JAVA. Лабораторная работа №7
Тема: Многопоточные приложения

Поток - последовательность выполняемых операций внутри программы. Потоки позволяют исполнение нескольких действий одновременно. Поддержка многопоточного программирования встроена в язык Java.

Поток может находиться в одном из следующих состояний:

- New – создан, но не запущен.
- Running – активно использует ресурс CPU.
- Blocked – ожидает ресурсов или событий.
- Resumed – готов к работе после остановки типа block или suspend.
- Поток помещается в очередь, ожидающую доступа к ресурсам ЦП.
- Suspended – поток перестал использовать ЦП.
- Самостоятельно или был принудительно снят с исполнения.

1.1 Класс java.lang.Thread

Данный класс предоставляет абстракцию, позволяющую выполнять инструкции языка Java в отдельном потоке. Имеет следующие конструкторы.

Thread()	Создает новый класс.
Thread(String name)	Создает новый именованный класс-поток.
Thread(Runnable target)	Создает новый класс-поток на основе экземпляра Runnable, чей метод run() будет исполняться в отдельном потоке.
Thread(Runnable target, String name)	Создает новый именованный класс-поток на основе Runnable.

Среди методов класса можно выделить:

public static Thread currentThread()	Возвращает ссылку на активный в настоящее время поток.
public void interrupt()	Прерывает исполнение потока.
public final boolean isAlive()	Определяет, работает ли поток.
public void run()	Метод запускается в отдельном потоке.
public void start()	Запускает выполнение метода run() в отдельном потоке.
public void join(Thread other)	Приостанавливает исполнение текущего потока до тех пор пока поток other не завершится.
public void stop()	Завершает выполнение потока.
public void resume()	Восстанавливает выполнение потока.
public void sleep(long delay)	Приостанавливает выполнение потока на delay миллисекунд.
public void yield()	Метод пробует отказаться от выполнения на ЦП в пользу других потоков.

Класс Thread имеет несколько публичных полей, отражающих относительные приоритеты исполнения: **MAX_PRIORITY, MIN_PRIORITY и NORM_PRIORITY.**

Создать новый поток можно одним из следующих способов:

- Наследование от класса Thread

Наследник класса Thread переопределяет метод run(). Вызов метода start() у экземпляра подкласса запускает выполнение потока. Операторы, помещенные в методе run() выполняются виртуальной машиной в новом потоке.

- Реализация интерфейса **java.lang.Runnable**

Интерфейс Runnable должен быть реализован классом, который должен выполняться в отдельном потоке. Для этого класс должен реализовывать всего один метод - run().. Экземпляр класса должен быть передан конструктору класса Thread в качестве аргумента. Запуск потока осуществляется методом start() класса Thread.

1.2 Пример многопоточного приложения на основе класса Thread

Следующий пример выводит на консоль сообщения от двух потоков исполнения. Можно отметить, что потоки выполняются одновременно.

```
public class ConcurrentDemo{
   public static void main(String []args){
      new ConcurrentThread1().start();
      new ConcurrentThread2().start();
   }
}
class ConcurrentThread1 extends Thread{
   public void run(){
      for (int i=0;i<=10000;i++) System.out.println("#1: "+i);
   }
}
class ConcurrentThread2 extends Thread{
   public void run(){
      for (int i=0;i<=10000;i++) System.out.println("#2: "+i);
   }
}
```

1.3 Пример создания потока методом реализации интерфейса Runnable

```
class SimpleCombat implements Runnable{
   public void run(){
     //конкретные действия поединка
     takePosition();
     fire();
   }
}
SimpleCombat combat = new SimpleCombat();
new Thread(combat).start();
```

1.4 Объект синхронизации

Метод или блок кода, выполнение которого не должно прерываться исполнением того же самого кода, но на другом потоке, можно помечать модификатором synchronized.

Для задания синхронизации между методами нужен некий объект. Обычно таким объектом является разделяемый ресурс. Синхронизованный метод становится «владельцем» монитора объекта. Объектом синхронизации может быть ссылочный тип (объект) или класс.

1.4.1 Синхронизация доступа к состоянию объекта

Методы, обращающиеся к состоянию объекта, синхронизованы для того чтобы исключить одновременное изменение состояния.

```
public class SeniorStable {
    private int horseCount = 10;
    public void synchonized returnHorse(){
        horseCount++;
     }
     public void synchonized takeHorse(){
        horseCount--;
     }
    public int synchonized getHorseCount(){
            return horseCount;
     }
 }
```

1.5 Межпотоковое взаимодействие

Следующие методы класса Object используются для реализации взаимодействия потоков между собой.

public final void wait() throws InterruptedException	Заставляет поток ожидать когда какой-то другой поток вызовет метод notify() или notifyAll() для данного объекта.
public final void notify()	Пробуждает поток, который вызвал метод wait() для этого же объекта.
public final void notifyAll()	Пробуждает все потоки, который вызвали метод wait() для этого же объекта.

1.6 Группы потоков

Потоки могут объединяться в группы с целью управления большим количеством потоков одновременно. Потоку разрешено получать информацию о своей группе.

Запрещено о других группах и о более крупных группах (являющихся надгруппами).

Если поток или группа потоков не обрабатывает исключение, то оно обрабатывается JVM.

Для установки собственного обработчика можно использовать метод uncaughtException(Thread thread, Throwable throwable)класса java.lang.ThreadGroup.

Следующий пример демонстрирует создание группы потоков.

```
public class thGroup{
   public static void main(Strings []args){
      ThreadGroup mainGr=new ThreadGroup("main group");
      Thread thread1=new Thread(mainGr, "thread1");
      ThreadGroup minorGr=new ThreadGroup(mainGr, "minor group");
   }
}
```

1.7 Исполнение по расписанию

В следующем примере сообщение выдается через несколько секунд после инициализации приложения.

```
public class TimerReminder {

   Timer timer;

   public TimerReminder(int seconds) {
      timer = new Timer();
      timer.schedule(new RemindTask(), seconds*1000);
   }

   class RemindTask extends TimerTask {
      public void run() {
         System.out.format("Time's up!%n");
         timer.cancel(); //Terminate the timer thread
      }
   }

   public static void main(String args[]) {
      System.out.format("About to schedule task.%n");
      new TimerReminder(5);
      System.out.format("Task scheduled.%n");
   }
}
```

2. Задания

2.1. Реализовать многопоточное приложение, производящее тестирование целых чисел типа long на простоту в заданном диапазоне (вводится с клавиатуры). Главный поток приложения формирует пропорциональные последовательности чисел (от 5) и создает для каждого отдельный поток обработки.

Самый простой путь **определения простых чисел** – проверить, имеет ли данное число n (n >= 2) делители в интервале [2; n-1]. Если делители есть, число n – составное, если – нет, то – простое. При реализации алгоритма разумно делать проверку на четность введенного числа, поскольку все четные числа делятся на 2 и являются составными числами, то, очевидно, что нет необходимости искать делители для этих чисел.

2.2. Реализовать многопоточное приложение, реализующее поиск подстроки в файлах. Список файлов передается в качестве параметра командной строки. Для каждого файла выделяется отдельный поток. Для вывода результатов поиска в консоль создается отдельный поток, считывающий данные по мере поступления из разделяемого списка объектов класса SearchResult, имеющего следующего поля «имя файла», «индекс вхождения».

2.3. Реализовать многопоточное приложение, реализующее вывод всех четных слов из списка файлов. Для каждого файла создается новый поток.