|  |
| --- |
| **ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ** |
|  |
|  |
|  |
| **Практическое занятие №11.**  **Тема: Разработка многопоточных программ** |
|  |
|  |
|  |

**Цель:** получение практических навыков проектирования и управление многопоточными программами в Java.

**Методические рекомендации**

**Введение в потоки**

Один поток – это одна единица исполнения кода. Каждый поток последовательно выполняет инструкции процесса, которому он принадлежит, параллельно с другими потоками этого процесса. Все потоки в рамках одного процесса (одной виртуальной машины Java) имеют общий набор ресурсов, общую память, общий набор объектов, с которыми могут работать.

Одноядерный процессор может обрабатывать команды только последовательно, по одной за раз (в упрощенном случае). Однако запуск нескольких параллельных потоков возможен и в системах с одноядерными процессорами. В этом случае система будет периодически переключаться между потоками, поочередно давая выполняться то одному, то другому потоку. Такая схема называется псевдопараллелизмом. Система запоминает состояние (контекст) каждого потока, перед тем как переключиться на другой поток, и восстанавливает его по возвращению к выполнению потока. В контекст потока входят такие параметры, как стек, набор значений регистров процессора, адрес исполняемой команды и прочее….

Проще говоря, при псевдопараллельном выполнении потоков процессор переключается между несколькими потоками, выполняя по очереди часть каждого из них (рис.1).

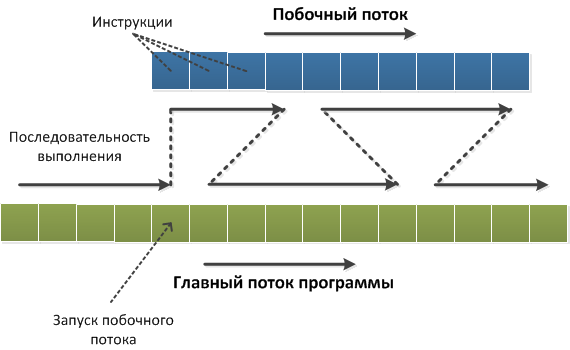


Рис.1. Псевдопараллельное выполнение потоков

Цветные квадраты на рисунке – это инструкции процессора (зеленые инструкции главного потока, синие – побочного). Выполнение идет слева направо. После запуска побочного потока его инструкции начинают выполняться вперемешку с инструкциями главного потока. Количество выполняемых инструкций за каждый подход не определено и зависит от кванта времени, который выделяется в данный момент потоку.

То, что инструкции параллельных потоков выполняются вперемешку, в некоторых случаях может привести к конфликтам – доступа к данным.

Многопоточная система в **Java** построена на основе класса **Thread**, его методах и дополняющем его интерфейсе **Runnable**. Чтобы создать новый поток исполнения, следует расширить класс **Thread** или же реализовать интерфейс **Runnable**.

В классе **Thread** определяется ряд методов, помогающих управлять потоками исполнения. Некоторые из тех методов перечислены ниже.

**getName()** - получить имя потока

**getPriority()** - получить приоритет потока

**isAlive()** - определить, выполняется ли поток

**join()** - ожидать завершение потока

**run()** - запуск потока

**sleep()** - приостановить поток на заданное время

**start()** - запустить поток

Метод **sleep()** вынуждает тот поток, из которого он вызывается, приостановить свое выполнение на указанное количество миллисекунд. Общая форма этого метода выглядит следующим образом:

**static void sleep(long мс) throws InterruptedException**

Метод **sleep()** может сгенерировать исключение типа **InterruptedException.**

Установить имя потока исполнения можно с помощью метода **setName()**. А для того чтобы получить имя потока исполнения, достаточно вызвать метод **getName**().

**Главный поток исполнения**

Когда программа нa **Java** запускается на выполнение, сразу же начинает выполняться один поток. Он обычно называется главным потоком программы, потому что он запускается вместе с ней. Главный поток исполнения важен по двум причинам.

• От этого потока исполнения порождаются все дочерние потоки.

• Зачастую он должен быть последним потоком, завершающим выполнение программы, поскольку в нем производятся различные завершающие действия.

Несмотря на то, что главный поток исполнения создается автоматически при запуске программы, им можно управлять через объект класса **Thread.** Для этого достаточно получить ссылку на него, вызвав метод **currentThread(),** который объявляется как открытый и статический в классе **Thread.** Этот метод возвращает ссылку на тот поток исполнения, из которого он был вызван.

Получив ссылку на главный поток, можно управлять им таким же образом, как и любым другим потоком исполнения. Например.

// Управление главным потоком исполнения

**class CurrentThreadDemo{**

**public static void main(String args[]){**

**Thread t = Thread.currentThread();**

**System.out.println("Teкyщий поток исполнения:" + t);**

// изменение имени потока исполнения

**t.setName("Новый поток");**

**System.out.println("Пocлe изменения имени потока:" + t);**

**try{**

**for(int n = 5; n > О ; n-- ){**

**System.out.println(n);**

**Thread.sleep(1000);**

**}**

**}**

**catch(InterruptedException е)**

**{**

**System.out.println("Глaвный поток исполнения прерван");**

**}**

**}**

**}**

В программе ссылка на текущий поток исполнения (в данном случае - главный поток) получается в результате вызова метода **currentThread()** и сохраняется в локальной переменной **t** . Затем выводятся сведения о потоке исполнения. Далее вызывается метод **setName()** для изменения внутреннего имени потока исполнения. После этого сведения о потоке исполнения выводятся заново. А в следующем далее цикле выводятся цифры в обратном порядке с задержкой на 1 секунду после каждой строки. Пауза организуется с помощью метода **sleep()**. Аргумент метода **sleep()**задает время задержки в миллисекундах. Цикл находится в блоке операторов try/catch. Метод **sleep()** из класса **Thread** может сгенерировать исключение типа **InterruptException**, если в каком-нибудь другом потоке исполнения потребуется прервать поток.

Ниже приведен результат, выводимый данной программой.

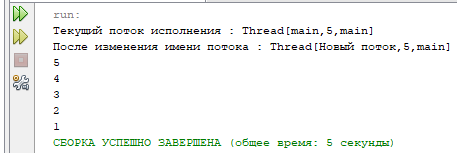


Рис. 2. Вид экрана

По умолчанию главный поток исполнения имеет имя **main** и приоритет, равный 5. Именем **main** обозначается также группа потоков исполнения, к которой относится данный поток. Группа потоков исполнения - это структура данных, которая управляет состоянием всего ряда потоков исполнения в целом. После изменения имени потока исполнения содержимое переменной **t** выводится снова - на этот раз новое имя потока исполнения.

**Создание потока исполнения**

Для создания потока исполнения следует получить экземпляр объекта типа **Thread**. В языке **Java** этой цели можно достичь следующими двумя способами:

• реализовав интерфейс **Runnable**;

• расширив класс **Thread**.

**Реализация интерфейса Runnable**

Самый простой способ создать поток исполнения состоит в том, чтобы объявить класс, реализующий интерфейс **Runnable**. Этот интерфейс предоставляет абстракцию единицы исполняемого кода. Поток исполнения можно создать из объекта любого класса, реализующего интерфейс **Runnable** . Для реализации интерфейса **Runnable** в классе должен быть объявлен единственный метод **run()**:

**public void run ()**

В теле метода **run()** определяется код, который, собственно, и составляет новый поток исполнения. Однако в методе **run()** можно вызывать другие методы, использовать другие классы, объявлять переменные таким же образом, как и в главном потоке исполнения. Единственное отличие заключается в том, что в методе **run()** устанавливается точка входа в другой, параллельный поток исполнения в программе. Этот поток исполнения завершится, когда метод **run()** возвратит управление.

После создания класса, реализующего интерфейс **Runnable**, в этом классе следует получить экземпляр объекта типа **Thread**. Для этой цели в классе **Thread** определен ряд конструкторов. Тот конструктор, который должен использоваться в данном случае, выглядит в общей форме следующим образом:

**Тhread**(**Runnable** объект\_потока, **String** имя\_потока).

В этом конструкторе параметр **объект\_ потока** обозначает экземпляр класса, реализующего интерфейс **Runnable**. Этим определяется место, где начинается выполнение потока. Имя нового потока исполнения передается данному конструктору в качестве параметра **имя\_ потока**.

После того как новый поток исполнения будет создан, он не запускается до тех пор, пока не будет вызван метод **start()**, объявленный в классе **Thread.** По существу, в методе **start()** вызывается метод **run()**.

Рассмотрим пример программы, где создается и запускается новый поток на выполнение.

**package threaddemo;**

**class NewThread implements Runnable {**

**Thread t;**

**NewThread() {**

**t = new Thread(this, "Демонстрационный поток");**

**System.out.println("Дoчepний поток создан: " + t);**

**t.start();** // запустить поток исполнения

**}**

// Точка входа во второй поток исполнения

**public void run() {**

**try {**

**for (int i = 5; i> 0; i--) {**

**System.out.println("Дoчepний поток : " + i);**

**Thread.sleep(500);**

**}**

**}**

**catch (InterruptedException е) {**

**System.out.println("Дoчepний поток прерван.");**

**}**

**System.out.println("Дoчepний поток завершен.");**

**}**

**}**

**classThreadDemo {**

**public static void main(String args[]) {**

**new NewThread();** // создание нового потока

**try {**

**for (inti = 5; i> 0; i--) {**

**System.out.println("Главный поток : " + i);**

**Thread.sleep(1000);**

**}**

**} catch (InterruptedException е) {**

**System.out.println("Главный поток прерван .");**

**}**

**System.out.println("Главный поток завершен.");**

**}**

**}**

Главный поток исполнения создается автоматически, начинает выполняться функция **main( )**. Для создания потока исполнения следует получить экземпляр объекта типа **Thread**. Объект класса **Thread** создается в следующем операторе из конструктора **NewThread( ):**

**t = new Thread (this, "Демонстрационный поток");**

Передача ссылки **this** на текущий объект в первом аргументе данного конструктора означает следующее: в новом потоке исполнения для текущего объекта по ссылке **this** следует вызвать метод **run( ).**

Далее, в приведенном выше примере программы вызывается метод **start( ),** в результате чего поток исполнения запускается, начиная с метода **run( ).** Это, в свою очередь, приводит к началу цикла **for** в дочернем потоке исполнения. После вызова метода **start( )** конструктор **NewThread( )** возвращает управление методу **main( ).** Возобновляя свое исполнение, главный поток входит в свой цикл **for**. Далее потоки выполняются параллельно, совместно используя ресурсы процессора в одноядерной системе, вплоть до завершения своих циклов.

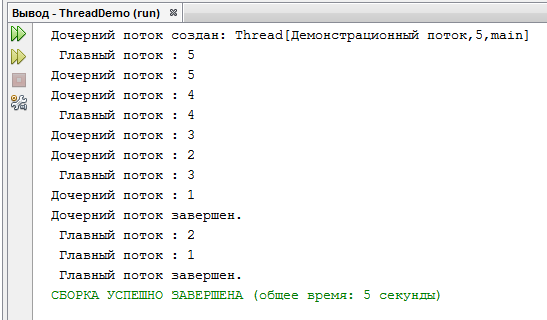


Рис. 3. Вид экрана

Как упоминалось ранее, в многопоточной программе главный поток исполнения зачастую должен завершаться последним. Если главный поток исполнения завершается раньше дочерних потоков, то исполняющая система **Java** может "зависнуть. В приведенном выше примере программы гарантируется, что главный поток исполнения завершится последним, поскольку он находится в состоянии ожидания в течение 1 000 миллисекунд в промежутках между последовательными шагами цикла, а дочерний поток исполнения - только 500 миллисекунд. Это заставляет дочерний поток исполнения завершиться раньше главного потока.

**Расширение класса Тhread**

Еще один способ создать поток исполнения состоит в том, чтобы сначала объявить класс, расширяющий класс **Thread**, а затем получить экземпляр этого класса. В расширяющем классе должен быть непременно переопределен метод **run( )**, который является точкой входа в новый поток исполнения. Кроме того, в этом классе должен быть вызван метод **start( )** для запуска нового потока на исполнение. Ниже приведена версия программы из предыдущего примера, переделенная с учетом расширения класса **Thread.**

**package extendthread;**

// Создание второго потока исполнения,

// за счет расширения класса Тhread

**class NewТhread extends Thread {**

**NewТhread() {** //создание нового потока исполнения

**super("Демонстрационный поток");**

**System.out.println( "Дoчepний поток : " + this);**

**start();** // запустить поток на исполнение

**}**

// точка входа во второй поток исполнения

**public void run(){**

**try {**

**for(inti= 5; i> 0; i--) {**

**System.out.println ("Дoчepний поток: " + i);**

**Thread.sleep(500);**

**}**

**}**

**catch(InterruptedException е ){**

**System.out.println("Дoчepний поток прерван.");**

**}**

**System.out.println("Дoчepний поток завершен.");**

**}**

**}**

**classExtendThread {**

**public static void main(String args[]) {**

**newNewТhread();** // создание нового потока исполнения

**try {**

**for(int i = 5; i> 0; i--) {**

**System.out.println(" Глaвный поток: " + i);**

**Thread.sleep(1000);**

**}**

**}**

**catch (InterruptedException е) {**

**System.out.println("Глaвный поток прерван.");**

**}**

**System.out.println("Глaвный поток завершен.");**

**}**

**}**

Эта версии программы выводит такой же результат, как и предыдущая. Здесь дочерний поток исполнения создается при конструировании объекта класса **NewThread**, наследующего класс **Thread**. Метод **super( )** в классе **NewThread** вызывает конструктор **Thread( ).**

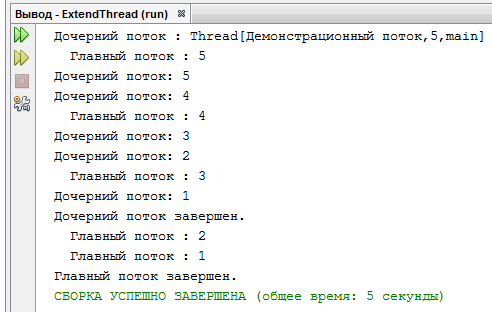


Рис. 4. Вид экрана

**Применение методов isAlive() и join()**

Как упоминалось ранее, нередко требуется, чтобы главный поток исполнения завершался последним. С этой целью метод **sleep( )** вызывался в предыдущих примерах из метода **main( )** с достаточной задержкой, чтобы все дочерние потоки исполнения завершились раньше главного. Но это неудовлетворительное решение, вызывающее следующий серьезный вопрос: откуда одному потоку исполнения известно, что другой поток завершился?

Определить, был ли поток исполнения завершен, можно двумя способами. Во­первых, для этого потока можно вызвать метод **isAlive( )**, определенный в классе **Thread**. Метод **isAlive( )** возвращает логическое значение **true**, если поток, для которого он вызван, еще исполняется. В противном случае он возвращает логическое значение **false**. И во-вторых, в классе **Thread** имеется метод **join( )**, который применяется чаще, чем метод

**isAlive( ),** чтобы дождаться завершения потока исполнения. Этот метод ожидает завершения того потока исполнения, для которого он вызван. Дополнительные формы метода **jоin( )** позволяют указывать максимальный промежуток времени, в течение которого требуется ожидать завершения указанного потока исполнения.

Ниже приведена версия программы, в которой с помощью метода **join( )** гарантируется, что главный поток завершится последним. В ней также демонстрируется применение метода **isAlive( ).**

**package demojoin;**

**class NewThread implements Runnable{**

**Stringnаmе;** // имя потока исполнения

**Thread t ;**

**NewThread(String threadname){**

**nаmе = threadname;**

**t = new Thread(this, nаmе);**

**System.out.println("Hoвый поток: " + t);**

**t.start();** // запустить поток исполнения

**}**

**public void run() {** // точка входа в поток исполнения

**try {**

**for (inti=5; i>0; i--) {**

**System.out.println( nаmе + " : " + i );**

**Thread.sleep(1000);**

**}**

**}**

**catch(InterruptedException е) {**

**System.out.println (nаmе + " прерван.");**

**}**

**System.out.println (nаmе + " завершен.");**

**}**

**}**

**classDemoJoin{**

**public static void main(String args[]) {**

**NewThread ob1= new NewThread ("Один");**

**NewThread ob2= new NewThread ("Двa");**

**NewThread ob3= new NewThread ("Tpи");**

**System.out.println ("Пoтoк Один запущен:"+ ob1.t.isAlive());**

**System.out.println ("Пoтoк Два запущен:"+ ob2.t.isAlive());**

**System.out.println ("Пoтoк Три запущен:"+ ob3.t.isAlive());**

// ожидание завершения потоков исполнения

**try {System.out.println("Ожидание завершения потоков.");**

**ob1.t.join();**

**ob2.t.join();**

**ob3.t.join();}**

**catch(InterruptedException е)**

**{System.out.println ("Глaвный поток прерван");}**

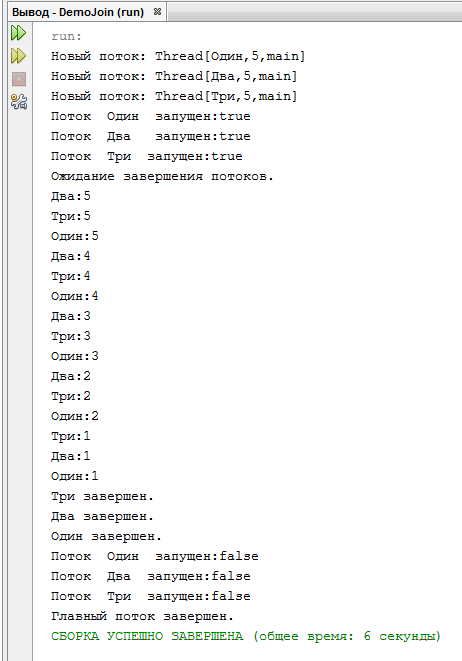
**System.out.println ("Пoтoк Один запущен:"+ ob1.t.isAlive()) ;**

**System.out.println ("Пoтoк Два запущен:"+ ob2.t.isAlive()) ;**

**System.out.println ("Пoтoк Три запущен:"+ ob3.t.isAlive()) ;**

**System.out.println ("Глaвный поток завершен.");**

**}}**



Как видно, потоки прекращают исполнение после того, как вызовы метода **join( )** возвращают управление.

**Практикум**

Переработать проект, созданный в соответствии с *индивидуальным заданием* на практическом занятии №9 (см. приложение) таким образом, чтобы *каждый из двух* *вычисляемых показателей* определялся в отдельном потоке исполнения.

Отчет оформляется по общеустановленным правилам в *электронном виде* со следующим содержанием:

1. титульный лист;
2. тема и цель лабораторной работы;
3. задание на лабораторную работу;
4. текст программы с комментариями;
5. результаты работы программы и
6. выводы по созданному проекту и варианту реализации нескольких потоков исполнения.

ПРИЛОЖЕНИЕ

**Варианты заданий**

|  |  |  |
| --- | --- | --- |
| **№**  **п/п** | **Обобщенный класс** | **Вычисляемые показатели** |
|  | Компьютер | Экземпляр с наибольшей оперативной памятью  Количество компьютеров заданной фирмы |
|  | Товар | Сумма покупки  Количество товаров одного типа |
|  | Предприятие малого бизнеса | Название предприятия с максимальным числом сотрудников  Число парикмахерских |
|  | Представитель университета | Количество студентов, обучающихся у конкретного преподавателя  Общее количество студентов |
|  | Транспортное средство | Максимальная емкость бензобака  Количество транспортных средств заданного типа |
|  | Телефон | Самая новая модель  Число телефонов одной фирмы |
|  | Программный продукт | Последняя версия  Количество программных продуктов заданной фирмы |
|  | Документ | Количество документов на заданную фамилию  Количество документов, номера которых заканчиваются нулём |
|  | Периферийное устройство компьютера | Минимальная цена устройства  Количество устройств определённой фирмы |
|  | Строительный товар | Сумма покупки  Количество товаров заданного типа |
|  | Страховой полис | Количество полисов на заданную фамилию  Полис с наименьшим номером |
|  | Насекомое | Максимальный размер крыльев  Количество экземпляров одного цвета |
|  | Недвижимость | Максимальная жилая площадь квартиры  Количество квартир заданной компании |
|  | Часы | Самый дорогой экземпляр  Количество часов определённого производителя |
|  | Товар | Количество товаров заданной фирмы  Общая сумма покупки |
|  | Магнитная карта для проезда на транспорте | Количество карт без поездок  Количество карт заданного типа |
|  | Книга | Количество книг одного автора  Название книги с наибольшим числом страниц |
|  | Канцелярские товары | Количество товаров определённой фирмы  Общая сумма покупки |
|  | Периферийное устройство компьютера | Минимальная цена устройства  Количество устройств определённой фирмы |
|  | Документ | Документ, выданный раньше всех других  Количество документов определённого типа |
|  | Путешествие (тур) | Самый дешевый тур на 7 и более дней  Число туров в выбранную страну |
|  | Товар | Самый дорогой товар  Количество товаров определённого бренда |
|  | Страховой полис | Полис с максимальным сроком действия  Количество полисов на заданную фамилию |
|  | Грузовое транспортное средство | Наибольший пробег на полном бензобаке  Количество грузовиков заданной фирмы |
|  | Предприятие малого бизнеса | Название предприятия с максимальным числом сотрудников  Число парикмахерских |
|  | Представление | Минимальное число зрителей в зале  Представление с самым коротким названием |
|  | Запоминающее устройство | Экземпляр с минимальным размером  Число устройств одной фирмы |
|  | Принтер | Представитель с наибольшей производительностью  Число принтеров одного производителя |
|  | Книга | Количество книг одного автора  Название книги с наибольшим числом страниц |
|  | Грузовое транспортное средство | Наибольший пробег на полном бензобаке  Количество грузовиков заданной фирмы |