Laboratory Work #30

Java Threads. Synchroniza-tion. The Singleton Pattern

LEARN. GROW. SUCCEED.

® 2020-2021. Department: <Software of Information Systems and Technologies>
Faculty of Information Technology and Robotics
Belarusian National Technical University
by Viktor Ivanchenko / ivanvikvik@bntu.by / Minsk

ЛАБОРАТОРНАЯ РАБОТА #30

Основы многопоточного программирования на Java. Синхронизация

Цель работы 

Изучить фундаментальные концепции параллельного программирования на Java с использованием потоков выполнения (*threads*) и закрепить приобретённые знания и навыки на примере разработки интерактивных многопоточных приложений.

Требования 

1) Создать многопоточное приложение на Java согласно индивидуальному варианту задания. Организовать изменение приоритетов потоков, а также предусмотреть их объединение в группы приоритетов с целью анализа влияния приоритета на выполнение действий потоками.
2) Спроектировать UML-диаграмму классов и интерфейсов, составляющих архитектуру приложения.
3) При проектировании и реализации программы рекомендуется использовать архитектурный шаблон MVC, а также фундаментальные SOLID и GRASP принципы.
4) Классы и другие сущности программы должны быть грамотно структурированы по соответствующим пакетам и иметь отражающую их функциональность названия.
5) Любая сущность, желающая получить доступ к разделяемым ресурсам, должна быть потоком.
6) Приложение должно корректно работать с общими разделяемыми ресурсами и избегать ситуации взаимной блокировки.
7) Необходимо гарантировать, чтобы общий ресурс в системе был только в одном экземпляре.

8) Реализовать неблокирующую синхронизацию доступа к общим ресурсам. Для чего рекомендуется использовать компоненты из библиотек ***java.util.concurrent*** и ***java.util.concurrent.locks***.
9) Для генерирования случайных чисел воспользуйтесь методами объекта класса ***java.util.Random***.
10) Программа должна быть снабжена дружелюбным и интуитивно понятным интерфейсом. Работа с консолью должна быть минимальной.
11) Приложение должно корректно обрабатывать любые исключительные ситуации, которые могут возникнуть в процессе работы программы, а также вести журналирование данных ситуаций. В качестве логгера рекомендуется использовать библиотеку логгирования Apache Log4j.
12) Для подтверждения работоспособности и адекватности программы, вся модель проекта должна быть покрыта соответствующими модульными тестами. Для модульного (*unit*) тестирования рекомендуется использовать тестовый фреймворк jUnit версии 4.0 и младше.
13) Необходимо по максимуму пытаться разрабатывать универсальный код.
14) Программа должна обязательно быть снабжена комментариями на английском языке, в которых необходимо указать краткое предназначение программы, номер лабораторной работы и её название, версию программы, ФИО разработчика, номер группы и дату разработки.
15) Исходный текст классов и демонстрационной программы рекомендуется также снабжать комментариями.
16) При разработке программ придерживайтесь соглашений по написанию кода на JAVA (Java Code-Convention*)*.

Индивидуальное задание

1. **Морской Порт *(Seaport)***. Корабли заходят в порт для разгрузки/загрузки. Работает несколько причалов. У одного причала может стоять один корабль. Корабль может загружаться у причала, разгружаться или выполнять оба действия.
2. **Маленькая библиотека *(Small Library)***. Доступны для чтения несколько книг. Одинаковых книг в библиотеке нет. Некоторые выдаются на руки, некоторые только в читальный зал. Читатель может брать на руки и в читальный зал несколько книг.
3. **Автостоянка *(Car Park)***. Доступно несколько машиномест. На одном месте может находиться только один автомобиль. Если все места заняты, то автомобиль не станет ждать больше определенного времени и уедет на другую стоянку.
4. **Центр поддержки и обслуживания клиентов *(Call-center)***. В организации работает несколько операторов. Оператор может обслуживать только одного клиента, остальные должны ждать в очереди. Клиент может положить трубку.
5. **Автобусные остановки *(Bus Stations)***. На маршруте несколько остановок. На одной остановке может останавливаться несколько автобусов одновременно, но не более заданного числа.
6. **Свободная касса *(Free Desk)***. В ресторане быстрого обслуживания есть несколько касс. Посетители стоят в очереди в конкретную кассу, но могут перейти в другую очередь при уменьшении или исчезновении там очереди.
7. **Тоннель *(Tunnel)***. В горах существует два тоннеля, в которых поезда могут двигаться в обоих направлениях. По обоим концам тоннеля собралось много поездов. Обеспечить безопасное прохождение тоннелей.

Best of LUCK with it, and remember to HAVE FUN while you're learning :)
Victor Ivanchenko

Контрольные вопросы 

1) Концепция многопоточности и понятие потока выполнения? Чем отличается поток от процесса?
2) Модель параллельного (многопоточного) программирования в Java.
3) Приведите общее описание сущности-потока в Java и его характеристики (свойства).
4) Что такое главный (дочерний) поток? Как получить ссылку на главный поток (или любой выполняющий поток)?
5) В каких состояниях может находится поток? Какая сущность отвечает за хранение состояния потока?
6) Опишите жизненный цикл сущности-потока.
7) На что влияет приоритет потока, как его изменить?
8) Опишите основные методы для управления потоком?
9) Способы создания потоков в Java, их преимущества и недостатки?
10) Зачем нужны группы потоков? Как их создать?
11) Чем отличаются интерактивные потоки от потоков-демонов (фоновых потоков)?
12) Как создать и запустить на выполнение демона?
13) Опишите архитектуру внутреннего мира JVM с точки зрения поток.
14) Что такое синхронизация потоков? Понятие монитора?
1. Как синхронизация реализуется в Java?
2. Работа с ключевым словом ***synchronized***, синхронизированные методы, синхронизированные блоки.
15) Что такое блокирующая и неблокирующая синхронизация? Их реализации?
16) Что такое взаимная блокировка (***deadlock***) и при каких обстоятельствах она происходит?
17) Что такое атомарность выполнения операции?
18) Для чего и как в Java используется ключевое слово ***volatile***?
19) Особенности работы с классом Thread и интерфейсом Runnable.
20) Методы isAlive() и getState() класса Thread.

21) Что такое “главный поток”, как получить ссылку на главный поток выполнения программы?
22) Метод join() класса Thread.
23) На что влияет приоритет потока, методы для работы с приоритетом потока.
24) Потоки демоны: назначение, создание, случаи применения.
25) Группы потоков: создание группы потоков, методы класса ThreadGroup.
26) Обработка не отловленных исключительных ситуаций, возникающих при работе потока.
27) Организация связи между потоками: назначение и работа с методами wait, notify.
28) Приостановка и возобновление работы с потоками: варианты реализации для Java 1 (suspend, resume) и Java 2.
29) Назначение пакетов java.util.concurrent, java.util.concurrent.locks, java.util.concurrent.atomic.
30) Исполнители (executors) в java.util.concurrent.
31) Механизм управления мьютексами Lock в java.util.concurrent.
32) Работа с Atomic-классами в java.util.concurrent.
33) Работа с синхронизированными коллекциями в java.util.concurrent.