Лабораторная работа No 10

Тема работы: «Поточная модель» 1. Цель работы

Закрепляет навыки создания и управления потоками.

2. Задание

Номер варианта соответствует вашему номеру по списку.

Необходимо в задании 1 лабораторной работы No 9 реализовать многопоточную работу приложения (потоки не синхронизированные).

3. Оснащение работы  
Задание по варианту, ЭВМ, среда разработки IntelliJ IDEA.

4. Основные теоретические сведения

Потоки. Класс Thread и интерфейс Runnable  
В русской терминологии за термином Thread укрепился перевод "Поток". Хотя это слово также можно перевести как "Нить". Иногда в зарубежных учебных материалах понятие потока объясняется именно на нитях. Продолжим

логический ряд - там где нити, там и клубок. А где клубок, там и кот. Сразу видно, что у переводчиков не было котов. Так и возникла путаница. Тем более что существуют другие потоки под термином Stream. Переводчики, вообще странный народ.

Когда запускается любое приложение, то начинает выполняться поток, называемый главным потоком (main). От него порождаются дочерние потоки. Главный поток, как правило, является последним потоком, завершающим выполнение программы.

Несмотря на то, что главный поток создаѐтся автоматически, им можно управлять через объект классаThread. Для этого нужно вызвать метод currentThread(), после чего можно управлять потоком.

Класс Thread содержит несколько методов для управления потоками. ✓ getName() - получить имя потока  
✓ getPriority() - получить приоритет потока  
✓ isAlive() - определить, выполняется ли поток

✓ join() - ожидать завершение потока  
✓ run() - запуск потока. В нѐм пишите свой код  
✓ sleep() - приостановить поток на заданное время

✓ start() - запустить поток  
Получим информацию о главном потоке и поменяем его имя.

Thread mainThread = Thread.currentThread();

mInfoTextView.setText("Текущий поток: " mainThread.getName());

// Меняем имя и выводим в текстовом поле mainThread.setName("CatThread"); mInfoTextView.append("\nНовое имя потока:

mainThread.getName());

+

" +

Имя у главного потока по умолчаниюmain, которое мы заменили на CatThread.

Вызовем информацию о названии потока без указания метода.

Thread mainThread = Thread.currentThread();

mInfoTextView.setText("Текущий поток: " + mainThread);

В этом случае можно увидеть строчку Thread[main,5,main] - имя потока, его приоритет и имя его группы.

Создание собственного потока

Создать собственный поток не сложно. Достаточно наследоваться от класса Thread.

Объявим внутри нашего класса внутренний класс и вызовем его по щелчку, вызвав метод start().

public class MyThread extends Thread {

public void run() {

Log.d(TAG, "Mой поток запущен..."); }

}

public void onClick(View view) { MyThread myThread = new MyThread(); myThread.start();

}

Как вариант, перенести вызов метода start() в конструктор.

public void onClick(View view) { MyThread myThread = new MyThread();

}

public class MyThread extends Thread {

// Конструктор

MyThread() {

// Создаѐм новый поток  
super("Второй поток");  
Log.i(TAG, "Создан второй поток " + this); start(); // Запускаем поток

}

public void run() {  
Log.d(TAG, "Mой поток запущен...");

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

Log.i(TAG, "Второй поток: " + i);

Thread.sleep(500); }

} catch (InterruptedException e) { Log.i(TAG, "Второй поток прерван");

} }

}

Создание потока с интерфейсом Runnable

Есть более сложный вариант создания потока. Для создания нового потока нужно реализовать интерфейс Runnable. Вы можете создать поток из любого объекта, реализующего интерфейс Runnable и объявить метод run().

Внутри метода run() вы размещаете код для нового потока. Этот поток завершится, когда метод вернѐт управление.

Когда вы объявите новый класс с интерфейсом Runnable, вам нужно использовать конструктор:

Thread(Runnable объект\_потока, String имя\_потока)

В первом параметре указывается экземпляр класса, реализующего интерфейс. Он определяет, где начнѐтся выполнение потока. Во втором параметре передаѐтся имя потока.

После создания нового потока, его нужно запустить с помощью метода start(), который, по сути, выполняет вызов метода run().

Создадим новый поток внутри учебного проекта в виде вложенного класса и запустим его.

package ru.alexanderklimov.expresscourse;

import android.os.Bundle;  
import android.support.v7.app.AppCompatActivity;

import

import

import

import

import

import ru.alexanderklimov.expresscourse.R.id.textViewInfo;

android.util.Log; android.view.View; android.widget.Button; android.widget.EditText; android.widget.TextView;

public class MainActivity extends AppCompatActivity { final String TAG = "ExpressCourse";

private Button mButton;

private EditText mResultEditText;

private TextView mInfoTextView;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState); setContentView(R.layout.activity\_main);

mButton = findViewById(R.id.buttonGetResult);

mResultEditText findViewById(R.id.editText);

mInfoTextView

findViewById(textViewInfo);

}

=

=

(Button)

(EditText)

(TextView)

public void onClick(View view) {

new MyRunnable(); // создаѐм новый поток

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

Log.i(TAG, "Главный поток: " + i);

Thread.sleep(1000); }

} catch (InterruptedException e) { Log.i(TAG, "Главный поток прерван");

} }

static

примера");

thread);

// Конструктор

MyRunnable() {

// Создаѐм новый второй поток  
thread = new Thread(this, "Поток для

Log.i(TAG, "Создан второй поток " +

thread.start(); // Запускаем поток }

// Обязательный метод для интерфейса Runnable

public void run() {

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

Log.i(TAG, "Второй поток: " + i);

Thread.sleep(500); }

} catch (InterruptedException e) { Log.i(TAG, "Второй поток прерван");

} }

Внутри класса Thread

} }

конструктора MyRunnable() мы

создаѐм

новый

объект

class MyRunnable implements Runnable {

Thread thread;

thread = new Thread(this, "Поток для примера");

В первом параметре использовался объект this, что означает желание вызвать метод run() этого объекта. Далее вызывается метод start(), в результате чего запускается выполнение потока, начиная с метода run(). В свою очередь метод запускает цикл для нашего потока. После вызова методаstart(), конструктор MyRunnable() возвращает управление приложению. Когда главный поток продолжает свою работу, он входит в свой цикл. После этого оба потока выполняются параллельно.

Можно запускать несколько потоков, а не только второй поток в дополнение к первому. Это может привести к проблемам, когда два потока пытаюсь работать с одной переменной одновременно.

Ключевое слово syncronized - синхронизированные методы

Для решения проблемы с потоками, которые могут внести путаницу, используется синхронизация.

Метод может иметь модификаторsyncronized. Когда поток находится внутри синхронизированного метода, все другие потоки, которые пытаются вызвать его в том же экземпляре, должны ожидать. Это позволяет исключить путаницу, когда несколько потоков пытаются вызвать метод.

syncronized void meow(String msg);

Кроме того, ключевое слово syncronized можно использовать в качестве оператора. Вы можете заключить в блок syncronizedвызовы методов какого- нибудь класса:

syncronized(объект) {

// операторы, требующие синхронизации

}

Looper  
Поток имеет в своѐм составе сущности Looper, Handler, MessageQueue.

Каждый поток имеет один уникальный Looper и может иметь много Handler.

Считайте Looper вспомогательным объектом потока, который управляет им. Он обрабатывает входящие сообщения, а также даѐт указание потоку завершиться в нужный момент.

Поток получает свой Looper и MessageQueue через метод Looper.prepare() после запуска. Looper.prepare()идентифицирует вызывающий потк, создаѐт Looper и MessageQueue и связывает поток с ними в хранилище ThreadLocal. Метод Looper.loop() следует вызывать для запуска Looper. Завершить его работу можно через метод looper.quit().

class LooperThread extends Thread {

public Handler mHandler;

public void run() { Looper.prepare();

mHandler = new Handler() {  
public void handleMessage(Message msg) {

// process incoming messages here

}

};

Looper.loop(); }

}

Используйте статический метод getMainLooper() для доступа к Looper главного потока:

Looper mainLooper = Looper.getMainLooper();

Создадим два потока. Один запустим в основном потоке, а второй отдельно от основного. Нам будет достаточно двух кнопок и метки.

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android "

xmlns:tools="http://schemas.android.com/tools" android:layout\_width="match\_parent"

android:layout\_height="match\_parent" android:orientation="vertical" tools:context=".MainActivity" >

<Button  
android:id="@+id/button\_start" android:layout\_width="match\_parent" android:layout\_height="wrap\_content" android:text="start()" />

<Button  
android:id="@+id/button\_run" android:layout\_width="match\_parent" android:layout\_height="wrap\_content" android:text="run()" />

<TextView  
android:id="@+id/textview\_info" android:layout\_width="wrap\_content" android:layout\_height="wrap\_content" />

</LinearLayout>

Обратите внимание, как запускаются потоки. Первый поток запускается с помощью метода start(), а второй - run(). Затем проверяем, в каком потоке мы находимся.

package ru.alexanderklimov.as23;

import android.os.Bundle;  
import android.os.Looper;  
import android.support.v7.app.AppCompatActivity; import android.view.View;  
import android.widget.Button;  
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

TextView mInfoTextView;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState); setContentView(R.layout.activity\_main);

Button startButton findViewById(R.id.button\_start);

Button runButton findViewById(R.id.button\_run);

mInfoTextView findViewById(R.id.textview\_info);

=

=

(Button)

(Button)

(TextView)

startButton.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View view) {

Thread thread = new Thread(new

thread.start(); //в фоновом потоке }

});

runButton.setOnClickListener(new View.OnClickListener() {

MyRunnable());

MyRunnable());

@Override

public void onClick(View view) {

Thread thread = new Thread(new

thread.run(); //в текущем потоке }

}); }

private class MyRunnable implements Runnable {

@Override

public void run() {

// Проверяем, в каком потоке находимся

=

if (Looper.getMainLooper().getThread() == Thread.currentThread()) {

mInfoTextView.setText("В основном

} else {

runOnUiThread(new Runnable() {

@Override

public void run() {

mInfoTextView.setText("В

} });

Эта тема достаточно сложная и для большинства не представляет интереса и необходимости изучать.

В Android потоки в чистом виде используются всѐ реже и реже, у системы есть собственные способы.

5. Порядок выполнения работы

6. Форма отчета о работе

Лабораторная работа No \_\_\_ Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_

Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_  
Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

7. Контрольные вопросы и задания

1. построить объектную модель;  
2. построить алгоритм решения данной задачи;  
3. реализовать приложение;  
4. провести тестирование полученного приложения.

1. Чтотакоепаток?  
2. Как создать несколько потоков?  
3. Как запустить потоки?  
4. Как синхронизировать патоки?  
5. Для чего необходимо синхронизировать патоки?