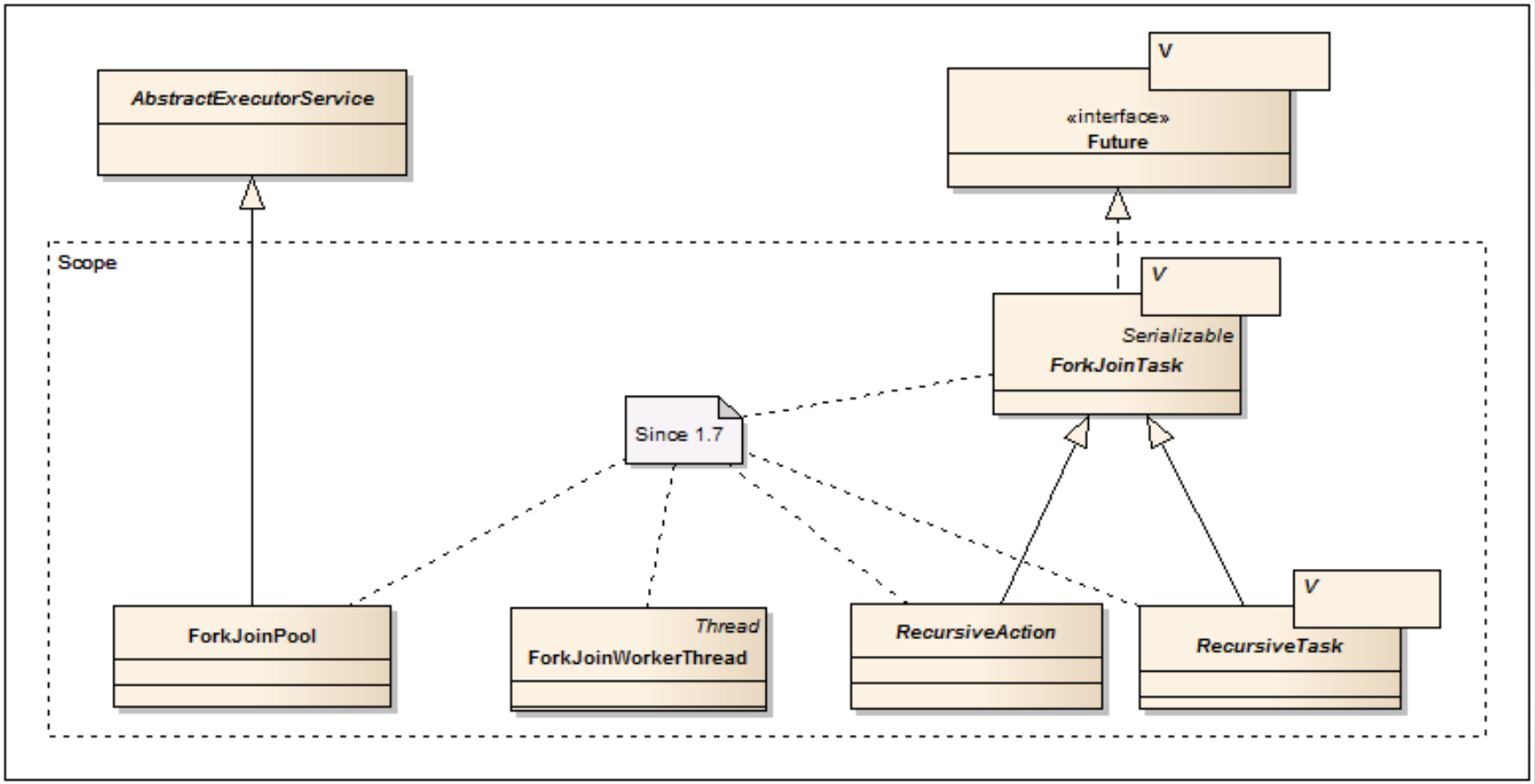
Несколько стандартных реализаций находятся в классе ThreadPoolExecutor:

- CallerRunsPolicy — запускает задачу в вызывающем потоке;
- AbortPolicy — кидает exception;
- DiscardPolicy — игнорирует задачу;
- DiscardOldestPolicy — удаляет самую старую незапущенную задачу из очереди, затем пытается добавить новую задачу еще раз

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

4. Executors

Fork Join



ForkJoinPool (jdk7)

Представляет собой точку входа для запуска корневых (main) ForkJoinTask задач.

- Подзадачи запускаются через методы задачи, от которой нужно отделиться (fork).
- По умолчанию создается пул потоков с количеством потоков равным количеству доступных для JVM процессоров (cores)

ForkJoinTask (jdk7)

Базовый класс для всех Fork Join задач.

Ключевые методы:

- fork() — добавляет задачу в очередь текущего потока ForkJoinWorkerThread для асинхронного выполнения;
- invoke() — запускает задачу в текущем потоке;
- join() — ожидает завершения подзадачи с возвращением результата;
- invokeAll(...) — объединяет все три предыдущие предыдущие операции, выполняя две или более задач за один заход;
- adapt(...) — создает новую задачу ForkJoinTask из Runnable или Callable объектов.

RecursiveTask и RecursiveAction (jdk7)

RecursiveTask - Абстрактный класс от ForkJoinTask, с объявлением метода compute, в котором должна производиться асинхронная операция в наследнике

RecursiveAction отличается от RecursiveTask тем, что не возвращает результат

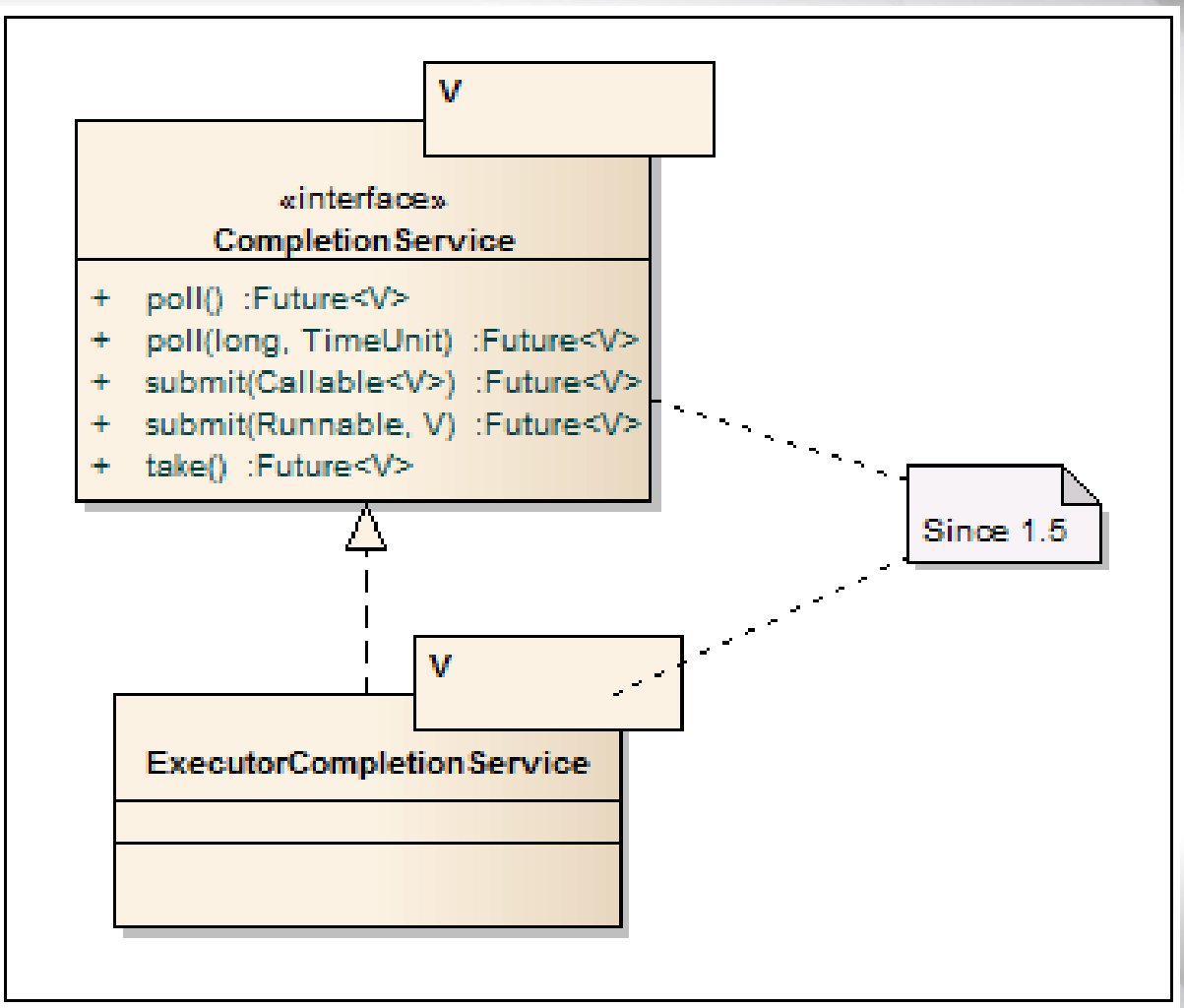
ForkJoinWorkerThread (jdk7)

Используется в качестве имплементации по умолчанию в ForkJoinPoll.

При желании можно описать наследника и перегрузить методы инициализации и завершения worker потока

4. Executors

Completion Service



CompletionService

Интерфейс сервиса с развязкой запуска асинхронных задач и получением результатов.

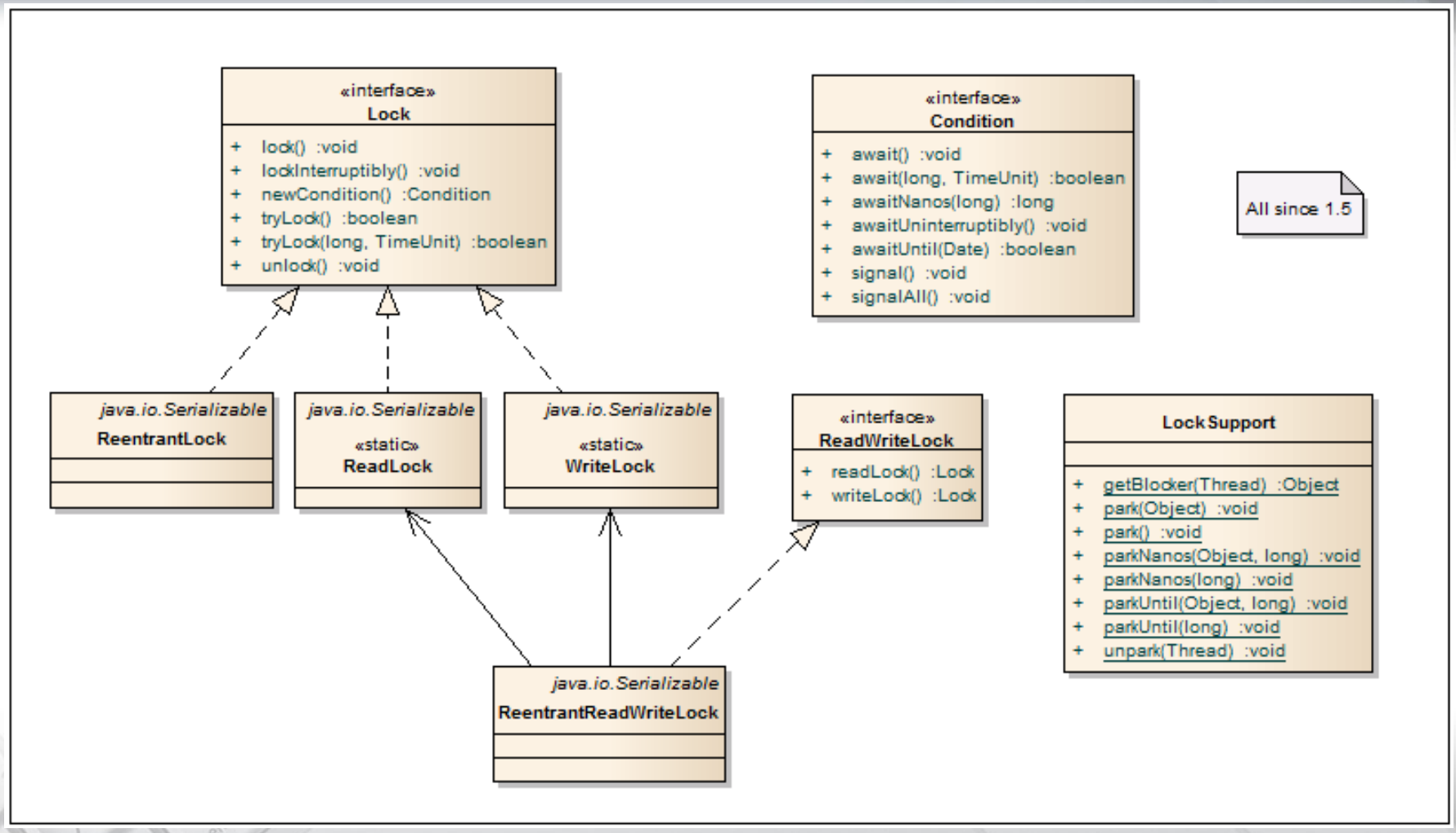
- для добавления задач используются методы submit,
- для вытаскивания результатов завершенных задач используются блокирующий метод take и неблокирующий poll

ExecutorCompletionService

По сути является надстройкой над любым классом, реализующим интерфейс Executor, например ThreadPoolExecutor или ForkJoinPool.

- Используется преимущественно тогда, когда хочется абстрагироваться от способа запуска задач и контроля за их исполнением.
- Если есть завершенные задачи — вытаскиваем их, если нет — ждем в take пока что-нибудь не завершится.
- В основе сервиса по умолчанию используется LinkedBlockingQueue, но может быть передана и любая другая реализация BlockingQueue

5. Locks


«interface»
Lock
+ lock() :void
+ lockInterruptibly() :void
+ newCondition() :Condition
+ tryLock() :boolean
+ tryLock(long, TimeUnit) :boolean
+ unlock() :void
«interface»
Condition
+ await() :void
+ await(long, TimeUnit) :boolean
+ awaitNanos(long) :long
+ awaitUninterruptibly() :void
+ awaitUntil(Date) :boolean
+ signal() :void
+ signalAll() :void
All since 1.5
java.io.Serializable
ReentrantLock
java.io.Serializable
«static»
ReadLock
java.io.Serializable
«static»
WriteLock
«interface»
ReadWriteLock
+ readLock() :Lock
+ writeLock() :Lock
Lock Support
+ getBlocker(Thread) :Object
+ park(Object) :void
+ park() :void
+ parkNanos(Object, long) :void
+ parkNanos(long) :void
+ parkUntil(Object, long) :void
+ parkUntil(long) :void
+ unpark(Thread) :void
java.io.Serializable
ReentrantReadWriteLock

Condition

Интерфейс, который описывает альтернативные методы стандартным wait/notify/notifyAll.

- Объект с условием чаще всего получается из локов через метод lock.newCondition().
- Тем самым можно получить несколько комплектов wait/notify для одного объекта

Lock

Базовый интерфейс из lock framework, предоставляющий более гибкий подход по ограничению доступа к ресурсам/блокам нежели при использовании synchronized.

- При использовании нескольких локов, порядок их освобождения может быть произвольный.
- Имеется возможность пойти по альтернативному сценарию, если лок уже кем то захвачен

ReentrantLock

Лок на вхождение. Только один поток может зайти в защищенный блок.

- Класс поддерживает «честную» (fair) и «нечестную» (non-fair) разблокировку потоков.
- При «честной» разблокировке соблюдается порядок освобождения потоков, вызывающих lock().
- При «нечестной» разблокировке порядок освобождения потоков не гарантируется, но, как бонус, такая разблокировка работает быстрее.
- По умолчанию, используется «нечестная» разблокировка.

ReadWriteLock

Дополнительный интерфейс для создания read/write локов.

Такие локи необычайно полезны, когда в системе много операций чтения и мало операций записи

ReentrantReadWriteLock

Очень часто используется в многопоточных сервисах и кешах, показывая очень хороший прирост производительности по сравнению с блоками synchronized.

По сути, класс работает в 2-х взаимоисключающих режимах: много reader'ов читают данные в параллель и когда только 1 writer пишет данные

ReentrantReadWriteLock

ReentrantReadWriteLock.ReadLock

Read lock для reader'ов, получаемый через readWriteLock.readLock()

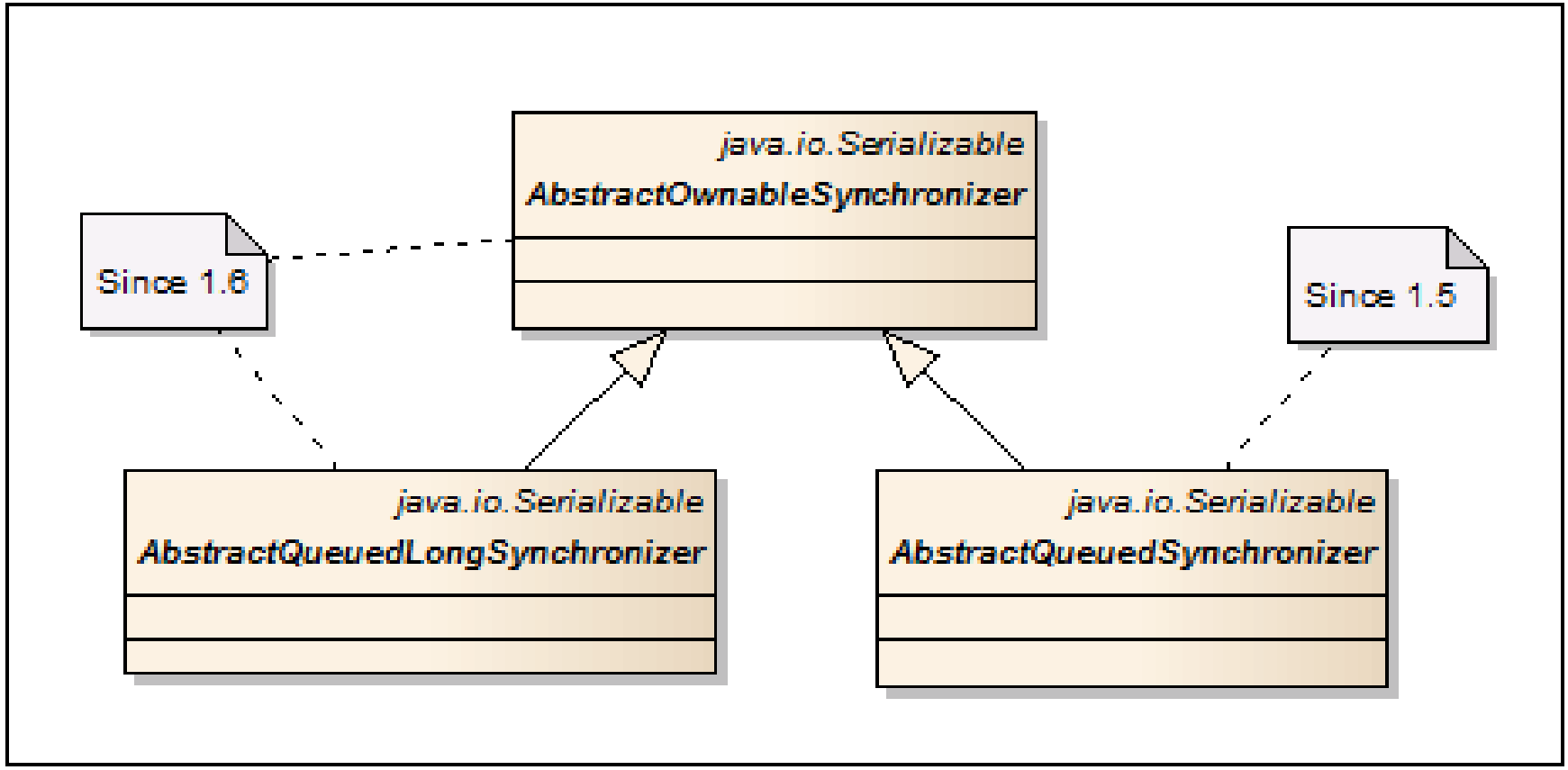
ReentrantReadWriteLock.WriteLock

Write lock для writer'ов, получаемый через readWriteLock.writeLock()

LockSupport

Предназначен для построения классов с локами.

Содержит методы для парковки потоков вместо устаревших методов Thread.suspend() и Thread.resume()


java.io.Serializable
AbstractOwnableSynchronizer
Since 1.6
Since 1.5
java.io.Serializable
AbstractQueuedLongSynchronizer
java.io.Serializable
AbstractQueuedSynchronizer

AbstractOwnableSynchronizer

Базовый класс для построения механизмов синхронизации.

- Содержит всего одну пару геттер/сеттер для запоминания и чтения эксклюзивного потока, который может работать с данными

AbstractQueuedSynchronizer

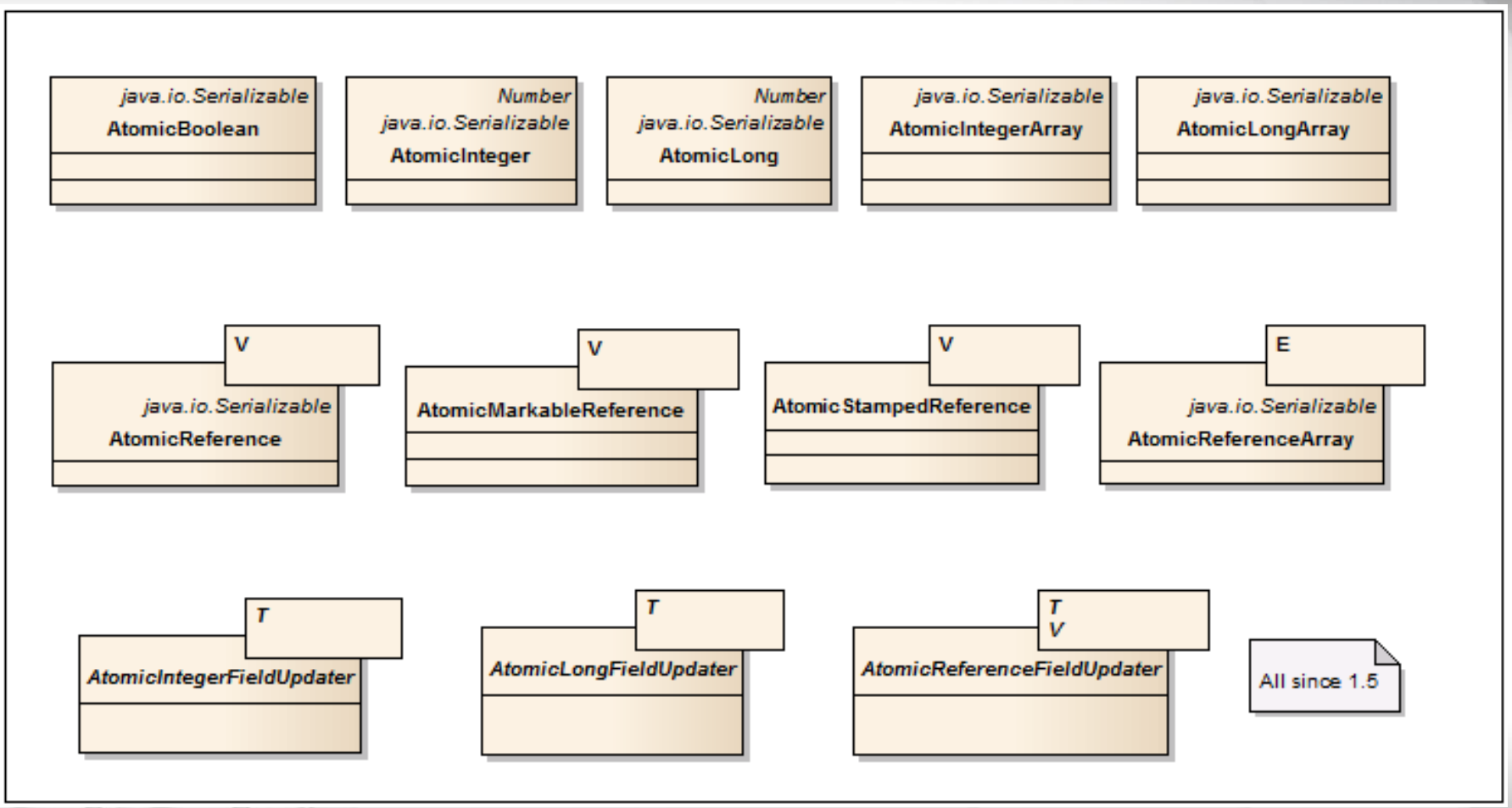
- Используется в качестве базового класса для механизма синхронизации в FutureTask, CountDownLatch, Semaphore, ReentrantLock, ReentrantReadWriteLock.
- Может применяться при создании новых механизмов синхронизации, полагающихся на одиночное и атомарное значение int

AbstractQueuedLongSynchronizer

Разновидность AbstractQueuedSynchronizer, которая поддерживает атомарное значение long

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

6. Atomics



6. Atomics

AtomicBoolean, AtomicInteger, AtomicLong, AtomicIntegerArray, AtomicLongArray

- Что если в классе нужно синхронизировать доступ к одной простой переменной типа int?
- Можно использовать конструкции с synchronized, а при использовании атомарных операций set/get, подойдет также и volatile.
- Но можно поступить еще лучше, использовав новые классы Atomic*.
- За счет использования CAS, операции с этими классами работают быстрее, чем если синхронизироваться через synchronized/volatile.
- Плюс существуют методы для атомарного добавления на заданную величину, а также инкремент/декремент

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

6. Atomics

AtomicReference

Класс для атомарных операцией с ссылкой на объект

AtomicMarkableReference

Класс для атомарных операцией со следующей парой полей: ссылка на объект и битовый флаг (true/false)

AtomicStampedReference

Класс для атомарных операцией со следующей парой полей: ссылка на объект и int значение

AtomicReferenceArray

Массив ссылок на объекты, который может атомарно обновляться

6. Atomics

AtomicIntegerFieldUpdater,
AtomicLongFieldUpdater,
AtomicReferenceFieldUpdater

Классы для атомарного обновления полей по их именам через reflection. Смещение полей для CAS определяется в конструкторе и кешируются, т.ч. тут нет сильного падения производительности из за reflection

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Многопоточное программирование на Java

Часть 2: Пакет java.util.concurrent

Беркунский Е.Ю., кафедра ИУСТ, НУК
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua