Практическая работа 11

Методические материалы

Многопоточность и асинхронность. Потоки, фрагменты и ViewModel. Класс AsyncTask.

AsyncTask и фрагменты. Работа с сетью. WebView.

Создание потоков и визуальный интерфейс

Когда мы запускаем приложение на Android, система создает поток, который называется основным потоком приложения или UI-поток. Этот поток обрабатывает все изменения и события пользовательского интерфейса. Однако для вспомогательных операций, таких как отправка или загрузка файла, продолжительные вычисления и т.д., мы можем создавать дополнительные потоки.

Для создания новых потоков нам доступен стандартный функционал класса Thread из базовой библиотеки Java из пакета java.util.concurrent, которые особой трудности не представляют. Тем не менее трудности могут возникнуть при обновлении визуального интерфейса из потока.

Например, создадим простейшее приложение с использованием потоков. Определим следующую разметку интерфейса в файле **activity_main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
```

```
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:textSize="22sp"
        app:layout_constraintBottom_toTopOf="@id/button"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Запустить поток"
        app:layout_constraintTop_toBottomOf="@id/textView"
        app:layout_constraintLeft_toLeftOf="parent"  />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Здесь определена кнопка для запуска фонового потока, а также текстовое поле для отображения некоторых данных, которые будут генерироваться в запущенном потоке.

Далее определим в классе **MainActivity** следующий код:

```
package com.example.threadapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
```

```
import android.widget.Button;
import android.widget.TextView;
import java.util.Calendar;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        TextView textView = findViewById(R.id.textView);
        Button button = findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Определяем объект Runnable
                Runnable runnable = new Runnable() {
                    @Override
                    public void run() {
                        // получаем текущее время
                        Calendar c = Calendar.getInstance();
                        int hours = c.get(Calendar.HOUR_OF_DAY);
                        int minutes = c.get(Calendar.MINUTE);
                        int seconds = c.get(Calendar.SECOND);
                        String time = hours + ":" + minutes + ":" + seconds;
                        // отображаем в текстовом поле
                        textView.setText(time);
                    }
```

```
            };
            // Определяем объект Thread - новый поток
            Thread thread = new Thread(runnable);
            // Запускаем поток
            thread.start();
        }
    });
  }
}
```

Итак, здесь к кнопке прикреплен обработчик нажатия, который запускает новый поток. Создавать и запускать поток в Java можно различными способами. В данном случае сами действия, которые выполняются в потоке, определяются в методе **run()** объекта **Runnable:**

```
Runnable runnable = new Runnable() {
  @Override
  public void run() {
    // получаем текущее время
    Calendar c = Calendar.getInstance();
    int hours = c.get(Calendar.HOUR_OF_DAY);
    int minutes = c.get(Calendar.MINUTE);
    int seconds = c.get(Calendar.SECOND);
    String time = hours + ":" + minutes + ":" + seconds;
    // отображаем в текстовом поле
    textView.setText(time);
  }
};
```

Для примера получаем текущее время и пытаемся отобразить его в элементе TextView.

Далее определяем объект потока - объект Thread, который принимает объект Runnable. И с помощью метода start() запускаем поток:

```
// Определяем объект Thread - новый поток
Thread thread = new Thread(runnable);
// Запускаем поток
thread.start();
```

Вроде ничего сложного. Но если мы запустим приложение и нажмем на кнопку, то мы столкнемся с ошибкой:

```
2020-12-24 18:46:49.083 19491-19536/com.example.threadapp E/AndroidRuntime: FATAL EXCEPTION: Thread-2
    Process: com.example.threadapp, PID: 19491
    android.view.ViewRootImpl$CalledFromWrongThreadException: Only the original thread that created a view hierarchy can touch its views.
        at android.view.ViewRootImpl.checkThread(ViewRootImpl.java:7313)
        at android.view.ViewRootImpl.requestLayout(ViewRootImpl.java:1161)
        at android.view.View.requestLayout(View.java:21995)
        at android.view.View.requestLayout(View.java:21995)
        at android.view.View.requestLayout(View.java:21995)
        at android.view.View.requestLayout(View.java:21995)
        at android.view.View.requestLayout(View.java:21995)
        at android.view.View.requestLayout(View.java:21995)
        at androidx.constraintlayout.widget.ConstraintLayout.requestLayout(ConstraintLayout.java:3239)
        at android.view.View.requestLayout(View.java:21995)
        at android.widget.TextView.checkForRelayout(TextView.java:8531)
        at android.widget.TextView.setText(TextView.java:5394)
        at android.widget.TextView.setText(TextView.java:5250)
        at android.widget.TextView.setText(TextView.java:5207)
        at com.example.threadapp.MainActivity$1$1.run(MainActivity.java:34)
        at java.lang.Thread.run(Thread.java:764)
```

Многопоточность в Android и Java

Поскольку изменять состояние визуальных элементов, обращаться к ним мы можем только в основном потоке приложения или UI-потоке.

Для решения этой проблемы - взаимодействия во вторичных потоках с элементами графического интерфейса класс **View()** определяет метод **post():**

```
boolean post (Runnable action)
```

В качестве параметра он принимает задачу, которую надо выполнить, и возвращает логическое значение - true, если задача Runnable успешно помещена в очередь сообщение, или false, если не удалось разместить в очереди

Также у класса View есть аналогичный метод:

postDelayed():

boolean postDelayed (Runnable action, long millsec)

Он также запускает задачу, только через определенный промежуток времени в миллисекундах, который указывается во втором параметре.

Так, изменим код **MainActivity** следующим образом

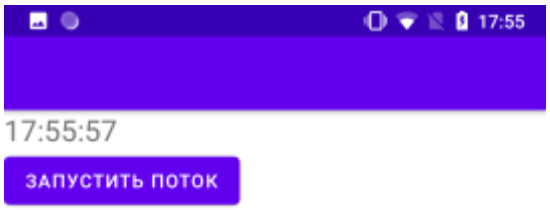
```
package com.example.threadapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import java.util.Calendar;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        TextView textView = findViewById(R.id.textView);
        Button button = findViewById(R.id.button);
```

```
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Определяем объект Runnable
                Runnable runnable = new Runnable() {
                    @Override
                    public void run() {
                        // получаем текущее время
                        Calendar c = Calendar.getInstance();
                        int hours = c.get(Calendar.HOUR_OF_DAY);
                        int minutes = c.get(Calendar.MINUTE);
                        int seconds = c.get(Calendar.SECOND);
                        String time = hours + ":" + minutes + ":" + seconds;
                        // отображаем в текстовом поле
                        textView.post(new Runnable() {
                            public void run() {
                                textView.setText(time);
                            }
                        });
                    }
                };
                // Определяем объект Thread - новый поток
                Thread thread = new Thread(runnable);
                // Запускаем поток
                thread.start();
            }
        });
    }
}
```

Теперь для обновления TextView применяется метод post:

```
textView.post(new Runnable() {
        public void run() {
            textView.setText(time);
    }
});
```

То есть здесь в методе run() передаемого в метод post() объекта Runnable мы можем обращаться к элементам визуального интерфейса и взаимодействовать с ними.



Многопоточность в Android и Java, Runnable и метод View.post

Подобным образом можно работать и с другими виджетами, которые наследуются от класса View.

Потоки, фрагменты и ViewModel

При использовании вторичных потоков следует учитывать следующий момент. Более оптимальным способом является работа потоков с фрагментом, нежели непосредственно с activity. Например, определим в файле activity_main.xml следующий интерфейс:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/progressBtn"
        android:text="Запуск"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toTopOf="@id/statusView"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

    <TextView
        android:id="@+id/statusView"
        android:text="Статус"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toTopOf="@id/indicator"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toBottomOf="@id/progressBtn" />
    <ProgressBar
        android:id="@+id/indicator"
        style="@android:style/Widget.ProgressBar.Horizontal"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:max="100"
        android:progress="0"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/statusView"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Здесь определена кнопка для запуска вторичной задачи и элементы TextView и ProgressBar, которые отображают индикацию выполнения задачи.

В классе **MainActivity** определим следующий код:

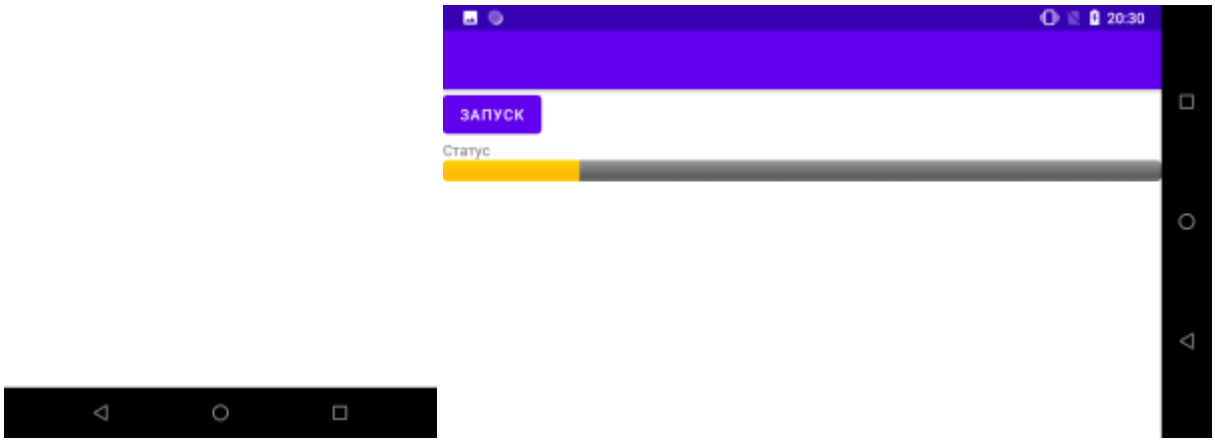
```
package com.example.threadapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ProgressBar;
```

```
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    int currentValue = 0;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ProgressBar indicatorBar = findViewById(R.id.indicator);
        TextView statusView = findViewById(R.id.statusView);
        Button btnFetch = findViewById(R.id.progressBtn);
        btnFetch.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                Runnable runnable = new Runnable() {
                    @Override
                    public void run() {

                        for(; currentValue <= 100; currentValue++){
                            try {
                                statusView.post(new Runnable() {
                                    public void run() {
                                        indicatorBar.setProgress(currentValue);
                                        statusView.setText("Статус: " + currentValue);
                                    }
                                });
```

```
                    Thread.sleep(400);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    };
    Thread thread = new Thread(runnable);
    thread.start();
  }
 });
 }
}
```

Здесь по нажатию кнопки мы запускаем задачу Runnable, в которой в цикле от 0 до 100 изменяем показатели ProgressBar и TextView, имитируя некоторую долгую работу.

Однако если в процессе работы задачи мы изменим ориентацию мобильного устройства, то произойдет пересоздание activity, и приложение перестанет работать должным образом.



Runnable и альбомный режим в Android и Java Runnable, Thread Fragment и фрагменты в Android

В данном случае проблема упирается в состояние, которым оперирует поток, а именно - переменную currentValue, к значению которой привязаны виджеты в Activity.

Добавление ViewModel

Для подобных случаев в качестве решения проблемы предлагается использовать **ViewModel**. Итак, добавим в ту же папку, где находится файл MainActivity.java, новый класс **MyViewModel** со следующим кодом:

```
package com.example.threadapp;


import androidx.lifecycle.LiveData;
import androidx.lifecycle.MutableLiveData;
import androidx.lifecycle.ViewModel;
```

```
public class MyViewModel extends ViewModel {


    private MutableLiveData<Boolean> isStarted = new
MutableLiveData<Boolean>(false);
    private MutableLiveData<Integer> value;
    public LiveData<Integer> getValue() {
        if (value == null) {
            value = new MutableLiveData<Integer>(0);
        }
        return value;
    }
    public void execute(){


        if(!isStarted.getValue()){
            isStarted.postValue(true);
            Runnable runnable = new Runnable() {
                @Override
                public void run() {


                    for(int i = value.getValue();  i <= 100; i++){
                        try {
                            value.postValue(i);
                            Thread.sleep(400);
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
                }
            };
```

```
            Thread thread = new Thread(runnable);
            thread.start();
        }
    }
}
```

Итак, здесь определен класс MyViewModel, который унаследован от класса **ViewModel**, специально предназначенного для хранения и управления состоянием или моделью.

В качестве состояния здесь определены для объекта. В первую очередь, это числовое значение, к которым будут привязан виджеты MainActivity. И во-вторых, нам нужен некоторый индикатор того, что поток уже запущен, чтобы по нажатию на кнопку не было запущено лишних потоков.

Для хранения числового значения предназначена переменная value:

```
private MutableLiveData<Integer> value;
```

Для привязки к этому значению оно имеет тип MutableLiveData. А поскольку мы будем хранить в этой переменной числовое значение, то тип переменной типизирован типом Integer.

Для доступа извне класса к этому значению определен метод etValue, который имеет тип LiveData и который при первом обращении к переменной устанавливает 0, либо просто возвращает значение переменной:

```
public LiveData<Integer> getValue() {
    if (value == null) {
        value = new MutableLiveData<Integer>(0);
    }
    return value;
}
```

Для индикации, запущен ли поток, определена переменная isStarted, которая хранит значение типа Boolean, то есть фактически true или false. По умолчанию она имеет значение false (то есть поток не запущен).

Для изменения числового значения, к которому будут привязаны виджеты, определен метод execute(). Он запускает поток, если поток не запущен:

```
if(!isStarted.getValue()){
```

Далее переключает значение переменной isStarted на true, поскольку мы запускаем поток.

В потоке также запускается цикл

```
for(int i = value.getValue();  i <= 100; i++){
```

И в данном случае мы пользуемся преимуществом класса ViewModel, который позволяет автоматически сохранять свое состояние.

Причем счетчик цикла в качестве начального значения берет значение из переменной value и увеличается на единицу, пока не дойдет до ста.

В самом цикле изменяется значение переменной value с помощью передачи значения в метод postValue()

```
value.postValue(i);
```

Таким образом, в цикле осуществится проход от 0 до 100, и при каждой итерации цикла будет изменяться значение переменной value.

Теперь задействуем наш класс MyViewModel и для этого изменим код класса **MainActivity:**

```
package com.example.threadapp;

import androidx.appcompat.app.AppCompatActivity;

import androidx.lifecycle.ViewModelProvider;
```

```
import android.os.Bundle;
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ProgressBar indicatorBar = findViewById(R.id.indicator);
        TextView statusView = findViewById(R.id.statusView);
        Button btnFetch = findViewById(R.id.progressBtn);
        MyViewModel model = new
ViewModelProvider(this).get(MyViewModel.class);

        model.getValue().observe(this, value -> {
            indicatorBar.setProgress(value);
            statusView.setText("Статус: " + value);
        });
        btnFetch.setOnClickListener(v -> model.execute());
    }
}
```

Чтобы задействовать MyViewModel, создаем объект класса ViewModelProvider, в конструктор которого передается объект-владелец ViewModel. В данном случае это текущий объект MainActivity:

```
new ViewModelProvider(this)
```

И далее с помощью метода get() создаем объект класса ViewModel, который будет использоваться в объекте MainActivity.

```
MyViewModel model = new ViewModelProvider(this).get(MyViewModel.class);
```

Получив объект MyViewModel, определяем прослушивание изменений его переменной value с помощью метода observe:

```
model.getValue().observe(this, value -> {
    indicatorBar.setProgress(value);
    statusView.setText("Статус: " + value);
});
```

Метод observe() в качестве первого параметра принимает владельца функции обсервера - в данном случае текущий объект MainActivity. А в качестве второго параметра - функцию обсервера (а точнее объект интерфейса Observer). Функция обсервера принимает один параметр - новое значение отслеживаемой переменной (то есть в данном случае переменной value). Получив новое значение переменной value, мы изменяем параметры виджетов.

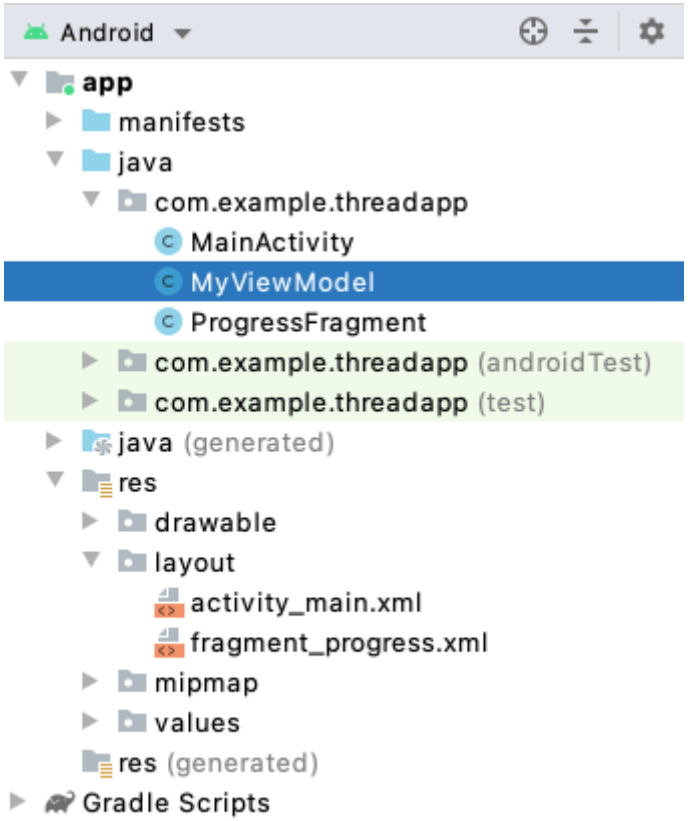
Таким образом, при каждом изменении значения в переменной value виджеты получат ее новое значение.

Теперь если мы запустим приложение, то вне зависимости от смены ориентации моильного устройства фоновая задача будет продолжать свою работу:

Runnable и альбомный режим в Android и Java Runnable, Thread Fragment setRetainInstance и фрагменты в Android и Java

Использование фрагментов

Аналогично мы можем использовать фрагементы. Итак, добавим в проект новый фрагмент, который назовем **ProgressFragment.**



ProgressBar во фрагменте в Android

Определим для него новый файл разметки интерфейса **fragment_progress.xml:**

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/progressBtn"
        android:text="Запуск"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toTopOf="@id/statusView"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

    <TextView
        android:id="@+id/statusView"
        android:text="Статус"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toTopOf="@id/indicator"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toBottomOf="@id/progressBtn" />
    <ProgressBar
        android:id="@+id/indicator"
        style="@android:style/Widget.ProgressBar.Horizontal"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:max="100"
        android:progress="0"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/statusView"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Сам класс фрагмента ProgressFragment изменим следующим образом:

```
package com.example.threadapp;
```

```
import android.os.Bundle;
import androidx.fragment.app.Fragment;
import androidx.lifecycle.ViewModelProvider;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.TextView;


public class ProgressFragment extends Fragment {


   @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {


       View view = inflater.inflate(R.layout.fragment_progress, container, false);


       ProgressBar indicatorBar = (ProgressBar)
view.findViewById(R.id.indicator);
       TextView statusView = (TextView) view.findViewById(R.id.statusView);
       Button btnFetch = (Button)view.findViewById(R.id.progressBtn);


       MyViewModel model = new
ViewModelProvider(requireActivity()).get(MyViewModel.class);


       model.getValue().observe(getViewLifecycleOwner(), value -> {
          indicatorBar.setProgress(value);
```

```
            statusView.setText("Статус: " + value);
        });
        btnFetch.setOnClickListener(v -> model.execute());
        return view;
    }
}
```

Здесь аналогичным образом применяется класс MyViewModel. Единственно для получения ассоциируемой с фрагментом Activity здесь применяется метод **requireActivity().** А для получения владельца жизненного цикла - метод getViewLifecycleOwner.

Теперь свяжем фрагмент с activity. Для этого определим в файле **activity_main.xml** следующий код:

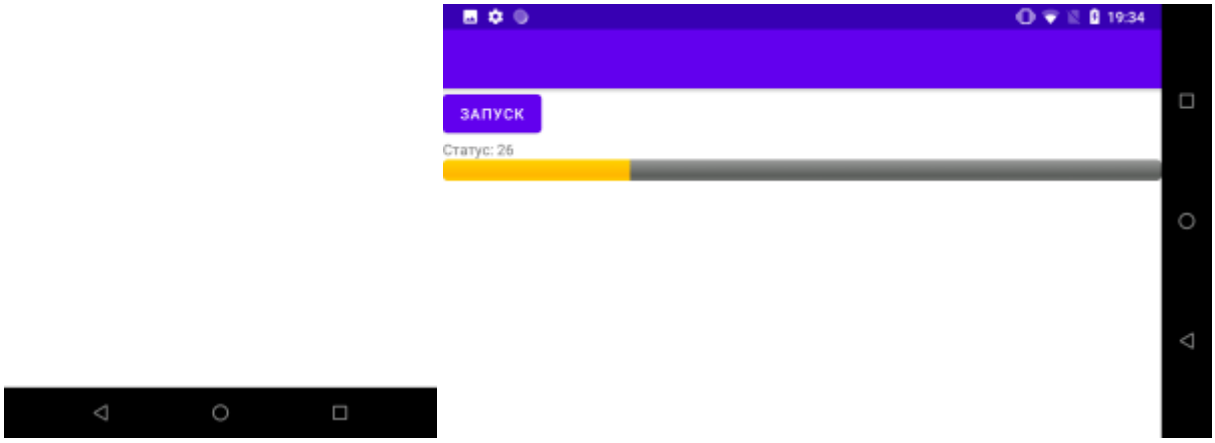
```
<androidx.fragment.app.FragmentContainerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:name="com.example.threadapp.ProgressFragment" />
```

А сам класс **MainActivity** сократим:

```
package com.example.threadapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class MainActivity extends AppCompatActivity {
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

И код с фрагментом будет работать аналогично:



Runnable и альбомный режим в Android и Java Runnable, Thread Fragment setRetainInstance и фрагменты в Android и Java

Класс AsyncTask

В прошлых материалах был описан общий подход, который применяется сейчас для запуска в приложении нового потока и обновления в нем пользовательского интерфейса. Рассмотрим другой подход, который представляет класс AsyncTask. Хотя примение AsyncTask в современных приложениях Android устарел. Тем не менее, поскольку он по-прежнему широко применяется, также рассмотрим его.

Чтобы использовать AsyncTask, нам надо:

1. Создать класс, производный от AsyncTask (как правило, для этого создается внутренний класс в activity или во фрагменте)

2. Переопределить один или несколько методов AsyncTask для выполнения некоторой работы в фоновом режиме

3. При необходимости создать объект AsyncTask и вызывать его метод execute() для начала работы

Итак, создадим простейшее приложение с использованием AsyncTask. Определим следующую разметку интерфейса в файле activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:orientation="vertical">

    <LinearLayout
        android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="22sp"
        android:id="@+id/clicksView"
        android:text="Clicks: 0"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/clicksBtn"
        android:text="Click" />
</LinearLayout>

<Button
    android:id="@+id/progressBtn"
    android:text="Запуск"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

<TextView
    android:id="@+id/statusView"
    android:text="Статус"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
<ProgressBar
    android:id="@+id/indicator"
    style="@android:style/Widget.ProgressBar.Horizontal"
    android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:max="100"
        android:progress="0" />
</LinearLayout>
```

Здесь определена кнопка для запуска фонового потока, а также текстовое поле и прогрессбар для индикации выполнения задачи. Кроме того, здесь определены дополнительная кнопка, которая увеличивает числов кликов, и текстовое поле, оторое выводит число кликов.

Далее определим в классе **MainActivity** следующий код:

```
import android.os.AsyncTask;
import android.os.SystemClock;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    int[] integers=null;
    int clicks = 0;
    ProgressBar indicatorBar;
    TextView statusView;
    TextView clicksView;
    Button progressBtn;
    Button clicksBtn;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    integers = new int[100];
    for(int i=0;i<100;i++) {
        integers[i] = i + 1;
    }
    indicatorBar = (ProgressBar) findViewById(R.id.indicator);
    statusView = findViewById(R.id.statusView);
    progressBtn = findViewById(R.id.progressBtn);
    progressBtn.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {

            new ProgressTask().execute();
        }
    });

    clicksView = findViewById(R.id.clicksView);
    clicksBtn = findViewById(R.id.clicksBtn);
    clicksBtn.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {

            clicks++;
            clicksView.setText("Clicks: " + clicks);
        }
```

```
        });
    }

    class ProgressTask extends AsyncTask<Void, Integer, Void> {
        @Override
        protected Void doInBackground(Void... unused) {
            for (int i = 0; i<integers.length;i++) {

                publishProgress(i);
                SystemClock.sleep(400);
            }
            return(null);
        }
        @Override
        protected void onProgressUpdate(Integer... items) {
            indicatorBar.setProgress(items[0]+1);
            statusView.setText("Статус: " + String.valueOf(items[0]+1));
        }
        @Override
        protected void onPostExecute(Void unused) {
            Toast.makeText(getApplicationContext(), "Задача завершена",
Toast.LENGTH_SHORT)
                    .show();
        }
    }
}
```

Класс задачи ProgressTask определен как внутренний класс. Он наследуется не просто от AsyncTask, а от его типизированной версии AsyncTask<Void, Integer, Void>. Она типизируется тремя типами:

- Класс для хранения информации, которая нужна для обработки задачи

- Тип объектов, которые используются для индикации процесса выполнения задачи

- Тип результата задачи

Эти типы могут быть представлены разными классами. В данном случае сущность задачи будет состоять в переборе массива integers, представляющего набор элементов Integer. И здесь нам не надо передавать в задачу никакой объект, поэтому первый тип идет как **Void**.

Для индикации перебора используются целые числа, которые показывают, какой объект из массива мы в данный момент перебираем. Поэтому в качестве второго типа используется Integer.

В качестве третьего типа используется опять Void, поскольку в данном случае не надо ничего возвращать из задачи.

AsyncTask содержит четыре метода, которые можно переопределить:

- Метод doInBackground(): выполняется в фоновом потоке, должен возвращать определенный результат

- Метод onPreExecute(): вызывается из главного потока перед запуском метода doInBackground()

- Метод onPostExecute(): выполняется из главного потока после завершения работы метода doInBackground()

- Метод onProgressUpdate(): позволяет сигнализировать пользователю о выполнении фонового потока

Так как метод doInBackground() не принимает ничего и не возвращает ничего, то в качестве его параметра используется Void... - массив Void, и в качестве возвращаемого типа - тоже Void. Эти типы соответствуют первому и третьему типам в AsyncTask<Void, Integer, Void>.

Метод doInBackground() перебирает массив и при каждой итерации уведомляет систему с помощью метода **publishProgress(item).** Так как в нашем случае для индикации используются целые числа, то параметр item должен представлять целое число.

Метод onProgressUpdate(Integer... items) получает переданное выше число и применяет его для настройки текстового поля и прогрессбара.

Метод onPostExecute() выполняется после завершения задачи и в качестве параметра принимает объект, возвращаемый методом doInBackground() - то есть в данном случае объект типа Void. Чтобы сигнализировать окончание работы, здесь выводится на экран всплывающее сообщение.

Запустим приложение. Запустим задачу, нажав на кнопку:

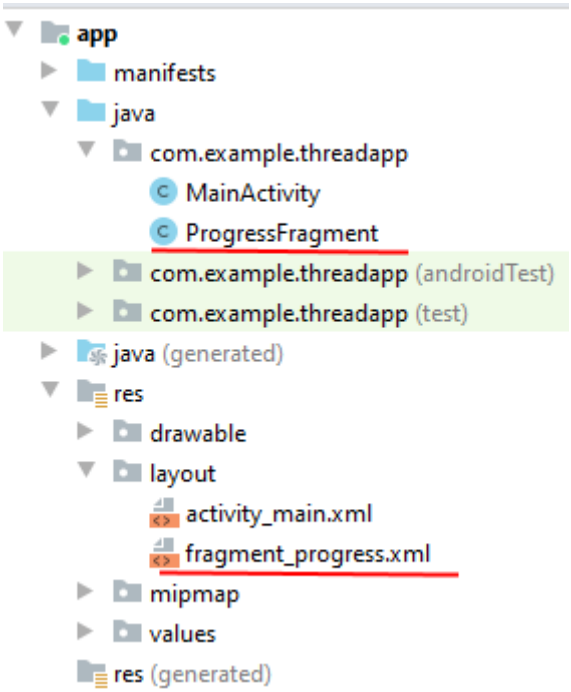
AsyncTask в Android

При этом пока выполняется задача, мы можем параллельно нажимать на вторую кнопку и увеличивать число кликов, либо выполнять какую-то другую работу в приложении.

AsyncTask и фрагменты

При использовании AsyncTask следует учитывать следующий момент. Более оптимальным способом является работа AsyncTask с фрагментом, нежели непосредственно с activity. Например, если мы возьмем проект из прошлого материала, запустим приложение и изменим ориентацию мобильного устройства, то произойдет пересоздание activity. В случае изменения орентации устройства поток AsyncTask будет продолжать обращаться к старой activity, вместо новой. Поэтому в этом случае лучше использовать фрагменты.

Итак, возьмем проект из прошлого материала и добавим в него новый фрагмент, который назовем **ProgressFragment.**



ProgressBar во фрагменте в Android

Определим для него новый файл разметки интерфейса **fragment_progress.xml**:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_progress"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:orientation="vertical">
    <Button
        android:id="@+id/progressBtn"
        android:text="Запуск"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <TextView
        android:id="@+id/statusView"
        android:text="Статус"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <ProgressBar
        android:id="@+id/indicator"
        style="@android:style/Widget.ProgressBar.Horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:max="100"
        android:progress="0" />
</LinearLayout>
```

Сам класс фрагмента **ProgressFragment** изменим следующим образом:

```
package com.example.eugene.asyncapp;
```

```
import android.widget.Button;
import android.os.AsyncTask;
import android.os.Bundle;
import android.app.Fragment;
import android.os.SystemClock;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ProgressBar;
import android.widget.TextView;
import android.widget.Toast;
import android.view.View.OnClickListener;

public class ProgressFragment extends Fragment {

    int[] integers=null;
    ProgressBar indicatorBar;
    TextView statusView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_progress, container, false);
        integers = new int[100];
```

```
    for(int i=0;i<100;i++) {
        integers[i] = i + 1;
    }
    indicatorBar = (ProgressBar) view.findViewById(R.id.indicator);
    statusView = (TextView) view.findViewById(R.id.statusView);
    Button btnFetch = (Button)view.findViewById(R.id.progressBtn);
    btnFetch.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {

            new ProgressTask().execute();
        }
    });
    return view;
}

class ProgressTask extends AsyncTask<Void, Integer, Void> {
    @Override
    protected Void doInBackground(Void... unused) {
        for (int i = 0; i<integers.length;i++) {

            publishProgress(i);
            SystemClock.sleep(400);
        }
        return null;
    }
    @Override
    protected void onProgressUpdate(Integer... items) {
        indicatorBar.setProgress(items[0]+1);
```

```
            statusView.setText("Статус: " + String.valueOf(items[0]+1));
        }
        @Override
        protected void onPostExecute(Void unused) {
            Toast.makeText(getActivity(), "Задача завершена",
Toast.LENGTH_SHORT)
                    .show();
        }
    }
}
```

Здесь определены все те действия, которые были рассмотрены в прошлой теме и которые ранее находились в классе MainActivity. Особо стоит отметить вызов **setRetainInstance(true)** в методе onCreate(), который позволяет сохранять состояние фрагмента вне зависимости от изменения ориентации.

Теперь свяжем фрагмент с activity. Для этого определим в файле **activity_main.xml** следующий код:

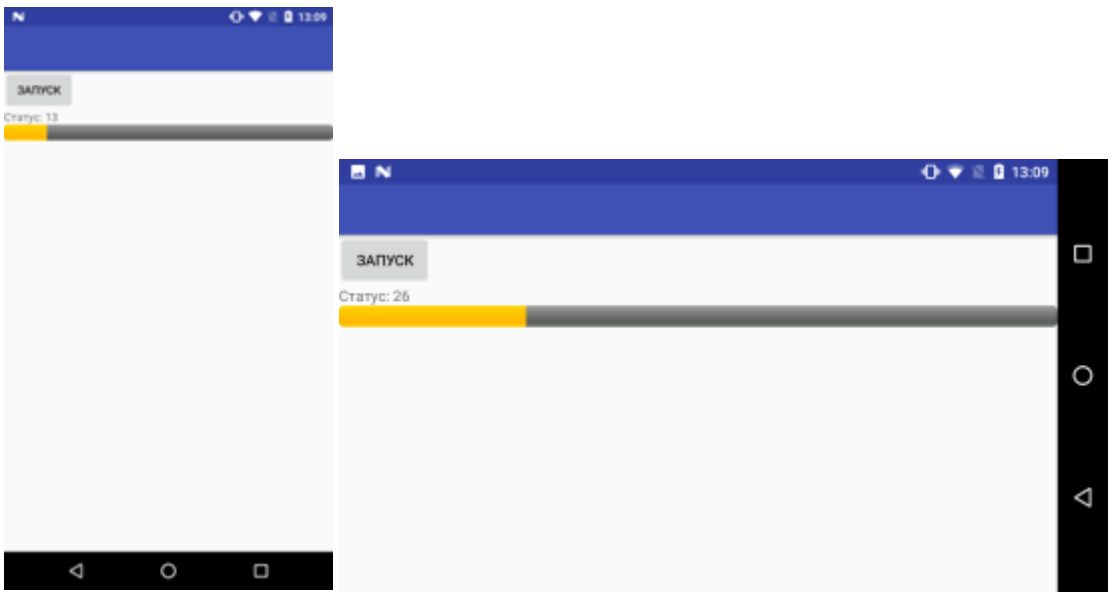
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment
        android:id="@+id/progressFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
```

```
        android:name="com.example.eugene.asyncapp.ProgressFragment"/>
</LinearLayout>
```

А сам класс MainActivity сократим:

```
package com.example.eugene.asyncapp;

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

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Теперь если мы запустим приложение, то вне зависимости от смены ориентации мобильного устройства фоновая задача будет продолжать свою работу:



AsyncTask и альбомный режим в Android AsyncTask и фрагменты в Android

Работа с сетью. WebView

WebView

WebView представляет простейший элемент для рендеринга html-кода, базирующийся на движке WebKit. Благодаря этому мы можем использовать WebView как примитивный веб-браузер, просматривая через него контент из сети интернет. Использование движка WebKit гарантирует, что отображение контента будет происходить примерно также, как и в других браузерах, построенных на этом движке - Google Chrome и Safari.

Некоторые основные методы класса WebView:

- **boolean canGoBack():** возвращает true, если перед текущей веб-страницей в истории навигации WebView еще есть страницы
- **boolean canGoForward()**: возвращает true, если после текущей веб-страницей в истории навигации WebView еще есть страницы
- **void clearCache(boolean includeDiskFiles)**: очищает кэш WebView
- **void clearFormData():** очищает данный автозаполнения полей форм
- **void clearHistory():** очищает историю навигации WebView
- **String getUrl():** возвращает адрес текущей веб-страницы

- **void goBack():** переходит к предыдущей веб-странице в истории навигации

- **void goForward():** переходит к следующей веб-странице в истории навигации

- **void loadData(String data, String mimeType, String encoding):** загружает в веб-браузере данные в виде html-кода, используя указанный mime-тип и кодировку

- **void loadDataWithBaseURL (String baseUrl, String data, String mimeType, String encoding, String historyUrl)**: также загружает в веб-браузере данные в виде html-кода, используя указанный mime-тип и кодировку, как и метод loadData(). Однако, кроме того, в качестве первого параметра принимает валидный адрес, с которым ассоциируется загруженные данные.

Зачем нужен этот метод, если есть loadData()? Содержимое, загружаемое методом loadData(), в качестве значения для window.origin будет иметь значение null, и таким образом, источник загружаемого содержимого не сможет пройти проверку на достоверность. Метод loadDataWithBaseURL() с валидными адресами (протокол может быть и HTTP, и HTTPS) позволяет установить источник содержимого.

- **void loadUrl(String url):** загружает веб-страницу по определенному адресу

- **void postUrl(String url, byte[] postData):** отправляет данные с помощью запроса типа "POST" по определенному адресу

- **void zoomBy(float zoomFactor):** изменяет масштаб на опредленный коэффициент

- **boolean zoomIn():** увеличивает масштаб

- **boolean zoomOut():** уменьшает масштаб

Работать с WebView очень просто. Определим данный элемент в разметке layout:

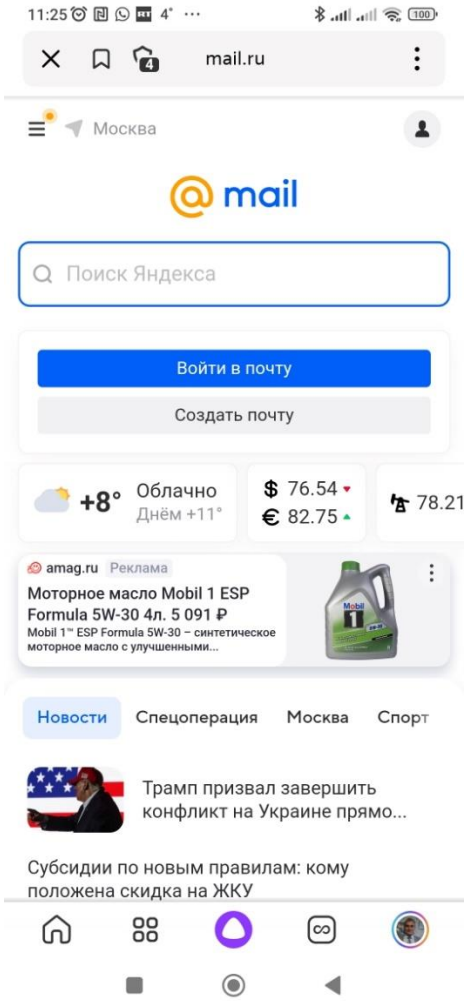
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <WebView
        android:id="@+id/webBrowser"
        android:layout_width="0dp"
        android:layout_height="0dp"

        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Для получения доступа к интернету из приложения, необходимо указать в файле манифеста AndroidManifest.xml соответствующее разрешение:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Чтобы загрузить определенную страницу в WebView, через метод loadUrl() надо установить ее адрес:

```
package com.example.viewapp;
import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.webkit.WebView;
public class MainActivity extends AppCompatActivity {
    @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);

       WebView browser=findViewById(R.id.webBrowser);
       browser.loadUrl("https://mail.ru");
   }
}
```



Элемент WebView в Android и Java

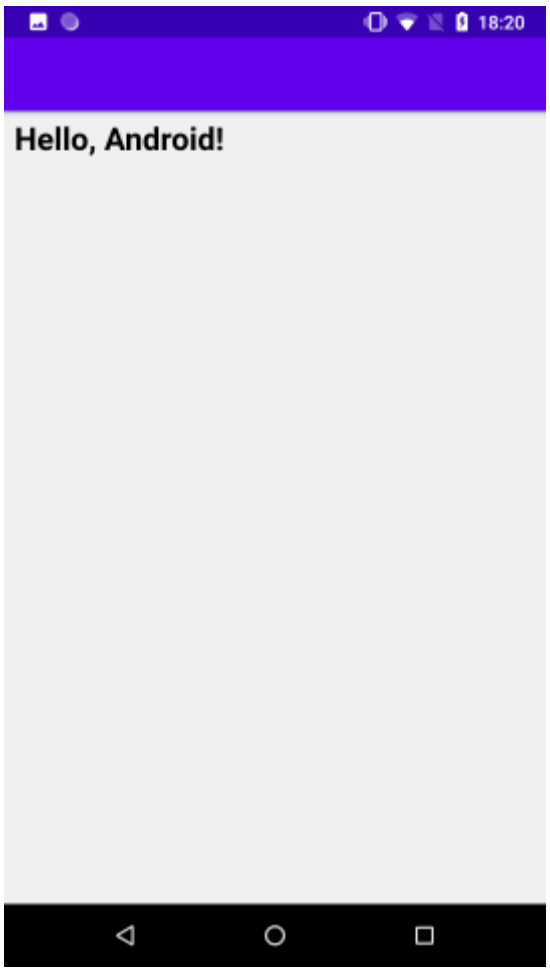
Вместо определения элемента в layout мы можем создать WebView в коде Activity:

```
WebView browser = new WebView(this);

setContentView(browser);

browser.loadUrl("http://mail.ru");
```

Загружаем конкретную Web страницу с помощью метода loadData():

```
WebView browser= findViewById(R.id.webBrowser);

browser.loadData("<html><body><h2>Hello, Android!</h2>
</body></html>", "text/html", "UTF-8");
```

Первым параметром метод принимает строку кода html, во втором - тип содержимого, а в третьем - кодировку.



Load data in WebView в Android и Java

JavaScript

По умолчанию в WebView отключен javascript, чтобы его включить надо применить метод **setJavaScriptEnabled(true)** объекта WebSettings:

```
import android.webkit.WebSettings;
//.....................................
WebView browser = findViewById(R.id.webBrowser);
WebSettings webSettings = browser.getSettings();
webSettings.setJavaScriptEnabled(true);
```

Загрузка данных и класс HttpURLConnection

На сегодняшний день если не все, то большинство Android-устройств имеют доступ к сети интернет. А большое количество мобильных приложений так или иначе взаимодействуют с средой интернет: загружают файлы, авторизуются и получают информацию с внешних веб-сервисов и т.д. Рассмотрим, как мы можем использовать в своем приложении доступ к сети интернет.

Среди стандартных элементов нам доступен виджет WebView, который может загружать контент с определенного url-адреса. Но этим возможности работы с сетью в Android не ограничиваются. Для получения данных с определенного интернет-ресурса мы можем использовать классы **HttpUrlConnection** (для протокола HTTP) и **HttpsUrlConnection** (для протокола HTTPS) из стандартной библиотеки Java.

Итак, создадим новый проект с пустой MainActivity. Первым делом для работы с сетью нам надо установить в файле манифеста **AndroidManifest.xml** соответствующее разрешение:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

В файле **activity_main.xml**, который представляет разметку для MainActivity, определим следующий код:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <Button
```

```
    android:id="@+id/downloadBtn"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Загрузка"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
<WebView
    android:id="@+id/webView"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@id/downloadBtn"
    app:layout_constraintBottom_toTopOf="@id/scrollView" />
<ScrollView
    android:id="@+id/scrollView"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@id/webView"
    app:layout_constraintBottom_toBottomOf="parent">

    <TextView android:id="@+id/content"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</ScrollView>
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

Здесь определена кнопка для загрузки данных, а сами данные для примера загружаются одновременно в виде строки в текстовое поле и в элемент WebView. Так как данных может быть очень много, то текстовое поле помещено в элемент ScrollView.

Поскольку загрузка данных может занять некоторое время, то обращение к интернет-ресурсу определим в отдельном потоке и для этого изменим код **MainActivity** следующим образом:

```
package com.example.httpapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.webkit.WebView;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URL;
import javax.net.ssl.HttpsURLConnection;

public class MainActivity extends AppCompatActivity {
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        TextView contentView = findViewById(R.id.content);
        WebView webView = findViewById(R.id.webView);
        webView.getSettings().setJavaScriptEnabled(true);
        Button btnFetch = findViewById(R.id.downloadBtn);
        btnFetch.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                contentView.setText("Загрузка...");
                new Thread(new Runnable() {
                    public void run() {
                        try{
                            String content = getContent("https://stackoverflow.com/");
                            webView.post(new Runnable() {
                                public void run() {

webView.loadDataWithBaseURL("https://stackoverflow.com/",content,
"text/html", "UTF-8", "https://stackoverflow.com/");
                                    Toast.makeText(getApplicationContext(), "Данные
загружены", Toast.LENGTH_SHORT).show();
                                }
                            });
                            contentView.post(new Runnable() {
                                public void run() {
                                    contentView.setText(content);
                                }
```

```
                    });
                }
                catch (IOException ex){
                    contentView.post(new Runnable() {
                        public void run() {
                            contentView.setText("Ошибка: " + ex.getMessage());
                            Toast.makeText(getApplicationContext(), "Ошибка",
Toast.LENGTH_SHORT).show();
                        }
                    });
                }
            }
        }).start();
    }
  });
}
private String getContent(String path) throws IOException {
    BufferedReader reader=null;
    InputStream stream = null;
    HttpsURLConnection connection = null;
    try {
        URL url=new URL(path);
        connection =(HttpsURLConnection)url.openConnection();
        connection.setRequestMethod("GET");
        connection.setReadTimeout(10000);
        connection.connect();
        stream = connection.getInputStream();
        reader= new BufferedReader(new InputStreamReader(stream));
        StringBuilder buf=new StringBuilder();
```

```
            String line;
            while ((line=reader.readLine()) != null) {
                buf.append(line).append("\n");
            }
            return(buf.toString());
        }
        finally {
            if (reader != null) {
                reader.close();
            }
            if (stream != null) {
                stream.close();
            }
            if (connection != null) {
                connection.disconnect();
            }
        }
    }
}
```

Непосредственно для самой загрузки определен метод getContent(), который будет загружать веб-страницу с помощью класса **HttpsURLConnection** и возвращать код загруженной страницы в виде строки.

Вначале создается элемент HttpsURLConnection:

```
URL url=new URL(path);
connection =(HttpsURLConnection)url.openConnection();
connection.setRequestMethod("GET"); // установка метода получения данных - GET
```

connection.setReadTimeout(10000); // установка таймаута перед выполнением - 10 000 миллисекунд

connection.connect(); // подключаемся к ресурсу

После подключение происходит считывание со входного потока:

stream = connection.getInputStream();

reader= new BufferedReader(new InputStreamReader(stream));

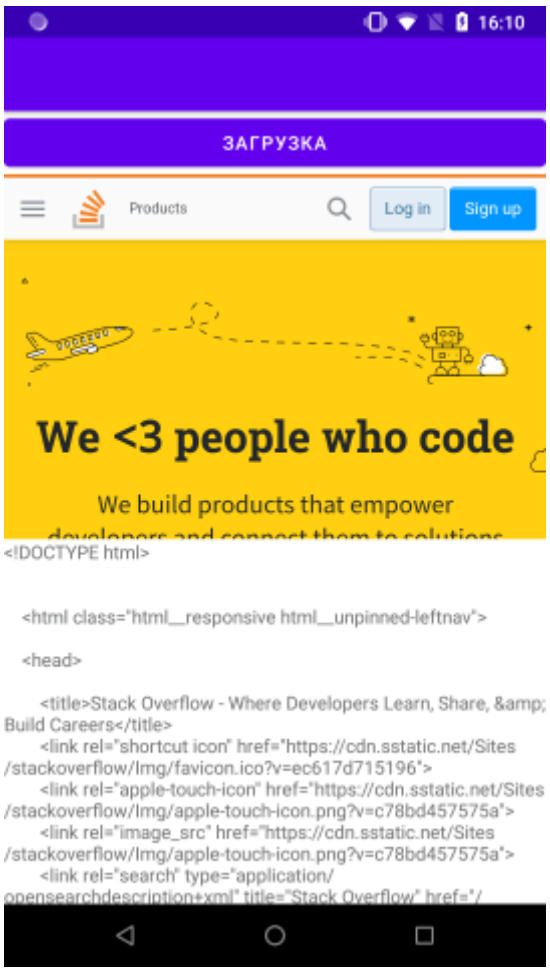
Используя входной поток, мы можем считать его в строку.

Этот метод getContent() затем будет вызываться в обработчике нажатия кнопки:

```
Button btnFetch = (Button)findViewById(R.id.downloadBtn);
btnFetch.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        contentView.setText("Загрузка...");
        new Thread(new Runnable() {
            public void run() {
                try{
                    String content = getContent("https://stackoverflow.com/");
```

Поскольку загрузка может занять долгое время, то метод getContent() в отдельном потоке с помощью объектов Thread и Runnable. Для примера в данном случае обращение идет к ресурсу "https://stackoverflow.com/".

Запустим приложение и нажмем на кнопку. И при наличии интернета приложение загрузит гравную страницу с "https://stackoverflow.com/" и отобразит ее в WebView и TextView:



HttpsURLConnection и загрузка данных из интернета в Android и Java

Конечно, данный способ вряд ли подходит для просмотра интернет-страниц, однако таким образом, мы можем получать какие-либо данные (не интернет-страницы) от различных веб-сервисов, например, в формате xml или json (например, различные курсы валют, показатели погоды), используя специальные api, и затем после обработки показывать их пользователю.

Задание

1. Реализовать пример с использованием потоков
2. Реализовать потоки, фрагменты и ViewModel
3. Изучить порядок и реализовать применение класса AsyncTask
4. Реализовать применение AsyncTask с фрагментом
5. Изучить работу с сетью и классом WebView. Реализовать пример.
6. Реализовать пример загрузки данных с применением класса HttpsURLConnection