Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**Отчёт**

**по лабораторной работе №5**

**Дисциплина: разработка мобильных приложений**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. А. Курбатский

Направление подготовки 02.03.02 Фундаментальная информатика и

информационные технологии

Направленность (профиль) Математическое и программное обеспечение

компьютерных технологий

Преподаватель

д-р техн. наук, проф. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. И. Шиян

**Цель работы:** изучить приемы и методы организации и управления потоками.

**Общие сведения о потоках**

Практически все современные ОС позволяют исполнять несколько таких наборов одновременно. Для чего это нужно? Современное программное обеспечение по большей часто асинхронно. Приложение ждет реакции пользователя, ждет прихода данных по сети, ждет готовности устройства. Приложение ждет. Если вы посмотрите на загрузку процессора, то увидите, что чаще всего 99% времени он простаивает. Соответственно, пока одна задача находится в стадии ожидания, можно заниматься другой. Например, во время ожидания прихода данных по сети можно отрисовать то, что уже пришло. Это если говорить о браузере. Да и вообще, в ОС одновременно происходит множество разных событий. Если бы всеми ими занимался один единственный поток – все работало бы значительно медленнее. Если бы вообще работало.

Следует разграничить два понятия: поток и процесс. Процесс – это задача операционной системы. У него собственное адресное пространство, с ним может быть проассоциировано несколько потоков. Поток же – это гораздо более мелкая единица. Все потоки разделяют адресное пространство породившего их процесса и имеют доступ к одним данным. Поскольку несколько потоков имеют доступ к одним данным, они их могут менять. Иногда это может привести к очень большим проблемам. Для решения этих проблем нужна синхронизация.

**Базовые классы для работы с потоками**

Многозадачность позволяют реализовывать следующие классы и интерфейсы:

* интерфейс Runnable;
* класс Thread;
* класс java.util.Timer;
* класс javax.swing.Timer.

**Простые методы управления потоками**

Коснемся нескольких возможностей класса Thread по передаче управления между потоками – это три простых метода:

* sleep();
* yield();
* join().

**Жизненный цикл потоков**

Поток может находиться в одном из состояний, соответствующих элементам статически вложенного перечисления Thread.State (рис.29):

* NEW – поток создан, но еще не запущен;
* RUNNABLE – поток выполняется;
* BLOCKED – поток блокирован;
* WAITING – поток ждет окончания работы другого потока;
* TIMEDWAITING – поток некоторое время ждет окончания другого потока;
* TERMINATED – поток завершен.

**Методы синхронизации потоков**

Во избежание одновременной записи методы могут объявляться как synchronized. Синхронизированный метод изолирует объект, после чего он становится недоступным для других потоков. Изоляция снимается, когда поток полностью выполнит соответствующий метод. Другой способ снятия изоляции — вызов метода wait() из изолированного метода.

**Ход работы:**

В процессе выполнения лабораторной работы было создано приложение, демонстрирующее гонку между несколькими потоками.

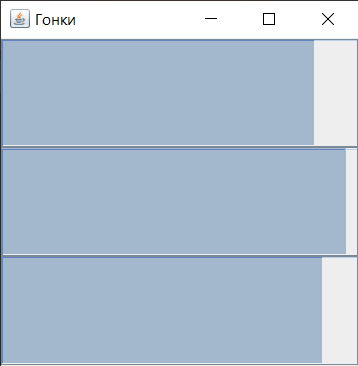


Рисунок 1 - Результат работы программы.

**Вывод:** были изучены приемы и методы организации и управления потоками.

**Листинг программы**

**Файл Raicing.java**

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.\*;  
import java.util.Random;  
  
public class RacingGame extends JFrame {  
 private JProgressBar progressBar1;  
 private JProgressBar progressBar2;  
 private JProgressBar progressBar3;  
  
 public RacingGame() {  
 setTitle("Гонки");  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 setLayout(new GridLayout(3, 1));  
 Random rand = new Random();  
 progressBar1 = new JProgressBar();  
 progressBar2 = new JProgressBar();  
 progressBar3 = new JProgressBar();  
  
 add(progressBar1);  
 add(progressBar2);  
 add(progressBar3);  
  
 setSize(300, 300);  
 setVisible(true);  
  
 startRace(rand);  
 }  
  
 private void startRace(Random rand) {  
 Thread thread1 = new Thread(new Runnable() {  
 public void run() {  
 for (int i = 0; i <= 100; i++) {  
 progressBar1.setValue(i);  
 try {  
 Thread.*sleep*(rand.nextInt(40, 100));  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 });  
  
 Thread thread2 = new Thread(new Runnable() {  
 public void run() {  
 for (int i = 0; i <= 100; i++) {  
 progressBar2.setValue(i);  
 try {  
 Thread.*sleep*(rand.nextInt(40, 100));  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 });  
  
 Thread thread3 = new Thread(new Runnable() {  
 public void run() {  
 for (int i = 0; i <= 100; i++) {  
 progressBar3.setValue(i);  
 try {  
 Thread.*sleep*(rand.nextInt(40, 100));  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 });  
  
 thread1.start();  
 thread2.start();  
 thread3.start();  
 }  
  
 public static void main(String[] args) {  
 SwingUtilities.*invokeLater*(new Runnable() {  
 public void run() {  
 new RacingGame();  
 }  
 });  
 }  
}