

Модуль 03 - Бассейн Java

Потоки

Резюме: Сегодня вы узнаете, как использовать основные механизмы многопоточности в Java.

Содержание

I Предисловие

II Инструкции

III Упражнение 00: Яйцо, курица... или человек?

IV Упражнение 01: Яйцо, Курица, Яйцо, Курица...

V Упражнение 02: Реальная многопоточность

VI Упражнение 03: Слишком много потоков...

Предисловие

- Любое современное клиент-серверное приложение основано на потоках.
- Потоки реализуют концепцию асинхронных операций, когда несколько слабо связанных задач выполняются «параллельно».
- Многопоточность в клиент-серверных приложениях позволяет переводить некоторые задачи в фоновый режим выполнения, чтобы клиенту не приходилось ждать ответа сервера.
Например, после того, как вы указали свой адрес электронной почты на веб-сайте, сразу же отображается страница, информирующая о том, что сообщение с подтверждением было отправлено на ваш адрес электронной почты, независимо от того, сколько времени потребуется, чтобы отправить сообщение на ваш адрес электронной почты в параллельном потоке.
- Каждый ваш запрос на веб-сайте выполняется в отдельном независимом потоке на сервере.
- Поведением потоков управляет операционная система и процессор.
- Поведение потоков недетерминировано. Вы никогда не знаете, какой поток будет запущен в конкретный момент, даже если вы перезапустите тот же многопоточный код.
- Советы по работе с потоками можно найти в классе Object.
- Темы — любимая тема на собеседованиях с джуниорами.


Глава II

Инструкции

- Используйте эту страницу как единственную ссылку. Не слушайте никаких слухов и домыслов о том, как приготовить свой раствор.
- Теперь у вас есть только одна версия Java, 1.8. Убедитесь, что на вашем компьютере установлены компилятор и интерпретатор этой версии.
- Вы можете использовать IDE для написания и отладки исходного кода.
- Код читается чаще, чем пишется. Внимательно прочитайте документ, в котором приведены правила форматирования кода. При выполнении каждой задачи убедитесь, что вы следуете общепринятым стандартам Oracle.
- Комментарии в исходном коде вашего решения запрещены. Они затрудняют чтение кода.
- Обратите внимание на разрешения ваших файлов и каталогов.
- Для оценки ваше решение должно находиться в вашем репозитории GIT.
- Ваши решения будут оценены вашими товарищами.
- Вы не должны оставлять в своем каталоге никаких других файлов, кроме тех, которые явно указаны в инструкциях к упражнению. Рекомендуется изменить ваш .gitignore, чтобы избежать несчастных случаев.
- Когда вам нужно получить точный вывод в ваших программах, запрещается отображать предварительно рассчитанный вывод вместо правильного выполнения упражнения.
- Есть вопрос? Спросите у соседа справа. В противном случае попробуйте с соседом слева.
- Ваш справочник: гугля/интернет/гугл. И еще кое-что. На Stackoverflow есть ответ на любой вопрос, который у вас может возникнуть. Научитесь правильно задавать вопросы.
- Внимательно прочитайте примеры. Они могут требовать вещи, которые иначе не указаны в теме.
- Используйте "System.out" для вывода

Глава III

Упражнение 00: Яйцо, курица... или человек?

	Exercise 00
Egg, Hen... or Human?	
Turn-in directory : <i>ex00/</i>	
Files to turn in : *.java	
Allowed functions : All Recommended types and their methods : Object, Thread, Runnable	

Истина рождается в споре — допустим, каждая ветка дает свой ответ. Тема, за которой последнее слово, правильная.

Вам нужно реализовать работу двух потоков. Каждый из них должен отобразить свой ответ несколько раз, например, 50:

```
$ java Program --count=50
Egg
Hen
Hen
Hen
Hen
...
Egg
```

В этом случае выигрывает яичная нить. Однако программа также содержит основной поток. Внутри потока выполняется метод `public static void main(String args[])`.

Нам нужно, чтобы этот поток отображал все свои ответы в конце выполнения программы.


Таким образом, окончательный вариант выглядит следующим образом:

```
$ java Program --count=50
Egg
Hen
Hen
...
Egg
Hen
...
Human
...
...
Human
```

- Это означает, что программа выводит сообщение Human 50 раз, которое печатает основной поток.

Глава IV

Упражнение 01: Яйцо, Курица, Яйцо, Курица...

	Exercise 01
Egg, Hen, Egg, Hen...	
Turn-in directory : <i>ex01/</i>	
Files to turn in : *.java	
Allowed functions : All	
Recommended types and their methods : Object, Thread, Runnable	
Keywords : Synchronized	

Давайте организуем спор.

Теперь каждый поток может предоставить свой ответ только после того, как это сделает другой поток.

Предположим, что поток "EGG" всегда печатает первым,


```
$ java Program --count=50
Egg
Hen
Egg
Hen
Egg
Hen
...
```

Примечание:

- Для решения этой задачи рекомендуем изучить принцип работы модели [Producer-Consumer](#).

Глава V

Упражнение 02. Реальная многопоточность

	Exercise 02
Real Multithreading	
Turn-in directory : <i>ex02/</i>	
Files to turn in : *.java	
Allowed functions : All	
Recommended types and their methods : Object, Thread, Runnable	
Keywords : Synchronized	

Попробуйте использовать многопоточность по прямому назначению: распределите вычисления по программе.

Предположим, что имеется массив целочисленных значений.

Ваша цель — вычислить сумму элементов массива, используя несколько «суммирующих» потоков.

Каждый поток вычисляет определенный раздел внутри массива.

Количество элементов в каждом разделе постоянно, кроме последнего (его размер может отличаться в большую или меньшую сторону).

Массив должен генерироваться случайным образом каждый раз.

Длина массива и количество потоков передаются в качестве аргументов командной строки.

Чтобы убедиться, что программа работает корректно, начнем с вычисления суммы элементов массива стандартным методом.

Максимальное количество элементов массива — 2 000 000.

Максимальное количество потоков не больше текущего количества элементов массива.

Максимальное значение по модулю каждого элемента массива равно 1000.

Все данные гарантированно верны.

Пример работы программы (каждый элемент массива равен 1):


```
$ java Program --arraySize=13 --threadsCount=3
Sum: 13
Thread 1: from 0 to 4 sum is 5
Thread 2: from 5 to 9 sum is 5
Thread 3: from 10 to 12 sum is 3
Sum by threads: 13
```

Примечание:

- В приведенном выше примере размер последней секции суммирования, используемой третьим потоком, меньше остальных.
- Потоки могут несогласованно выводить результаты работы.

Глава VI

Упражнение 03. Слишком много потоков...

	Exercise 03
Too Many Threads...	
Turn-in directory : <i>ex03/</i>	
Files to turn in : *.java	
Allowed functions : All	
Recommended types and their methods : Object, Thread, Runnable	
Keywords : Synchronized	

Предположим, что нам нужно скачать список файлов из сети. Одни файлы загружаются быстрее, другие медленнее.

Очевидно, что для реализации этой функциональности мы можем использовать многопоточную загрузку, при которой каждый поток загружает определенный файл. Но что делать, если файлов слишком много? Одновременно нельзя запускать большое количество потоков. Поэтому многие из них будут ждать.

Кроме того, мы должны помнить, что постоянное создание и завершение потоков — очень дорогостоящая операция, которой следует избегать. Разумнее запускать *N* потоков одновременно, и когда любой из них закончит загрузку файла, он может взяться за следующий файл в очереди.

очередь.

Нам необходимо создать файл `files_urls.txt` (имя файла должно быть явно указано в коде программы), в котором вы указываете список адресов файлов для скачивания, например:

```
1 https://i.pinimg.com/originals/11/19/2e/11192eba63f6f3aa591d3263fdb66bd5.jpg
2 https://pluspng.com/img-png/balloon-hd-png-balloons-png-hd-2750.png
3 https://i.pinimg.com/originals/db/a1/62/dba162603c71cac00d3548420c52bac6.png
4 https://pngimg.com/uploads/balloon/balloon_PNG4969.png
5 http://tldp.org/LDP/intro-linux/intro-linux.pdf
```

Пример работы программы:

```
$ java Program.java --threadsCount=3
Thread-1 start download file number 1
Thread-2 start download file number 2
Thread-1 finish download file number 1
Thread-1 start download file number 3
Thread-3 start download file number 4
Thread-1 finish download file number 3
Thread-2 finish download file number 2
Thread-1 start download file number 5
Thread-3 finish download file number 4
Thread-1 finish download file number 5
```

Заметки:

- Результат, созданный реализованной программой, может отличаться от показанного на рисунке.
- Каждый файл загружается только один раз одним потоком.
- Программа может содержать «бесконечный цикл» без условия выхода (в этом случае программу можно остановить, прервав процесс).