ЛАБОРАТОРНАЯ РАБОТА №3. ПОТОКИ ВЫПОЛНЕНИЯ

Цель: изучить основные способы реализации многопоточности в Java.

3.1 Теоретическая часть

Реализация многопоточности в Java

Для реализации многопоточности мы должны воспользоваться классом java.lang.Thread. В этом классе определены все методы, необходимые для создания потоков, управления их состоянием и синхронизации.

Как пользоваться классом Thread? Есть две возможности.

Во-первых, вы можете создать свой дочерний класс на базе класса Thread. При этом вы должны переопределить метод run. Ваша реализация этого метода будет работать в рамках отдельного потока.

Во-вторых, ваш класс может реализовать интерфейс Runnable. При этом в рамках вашего класса необходимо определить метод run, который будет работать как отдельный поток.

Второй способ особенно удобен в тех случаях, когда ваш класс должен быть унаследован от какого-либо другого класса и при этом вам нужна многопоточность. Так как в языке программирования Java нет множественного наследования, невозможно создать класс, для которого в качестве родительского будут выступать классы Applet и Thread. В этом случае реализация интерфейса Runnable является единственным способом решения задачи.

Методы класса Thread

В классе Thread определены три поля, несколько конструкторов и большое количество методов, предназначенных для работы с потоками. Ниже приведено краткое описание полей, конструкторов и методов.

С помощью конструкторов вы можете создавать потоки различными способами, указывая при необходимости для них имя и группу. Имя предназначено для идентификации потока и является необязательным атрибутом. Что же касается групп, то они предназначены для организации защиты потоков друг от друга в рамках одного приложения.

Методы класса Thread предоставляют все необходимые возможности для управления потоками, в том числе для их синхронизации.

Реализация интерфейса Runnable

Описанный выше способ создания потоков как объектов класса Thread или унаследованных от него классов кажется достаточно естественным. Однако этот

способ не единственный. Если вам нужно создать только один поток, работающий одновременно с кодом Applet, то проще выбрать второй способ с использованием интерфейса Runnable.

Идея заключается в том, что основной класс апплета, который является дочерним по отношению к классу Applet, дополнительно реализует интерфейс Runnable, как это показано ниже:

```
public class MultiTask extends
  Applet implements Runnable
{
  Thread m_MultiTask = null;
  . . .
  public void run()
  {
    . . .
  }
  public void start()
  {
    if (m_MultiTask == null)
    {
      m_MultiTask = new Thread(this);
      m_MultiTask.start();
    }
  }
  public void stop()
  {
    if (m_MultiTask != null)
    {
      m_MultiTask.stop();
      m_MultiTask = null;
    }
  }
}
```

Внутри класса необходимо определить метод run, который будет выполняться в рамках отдельного потока. При этом можно считать, что код апплета и код метода run работают одновременно как разные потоки.

Для создания потока используется оператор new. Поток создается как объект класса Thread, причем конструктору передается ссылка на класс апплета:

```
m_MultiTask = new Thread(this);
```

При этом, когда поток запустится, управление получит метод run, определенный в классе Applet. Как запустить поток? Запуск выполняется, как и раньше, методом start. Обычно поток запускается из метода start Applet, когда пользователь отображает страницу сервера Web, содержащую Applet. Остановка потока выполняется методом stop.

Завершение и остановка потоков

Как уже указывалось, нормальное завершение нити происходит при выходе из метода run. Кроме того, иногда в приложении требуется выполнить временную остановку нити, чтобы потом возобновить ее работу.

В классе Thread есть методы stop (завершить), suspend (приостановить) и resume (возобновить), но все они объявлены deprecated (устаревшими) и их использование не рекомендовано. О причинах можно почитать подробнее в документации по Java и их здесь рассматривать не будем.

Как же поступить в случае, если нужно приостановить выполнение процесса? Для этого нужно не останавливать процесс, а обеспечить такой алгоритм, при котором он работает вхолостую.

Приоритеты потоков

Если процессор создал несколько потоков, то все они выполняются параллельно, причем время центрального процессора (или нескольких центральных процессоров в мультипроцессорных системах) распределяется между этими потоками.

Распределением времени центрального процессора занимается специальный модуль операционной системы – планировщик. Планировщик по очереди передает управление отдельным потокам, так что даже в однопроцессорной системе создается полная иллюзия параллельной работы запущенных потоков.

Распределение времени выполняется по прерываниям системного таймера. Поэтому каждому потоку дается определенный интервал времени, в течение которого он находится в активном состоянии.

Заметим, что распределение времени выполняется для потоков, а не для процессов. Потоки, созданные разными процессами, конкурируют между собой за получение процессорного времени.

Для оптимизации параллельной работы нитей в Java имеется возможность устанавливать приоритеты нитей. Нити с большим приоритетом имеют преимущество в получении времени процессора перед нитями с более низким приоритетом.

Работа с приоритетами обеспечивается методами класса Thread.

Метод, который устанавливает приоритет потока, – public final void setPriority(int newPriority).

Метод, который позволяет установить приоритет потока, – public final int getPriority().

Значение параметра в методе setPriority не может быть произвольным. Оно должно находиться в пределах от MIN_PRIORITY до MAX_PRIORITY. При своем создании нить имеет приоритет NORM_PRIORITY.

Средства синхронизации потоков в Java

Проанализируем эту проблему с другой точки зрения. У нас есть некоторый ресурс, в данном случае – случайное число (переменная randValue), доступ к которому нужно упорядочить. То есть нужно заблокировать доступ к этому ресурсу для всех нитей, кроме одной, пока эта нить не выполнит над ним необходимые действия.

В Java есть возможности по синхронизации нитей, построенные на этом принципе. То есть определяется некоторый ресурс, который может быть заблокирован. Таким ресурсом может быть любой объект (но не данные элементарного типа). Далее определяются критические участки программы. При входе в такой участок ресурс блокируется и становится недоступным для всех других нитей (с некоторой оговоркой, которую мы рассмотрим позже). При выходе из критического участка выполняется разблокировка ресурса.

В качестве критического участка программы в Java может быть определен некоторый нестатический метод или блок. При вызове метода блокируется объект, для которого вызван данный метод (this). Для блока блокируемый объект указывается явно.

Синтаксис определения критических участков следующий.

Для метода мы просто при описании метода указываем описатель synchronized, например:

```
public void synchronized f() {
  ...
}
```

Для блока мы непосредственно перед блоком ставим конструкцию synchronized (объект), где объект – это ссылка на блокируемый объект. Например:

```
synchronized(ref) {
  ...
}
```

Здесь в качестве критического участка определяется блок, а в качестве блокируемого объекта – объект, на который ссылается ref.

3.2 Содержание отета

1 Титульный лист.
2 Цель работы.
3 Краткие теоретические сведения.
4 Результаты выполнения практического задания.
5 Выводы.

3.3 Задания к лабораторной работе №3

1 Создать Applet, используя поток: строка движется горизонтально, отражаясь от границ Applet и меняя при этом случайным образом свой цвет.

2 Создать Applet, используя поток: строка движется по диагонали. При достижении границ Applet все символы строки случайным образом меняют регистр.

3 Организовать сортировку массива методами Шелла, Хоара, пузырька, на основе бинарного дерева в разных потоках.

4 Реализовать сортировку графических объектов, используя алгоритмы из задания 3.4.

5 Создать Applet с точкой, движущейся по окружности с постоянной угловой скоростью. Сворачивание браузера должно приводить к изменению угловой скорости движения точки для следующего запуска потока.

6 Изобразить точку, пересекающую с постоянной скоростью окно слева направо (справа налево) параллельно горизонтальной оси. Как только точка доходит до границы окна, в этот момент от левого (правого) края с вертикальной координатой у, выбранной с помощью датчика случайных чисел, начинает свое движение другая точка и т. д. Цвет точки также можно выбирать с помощью датчика случайных чисел. Для каждой точки создается собственный поток.

7 Изобразить в приложении правильные треугольники, вращающиеся в плоскости экрана вокруг своего центра. Каждому объекту соответствует поток с заданным приоритетом.

8 Условия предыдущих задач изменяются таким образом, что центр вращения перемещается от одного края окна до другого с постоянной скоростью параллельно горизонтальной оси.

9 Создать фрейм с тремя шариками, одновременно летающими в окне. С каждым шариком связан свой поток со своим приоритетом.

10 Два изображения выводятся в окно. Затем они постепенно исчезают с различной скоростью в различных потоках (случайным образом выбираются точки изображения, и их цвет устанавливается в цвет фона).

11 Условие предыдущей задачи изменить на применение эффекта постепенного проявления двух изображений.

12 Создать Applet «Бегущая строка».

13 Реализовать приложение, в котором пользователь имеет возможность указывать маски файлов для поиска и набор путей, по которым эти файлы нужно искать (например, список логических дисков).

14 Написать секундомер – класс Stopwatch – для замера времени в отдельном потоке выполнения. В классе должны быть реализованы следующие методы:

- start – начинает отсчет времени;
- stop – прерывает отсчет времени;
- reset – сбрасывает текущее значение секундомера;
- getTime – возвращает отсчитанное время в миллисекундах.

Для демонстрации работы секундомера написать консольное приложение. Пользователю должны быть доступными следующие команды:

- start N – запустить секундомер и дать ему идентификатор N;
- stop N – остановить секундомер с идентификатором N;
- reset N – сбросить время у секундомера с идентификатором N;
- time N – показать время у секундомера с идентификатором N;
- help – список команд;
- exit – выход.

15 Подсчет простых чисел. Написать Swing-приложение для подсчета простых чисел в параллельных потоках. Каждый поток подсчитывает простые числа из заданного диапазона. Всего должно быть запущено три потока подсчета. Полученные числа выдаются на экран. Можно запускать/останавливать подсчет несколько раз. Пользователю доступны следующие элементы управления:

- кнопка запуска всех потоков;
- кнопка остановки всех потоков;
- поле (JTextArea) для вывода результата в виде

<номер потока>: <число>;

- четыре поля ввода диапазонов чисел для подсчета. Получается три диапазона (между четырьмя числами) для каждого потока;
- три строки с информацией о состоянии каждого потока.

Контрольные вопросы

1 Дайте определение понятию «процесс».

2 Дайте определение понятию «поток».

3 Дайте определение понятию «синхронизация потоков».

4 Как взаимодействуют программы, процессы и потоки?

5 В каких случаях целесообразно создавать несколько потоков?

6 Что может произойти, если два потока будут выполнять один и тот же код в программе?

7 Что вы знаете о главном потоке программы?

8 Какие есть способы создания и запуска потоков?

9 Какой метод запускает поток на выполнение?
10 Какой метод описывает действие потока во время выполнения?
11 Когда поток завершает свое выполнение?