Pro Java №29

В Java есть несколько классов и интерфейсов, связанных с выполнением задач в многопоточной среде. Интерфейс `Executor` и его реализации позволяют управлять выполнением асинхронных задач.

```
ExecutorService executor = Executors.newSingleThreadExecutor();
```

`SingleThreadExecutor` создает один поток, который последовательно выполняет задачи по мере их поступления. Это удобно, когда нужно обеспечить выполнение задач в строго определенном порядке.

Метод `execute()` интерфейса `Executor` используется для передачи задачи в пул потоков для асинхронного выполнения. Он принимает объект, реализующий интерфейс `Runnable`, и добавляет его в очередь задач для выполнения.

Метод `shutdown()` используется для корректного завершения работы пула потоков. Когда вызывается метод `shutdown()`, пул перестает принимать новые задачи, но при этом позволяет завершить выполнение уже запущенных задач. Это необходимо для корректного завершения работы приложения и предотвращения потенциальных утечек ресурсов.

Пул потоков с фиксированным числом потоков (`FixedThreadPool`) поддерживает определенное количество потоков, которые выполняются одновременно. Если все потоки заняты, новые задачи будут ожидать в очереди до тех пор, пока один из потоков не освободится.

Класс `Executors` предоставляет фабричный метод для создания пула потоков с динамическим увеличением числа потоков. `CachedThreadPool` создает потоки по мере необходимости и повторно использует ранее созданные потоки, если они доступны. Потоки, которые не используются более 60 секунд, автоматически завершаются и удаляются из пула. `CachedThreadPool` идеально подходит для циклов, где требуется постоянно запускать новые потоки. Этот пул потоков динамически создает потоки по мере необходимости и повторно использует ранее созданные потоки, если они доступны.

ScheduledThreadPool позволяет запускать потоки с определенной задержкой. Размер ядра пула (`core pool size`) определяет, сколько потоков будут храниться в ожидании дальнейших действий без

автоматического удаления через минуту бездействия. Пока один из основных потоков не завершит работу, новый поток не будет запущен.

Интерфейс `Callable` из пакета `java.util.concurrent` используется для выполнения задач, которые возвращают результат и могут выбрасывать исключения. Основное отличие от интерфейса `Runnable` заключается в том, что метод `call()` интерфейса `Callable` возвращает значение, тогда как метод `run()` интерфейса `Runnable` не возвращает значения. Метод `submit()` используется для передачи задачи в пул потоков, а метод `get()` объекта `Future` позволяет получить результат выполнения задачи.

Интерфейс `Future` из пакета `java.util.concurrent` используется для представления результата асинхронной задачи, которая может быть завершена в будущем.

`ConcurrentHashMap` из пакета `java.util.concurrent` представляет собой потокобезопасную реализацию интерфейса `Map`.

Из обычных коллекций можно сделать потокобезопасные через `Collections.synchronized*()`. Однако работа потоков с `ConcurrentHashMap` будет происходить быстрее, так как она не блокирует всю коллекцию, а лишь части.