Частное учреждение образования
Колледж бизнеса и права

УТВЕРЖДАЮ
Заведующий
методическим кабинетом
_____Е.В. Паскал
«___»_____2021

Специальность: 2-40 01 01 «Программное обеспечение информационных технологий»	Дисциплина: «Основы кроссплатформенного программирования»

ЛАБОРАТОРНАЯ РАБОТА № 10
Инструкционно-технологическая карта

Тема: «Параллельное программирование на языке Java»

Цель: Научиться реализовывать механизм параллельного программирования, научиться ставить задачи в очередь в Java

Время выполнения: 4 часа

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить теоретические сведения;
2. Ответить на контрольные вопросы;
3. Откомпилировать примеры программ из раздела «Теоретические сведения»;
4. Выполнить ИДЗ.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ ДЛЯ ВЫПОЛНЕНИЯ РАБОТЫ

Параллельное программирование – это означает, что задачи выполняются одновременно, но под капотом система может действительно переключаться между задачами. Смысл параллельного программирования в том, что оно выгодно даже на однопроцессорной машине.

Параллельное программирование на однопроцессорной машине:

1. Предположим, что пользователю нужно загрузить пять изображений, и каждое изображение приходит с другого сервера, и каждое изображение занимает пять секунд, и теперь предположим, что пользователь загружает все первые изображения, это занимает 5 секунд, а затем все вторые изображения, это занимает еще 5 секунд и так далее, и к концу времени это заняло 25 секунд. Это быстрее, чтобы загрузить немного изображения

первого, затем немного изображения два, три, четыре, пять, а затем вернуться и немного изображения первого и так далее.

Tasks overlap in time

Task1 ------ ------ ------ ------

Task2 ------ ------ ------ ------

--->

Time

Если для каждого из них требуется 5 секунд и разбить его на маленькие кусочки, общая сумма по-прежнему составляет 25 секунд. Тогда почему быстрее загружать его одновременно.

Это происходит потому что, когда вызывается изображение с первого сервера, и это занимает 5 секунд, не потому, что максимальная пропускная способность входящего трафика увеличена, а потому, что серверу требуется время, чтобы отправить его пользователю. В основном пользователь сидит в ожидании большую часть времени. Таким образом, пока пользователь ожидает первое изображение, он может начать загрузку второго изображения. Таким образом, если сервер работает медленно, выполняя его в нескольких потоках одновременно, можно загружать дополнительные изображения без особого дополнительного времени.

Теперь, в конце концов, если вы загружаете много изображений одновременно, входящая пропускная способность может быть максимально увеличена, и тогда добавление большего количества потоков не ускорит его, но до некоторой степени это отчасти бесплатно.

Помимо скорости, еще одним преимуществом является снижение задержки. Выполнение небольшого количества за один раз уменьшает задержку, поэтому пользователь может видеть некоторую обратную связь по мере развития событий.

Необходимость параллельного программирования

- Потоки полезны только тогда, когда задача относительно велика и в значительной степени самодостаточна. Когда пользователю необходимо выполнить лишь небольшое количество комбинаций после большого количества отдельной обработки, возникают некоторые издержки при запуске и использовании потоков. Так что, если задача действительно мала, никто никогда не получит деньги за накладные расходы.
- Также, как упоминалось выше, потоки наиболее полезны, когда пользователи ждут. Например, когда один ожидает один сервер, другой может читать с другого сервера.

Основные шаги для параллельного программирования

1.Во-первых, поставить задачу в очередь. Служба исполнителя вызовов устанавливает новый фиксированный пул потоков и предоставляет размер. Этот размер указывает максимальное количество одновременных задач. Например, если добавить в очередь тысячу элементов, но размер пула равен 50, то только 50 из них будут работать одновременно. Только когда один из первых пятидесяти завершит выполнение, 51-й будет принят к

исполнению. Число типа 100 в качестве размера пула не будет перегружать систему.

2. ExecutorService taskList = Executors.newFixedThreadPool(poolSize);

3. Затем пользователь должен поместить некоторые задачи выполняемого типа в очередь задач. Runnable — это всего лишь один интерфейс, который имеет один метод, называемый run. Система вызывает метод run в соответствующее время, когда он переключается между задачами, запуская отдельный поток.

taskList.execute(someRunnable)

4. Метод Execute немного ошибочен, потому что, когда задача добавляется к задаче в очереди, созданной выше, с исполнителями, усеивающими новый фиксированный пул потоков, она не обязательно начинает выполнять ее сразу. Он начинает выполняться, когда один из тех, кто выполняет одновременно (размер пула), заканчивает выполнение.

Существует пять различных подходов для реализации параллельного программирования с различными преимуществами и недостатками.

Подход первый: отдельный класс, который реализует Runnable

1. Первое, что нужно сделать, — это создать отдельный класс и совершенно отдельный класс, который реализует работающий интерфейс.

2. public class MyRunnable implements Runnable {
public void run() { ... }
}

4. Во-вторых, сделайте несколько экземпляров основного класса и передайте их для выполнения. Давайте применим этот первый подход к созданию потоков, которые просто учитываются. Таким образом, каждый поток будет печатать имя потока, номер задачи и значение счетчика.

5. После этого используйте метод паузы, чтобы сидеть в ожидании, чтобы система переключалась вперед и назад. Таким образом, операторы печати будут чередоваться.

6. Передайте аргументы конструктора конструктору Runnable, чтобы разные экземпляры учитывались разное количество раз.

7. Вызов метода shutdown означает отключение потока, который наблюдает, чтобы увидеть, были ли добавлены какие-либо новые задачи или нет.

Практическая реализация

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
public class Counter implements Runnable
{
   private final MainApp mainApp;
   private final int loopLimit;
   private final String task;
     // Конструктор для получения ссылки на основной класс
   public Counter
        (MainApp mainApp, int loopLimit, String task)
   {
```

```
        this.mainApp = mainApp;
        this.loopLimit = loopLimit;
        this.task = task;
    }
     // Печатает имя потока, номер задачи и
    // значение счетчика
    // Вызывает метод pause, чтобы обеспечить многопоточность
    @Override
    public void run()
    {
        for (int i = 0; i < loopLimit; i++)
        {
            System.out.println("Thread: " +
            Thread.currentThread().getName() + " Counter: "+ (i + 1) + " Task: " + task);
            mainApp.pause(Math.random());
        }
    }
}
class MainApp
{
    // Запускает темы. Размер бассейна 2 означает в любое время
    // может быть только два потока одновременно
    public void startThread()
    {
        ExecutorService taskList = Executors.newFixedThreadPool(2);
        for (int i = 0; i < 5; i++)
        {
            // Делает задачи доступными для выполнения.
            // В нужное время звонки запускаются
            // метод работающего интерфейса
            taskList.execute(new Counter(this, i + 1,"task " + (i + 1)));
        }
          // закрывает поток, который смотрит
        // вы добавили новые задачи.
        taskList.shutdown();
    }
     // Приостановить выполнение на мгновение
    // чтобы система переключалась назад и вперед
    public void pause(double seconds)
    {
        try
        {
            Thread.sleep(Math.round(1000.0 * seconds));
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }
     // Метод драйвера
    public static void main(String[] args)
```

```
    {
        new MainApp().startThread();
    }
}
```

Преимущества:

- Слабая связь: поскольку отдельный класс можно использовать повторно, он способствует слабой связи.
- Конструкторы: Аргументы могут быть переданы конструкторам для разных случаев. Например, описание различных ограничений цикла для потоков.
- Условия гонки: если данные были переданы, маловероятно, что отдельный класс будет использоваться в качестве подхода, и если у него нет общих данных, то не нужно беспокоиться об условиях гонки.

Недостатки:

Было немного неудобно перезванивать в основное приложение. Ссылка должна передаваться по конструктору, и даже если есть доступ к ссылке, в основном приложении могут быть вызваны только открытые методы (метод pause в данном примере).

Внутренний класс, который реализует Runnable

При таком подходе пользователь физически помещает определение класса, который реализует Runnable, в определение класса основного класса.

```
public class OuterClass{
private class InnerClass implements Runnable{
public void run(){ }}}
```

Затем метод run () помещается во внутренний класс и передается методу execute. Выполнить на самом деле не означает выполнить. Это означает, что доступны для исполнения. Например, если у пользователя размер пула 5, а в очереди есть 10 задач, шестая не начинает выполняться, пока не завершится одна из первых пяти.

```
taskList.execute(new InnerClass());
```

Практическая реализация:

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
  // Java-программа для отображения одновременно
// Программирование в действии
public class OuterClass {
     // Метод драйвера
    public static void main(String[] args)
    {
        new OuterClass().startThreads();
    }
     // Запускает потоки и вызывает метод run
    // метод работающего интерфейса
    private void startThreads()
    {
        ExecutorService taskList = Executors.newFixedThreadPool(2);
         taskList.execute(new InnerClass(1));
```

```
    taskList.execute(new InnerClass(2));
    taskList.execute(new InnerClass(3));
    taskList.execute(new InnerClass(4));
    taskList.execute(new InnerClass(5));
    taskList.shutdown();
}
  // Пауза выполнения позволяет время
// система для переключения вперед и назад
private void pause(double seconds)
{
    try {
        Thread.sleep(Math.round(1000.0 * seconds));
    }
    catch (InterruptedException e) {
        e.printStackTrace();
    }
}
  // Внутренний класс
public class InnerClass implements Runnable {
      private int loopLimit;
      // Конструктор для определения
    // разные ограничения
    InnerClass(int loopLimit)
    {
        this.loopLimit = loopLimit;
    }
      // Печатает имя и значение потока
    // переменной счетчика
    @Override
    public void run()
    {
        for (int i = 0; i < loopLimit; i++) {
            System.out.println(
                Thread.currentThread().getName()
                + " Counter: " + i);
            pause(Math.random());
        }       }    }   }
```

Анонимный Внутренний Класс, который реализует Runnable

Внутренние анонимные классы. Внутренние классы используются так часто в приложениях, что разработчики часто хотят сократить синтаксис с помощью внутренних анонимных классов, где пользователь дает определение класса и создает экземпляр класса одним махом. Поскольку анонимные внутренние классы являются внутренними классами, они имеют те же плюсы, что и внутренние классы, но они короче и более лаконичны. Недостатки, которые приходят с ними, заключаются в том, что они немного запутывают новичков, нет конструкторов и их нельзя использовать в других местах.

Практическая реализация:

```
import java.util.concurrent.ExecutorService;
```

```
import java.util.concurrent.Executors;
// Параллельное программирование с использованием
// Анонимный внутренний класс
public class MyClass {
    // Метод драйвера
   public static void main(String[] args)
   {
      new MyClass().startThreads();
   }
     // Запуск потоков с размером пула 2
   private void startThreads()
   {
      ExecutorService taskList = Executors.newFixedThreadPool(2);
       for (int i = 0; i < 5; i++) {
         int finalI = i + 1;
           // Даем определение класса            // и создаем все сразу
         taskList.execute(new Runnable() {
              // Печатает имя и значение потока              // переменной счетчика
            @Override
            public void run()
            {
               for (int j = 0; j < finalI; j++) {
                  System.out.println(
                     Thread
                        .currentThread()
                        .getName()
                     + " Counter:" + j);
                  pause(Math.random());
               }             }          });       }
      taskList.shutdown();
   }
   // Пауза выполнения позволяет время
   // система для переключения вперед и назад
   private void pause(double seconds)
   {
      try {
         Thread.sleep(
            Math.round(1000.0 * seconds));
      }
      catch (InterruptedException e) {
         e.printStackTrace();
      }
   }
}
```

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что означает параллельное программирование?
2. В чем необходимость параллельного программирования?
3. Объясните один из подходов параллельного программирования?
4. Что представляет собой анонимный внутренний класс?

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Задание №1: Об “обедающих философах”

Для реализации потребуется пять процессоров. Суть задачи следующая: пять философов сидят за круглым столом. Они проводят жизнь, чередуя приемы пищи и размышления. В центре стола находится большое блюдо спагетти. Философам, чтобы съесть порцию спагетти, требуется две вилки. Вилок всего пять: между каждой парой философов лежит по одной вилке. Каждому философу дозволительно пользоваться только вилками, которые лежат рядом с ним (слева и справа).

Задача – написать программу, моделирующую поведение философов. Программа должна избегать ситуации, в которой все философы голодны, то есть ни один из них не может взять себе две вилки (например, когда каждый философ держит по одной вилке и не хочет отдавать ее). Раз вилок всего пять, то одновременно могут есть не более, чем двое философов. Два сидящих рядом философа не могут есть одновременно. Предположим, что периоды раздумий и приемов пищи различны – для их имитации в программе можно использовать генератор случайных чисел. Имитация поведения каждого философа может быть разбита на следующие блоки: поразмыслить, взять вилки, поесть, отдать вилки. Вилки являются разделяемым ресурсом.

Запрограммируйте остановку алгоритма по достижении контрольного времени.

ДОМАШНЕЕ ЗАДАНИЕ

Индивидуальное задание

ЛИТЕРАТУРА

У. Савитч, язык Java. Курс программирования, Вильямс, 2017

Преподаватель А.С.Кибисова

Рассмотрено на заседании цикловой комиссии программного обеспечения информационных технологий
Протокол № ____ от «___» _____ 2021
Председатель ЦК _____В.Ю.Михалевич