Практическая работа 15

Методические материалы.

Сервисы. Диалоговые окна.

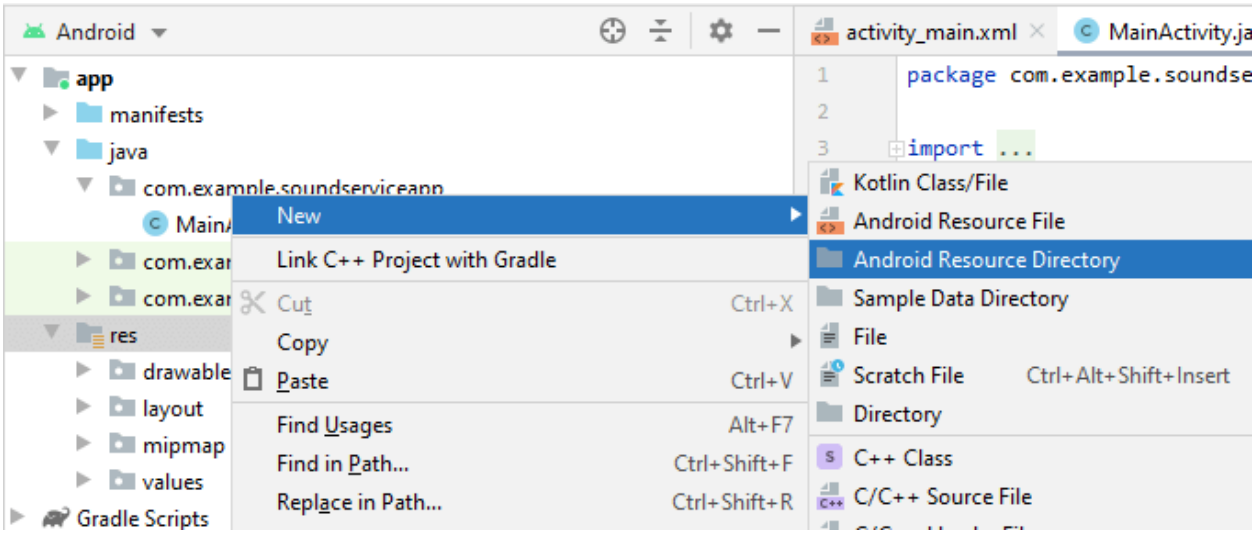
Введение в сервисы Android

Сервисы представляют собой особую организацию приложения. В отличие от activity они не требуют наличия визуального интерфейса. Сервисы позволяют выполнять долговременные задачи без вмешательства пользователя.

Все сервисы наследуются от класса **Service** и проходят следующие этапы жизненного цикла:

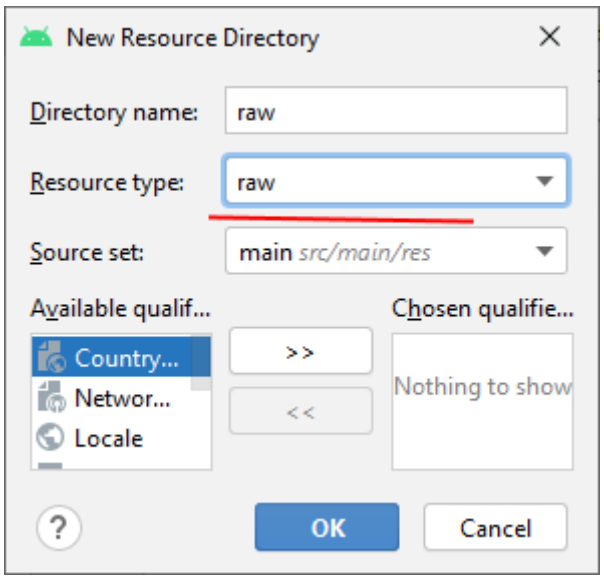
- Метод onCreate(): вызывается при создании сервиса
- Метод onStartCommand(): вызывается при получении сервисом команды, отправленной с помощью метода startService()
- Метод onBind(): вызывается при закреплении клиента за сервисом с помощью метода bindService()
- Метод onDestroy(): вызывается при завершении работы сервиса

Создадим простейшее приложение с сервисом. Наш сервис будет воспроизводить музыкальный файл. И вначале добавим в проект в каталог res папку **raw**. Для этого нажмем правой кнопкой мыши на каталог res и в контекстном меню выберем пункт **New -> Android Resource Directory**.



Добавление папки raw для сервисов в Android Studio и Java

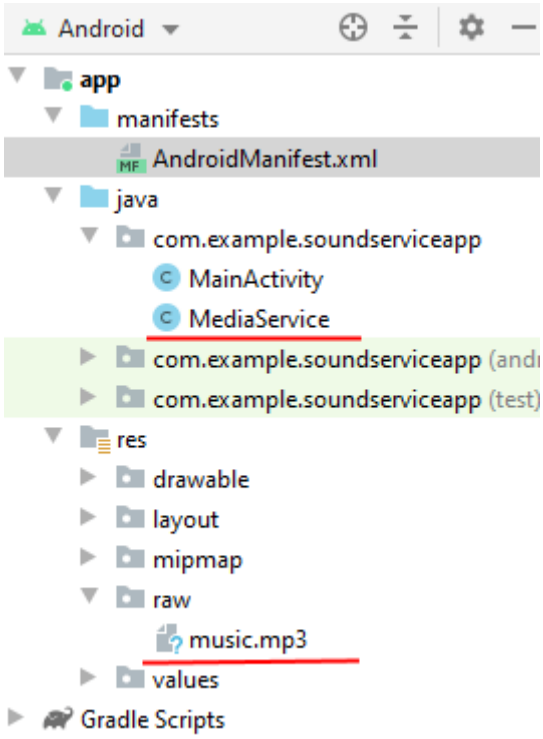
Далее укажем в качестве типа папки - **raw**:



Добавление папки ресурсов raw для сервисов в Android Studio и Java

И поместим в эту папку (**res/raw**) какой-нибудь mp3-файл.

Затем добавим новый класс сервиса. Назовем его **MediaService**. В итоге получится следующий проект:



Добавление сервисов в Android Studio и Java

Для воспроизведения аудио-файла определим в классе **MediaService** следующий код:

```
package com.example.soundserviceapp;

import android.app.Service;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.IBinder;

public class MediaService extends Service {
    MediaPlayer ambientMediaPlayer;
    @Override
    public IBinder onBind(Intent intent) {
```

```
        throw new UnsupportedOperationException("Not yet implemented");
    }
    @Override
    public void onCreate(){
        ambientMediaPlayer=MediaPlayer.create(this, R.raw.music);
        ambientMediaPlayer.setLooping(true);
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId){
        ambientMediaPlayer.start();
        return START_STICKY;
    }
    @Override
    public void onDestroy() {
        ambientMediaPlayer.stop();
    }
}
```

Для воспроизведения музыкального файла сервис будет использовать компонент **MediaPlayer.**

В сервисе переопределяются все четыре метода жизненного цикла. Но по сути метод **onBind()** не имеет никакой реализации.

В методе **onCreate()** инициализируется медиа-проигрыватель с помощью музыкального ресурса, который добавлен в папку res/raw.

В методе **onStartCommand()** начинается воспроизведение.

Метод **onStartCommand()** может возвращать одно из значений, которое предполагает различное поведение в случае, если процесс сервиса был неожиданно завершен системой:

- **START_STICKY**: в этом случае сервис снова возвращается в запущенное состояние, как будто если бы снова был бы вызван метод onStartCommand() без передачи в этот метод объекта Intent

- **START_REDELIVER_INTENT**: в этом случае сервис снова возвращается в запущенное состояние, как будто если бы снова был бы вызван метод onStartCommand() с передачей в этот метод объекта Intent

- **START_NOT_STICKY**: сервис остается в остановленном положении

Метод onDestroy() завершает воспроизведение.

Чтобы управлять сервисом, изменим activity. Сначала добавим в файл **activity_main.xml** пару кнопок для управления сервисом:

```
<?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/start"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
```

```
        android:text="Старт"
        android:onClick="click"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toLeftOf="@id/stop"
        app:layout_constraintTop_toTopOf="parent" />
    <Button
        android:id="@+id/stop"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Стоп"
        android:onClick="click"
        app:layout_constraintLeft_toRightOf="@id/start"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

И изменим код **MainActivity**:

```
package com.example.soundserviceapp;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {

    @Override
```

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

    public void click(View v) {
        Intent i=new Intent(this, MediaService.class);
        if (v.getId()==R.id.start) {
            startService(i);
        }
        else {
            stopService(i);
        }
    }
}
```

Для запуска сервиса используется объект Intent:

```
Intent i=new Intent(this, MediaService.class);
```

Для запуска сервиса в классе **Activity** определен метод **startService()**, в который передается объект **Intent.** Этот метод будет посылать команду сервису и вызывать его метод **onStartCommand()**, а также указывать системе, что сервис должен продолжать работать до тех пор, пока не будет вызван метод stopService().

Метод **stopService()** также определен к классе Activity и принимает объект Intent. Он останавливает работу сервиса, вызывая его метод onDestroy()

И в конце нам надо зарегистрировать сервис в файле манифеста:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.soundserviceapp">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.SoundServiceApp">

        <service
            android:name=".MediaService"
            android:enabled="true"
            android:exported="true" >
        </service>

        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Регистрация сервиса производится в узле application с помощью добавления элемента <service>. В нем определяется атрибут android:name, который хранит название класса сервиса. И кроме того может принимать еще ряд атрибутов:

- android:enabled: если имеет значение "true", то сервис может ли создаваться системой. Значение по умолчанию - "true".

- android:exported: указывает, могут ли компоненты других приложений обращаться к сервису. Если имеет значение "true", то могут, если имеет значение "false", то нет.

- android:icon: значок сервиса, представляет собой ссылку на ресурс drawable

- android:isolatedProcess: если имеет значение true, то сервис может быть запущен как специальный процесс, изолированный от остальной системы.

- android:label: название сервиса, которое отображается пользователю

- android:permission: набор разрешений, которые должно применять приложение для запуска сервиса

- android:process: название процесса, в котором запущен сервис. Как правило, имеет то же название, что и пакет приложения.

Запустим приложение и нажмем на кнопку запуска сервиса:

Сервисы startService в Android и Java

После этого начнется воспроизведение добавленной нами в приложение мелодии.

Диалоговые окна

DatePickerDialog и TimePickerDialog

По умолчанию в Android уже определены два диалоговых окна - **DatePickerDialog** и **TimePickerDialog**, которые позволяют выбрать дату и время.

Кроме установки даты DatePickerDialog позволяет обработать выбор даты с помощью слушателей **OnDateChangedListener** и **OnDateSetListener**. Что позволяет использовать выбранную дату далее в приложении.

Подобным образом TimePickerDialog позволяет обработать выбор времени с помощью слушателей **OnTimeChangedListener** и **OnTimeSetListener**

Пи работе с данными компонентами надо учитывать, что отсчет месяцев в DatePickerDialog начинается с нуля, то есть январь будет иметь номер 0, а декабрь - 11. И аналогично в TimePickerDialog отсчет секунд и минут будет идти с 0 до 59, а часов - с 0 до 23.

Используем DatePickerDialog и TimePickerDialog в приложении. Определим следующую разметку интерфейса в **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/currentDateTime"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        app:layout_constraintBottom_toTopOf="@id/timeButton"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/timeButton"
```

```
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Изменить время"
        android:onClick="setTime"
        app:layout_constraintBottom_toTopOf="@id/dateButton"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/currentDateTime"  />

    <Button
        android:id="@+id/dateButton"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Изменить дату"
        android:onClick="setDate"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/timeButton" />

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

Здесь определены две кнопки для выбора даты и времени и текстовое поле, отображающее выбранные дату и время. И изменим код **MainActivity**:

```
package com.example.dialogsapp;

import androidx.appcompat.app.AppCompatActivity;

import android.app.DatePickerDialog;
import android.app.TimePickerDialog;
```

```
import android.os.Bundle;
import android.text.format.DateUtils;
import android.view.View;
import android.widget.DatePicker;
import android.widget.TextView;
import android.widget.TimePicker;

import java.util.Calendar;

public class MainActivity extends AppCompatActivity {

    TextView currentDateTime;
    Calendar dateAndTime=Calendar.getInstance();
    @Override
    public void onCreate(Bundle savedInstance) {
        super.onCreate(savedInstance);
        setContentView(R.layout.activity_main);
        currentDateTime = findViewById(R.id.currentDateTime);
        setInitialDateTime();
    }

    // отображаем диалоговое окно для выбора даты
    public void setDate(View v) {
        new DatePickerDialog(MainActivity.this, d,
                dateAndTime.get(Calendar.YEAR),
                dateAndTime.get(Calendar.MONTH),
                dateAndTime.get(Calendar.DAY_OF_MONTH))
                .show();
    }
```

```
    // отображаем диалоговое окно для выбора времени
    public void setTime(View v) {
        new TimePickerDialog(MainActivity.this, t,
                dateAndTime.get(Calendar.HOUR_OF_DAY),
                dateAndTime.get(Calendar.MINUTE), true)
                .show();
    }
    // установка начальных даты и времени
    private void setInitialDateTime() {

        currentDateTime.setText(DateUtils.formatDateTime(this,
                dateAndTime.getTimeInMillis(),
                DateUtils.FORMAT_SHOW_DATE |
DateUtils.FORMAT_SHOW_YEAR
                        | DateUtils.FORMAT_SHOW_TIME));
    }

    // установка обработчика выбора времени
    TimePickerDialog.OnTimeSetListener t=new
TimePickerDialog.OnTimeSetListener() {
        public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
            dateAndTime.set(Calendar.HOUR_OF_DAY, hourOfDay);
            dateAndTime.set(Calendar.MINUTE, minute);
            setInitialDateTime();
        }
    };

    // установка обработчика выбора даты
```

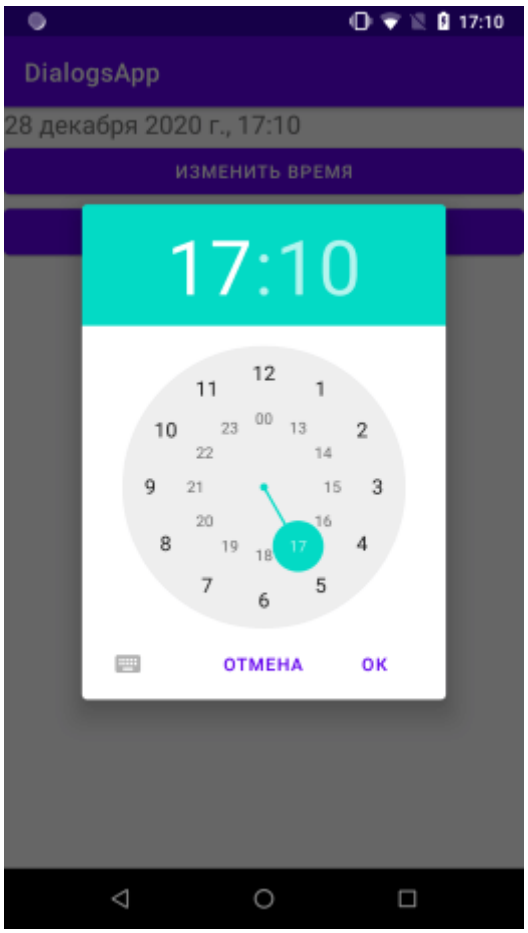
```
   DatePickerDialog.OnDateSetListener d=new
DatePickerDialog.OnDateSetListener() {
      public void onDateSet(DatePicker view, int year, int monthOfYear, int
dayOfMonth) {
         dateAndTime.set(Calendar.YEAR, year);
         dateAndTime.set(Calendar.MONTH, monthOfYear);
         dateAndTime.set(Calendar.DAY_OF_MONTH, dayOfMonth);
         setInitialDateTime();
      }
   };
}
```

Ключевым классом здесь является **java.util.Calendar**, который хранится в стандартной библиоетке классов Java. В методе **setInitialDateTime()** мы получаем из экземпляра этого класса количество миллисекунд **dateAndTime.getTimeInMillis()** и с помощью форматирования выводим на текстовое поле.

Метод **setDate()**, вызываемый по нажатию на кнопку, отображает окно для выбора даты. При создании окна его объекту передается обработчик выбора даты **DatePickerDialog.OnDateSetListener**, который изменяет дату на текстовом поле.

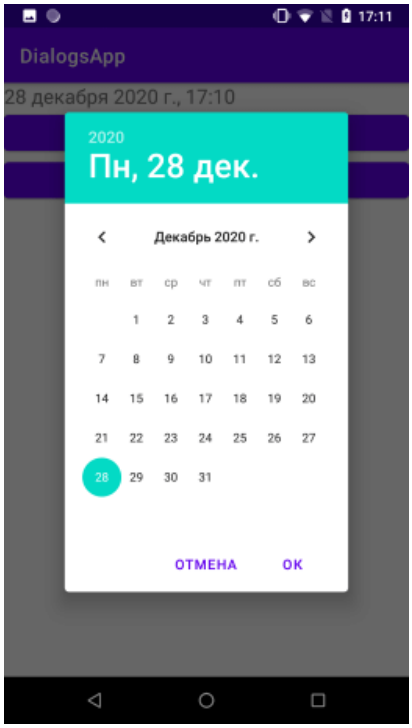
Аналогично метод setTime() отображает окно для выбора времени. Объект окна использует обработчик выбора времени **TimePickerDialog.OnTimeSetListener**, который изменяет время на текстовом поле.

И поле запуска, нажав на кнопку изменения времени, мы сможем установить время:



TimePickerDialog in Android

Аналогично работает окно установки даты:



DatePickerDialog in Android

DialogFragment и создание своих диалоговых окон

Для создания своих диалоговых окон используется компонент **AlertDialog** в связке с классом фрагмента **DialogFragment**. Рассмотрим их применение.

Вначале добавим в проект новый класс фрагмента, который назовем **CustomDialogFragment**:

```
package com.example.dialogsapp;


import android.app.AlertDialog;
import android.app.Dialog;
import android.os.Bundle;
import androidx.fragment.app.DialogFragment;
import androidx.annotation.NonNull;


public class CustomDialogFragment extends DialogFragment {


    @NonNull
    public Dialog onCreateDialog(Bundle savedInstanceState) {


        AlertDialog.Builder builder=new AlertDialog.Builder(getActivity());
        return builder.setTitle("Диалоговое окно").setMessage("Для закрытия окна нажмите ОК").create();
    }
}
```

Класс фрагмента содержит всю стандартную функциональность фрагмента с его жизненным циклом, но при этом наследуется от класса **DialogFragment**, который добавляет ряд дополнительных функций. И для его создания мы можем использовать два способа:

- Переопределение метода onCreateDialog(), который возвращает объект Dialog.

- Использование стандартного метода onCreateView().

Для создания диалогового окна в методе onCreateDialog() применяется класс AlertDialog.Builder. С помощью своих методов он позволяет настроить отображение диалогового окна:

- setTitle: устанавливает заголовок окна

- setView: устанавливает разметку интерфейса окна

- setIcon: устанавливает иконку окна

- setPositiveButton: устанавливает кнопку подтверждения действия

- setNeutralButton: устанавливает "нейтральную" кнопку, действие которой может отличаться от действий подтверждения или отмены

- setNegativeButton: устанавливает кнопку отмены

- setMessage: устанавливает текст диалогового окна, но при использовании setView данный метод необязателен или может рассматриваться в качестве альтернативы, если нам надо просто вывести сообщение.

- create: создает окно

В данном же случае диаговое кно просто выводит некоторое сообщение.

Для вызова этого диалогового окна в файле activity_main.xml определим кнопку:

```
<?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:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Dialog"
        android:onClick="showDialog"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"  />

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

В коде **MainActivity** определим обработчик нажатия кнопки, который будет запускать диалоговое окно:

```
package com.example.dialogsapp;

import androidx.appcompat.app.AppCompatActivity;

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

```
public class MainActivity extends AppCompatActivity {


    @Override
    public void onCreate(Bundle savedInstance) {
        super.onCreate(savedInstance);
        setContentView(R.layout.activity_main);
    }


    public void showDialog(View v) {


        CustomDialogFragment dialog = new CustomDialogFragment();
        dialog.show(getSupportFragmentManager(), "custom");
    }
}
```

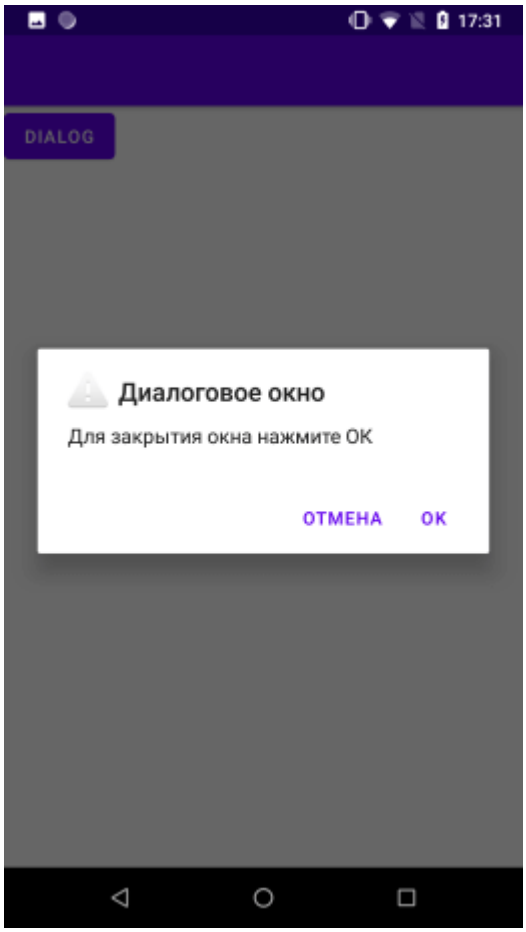
Для вызова диалогового окна создается объект фрагмента **CustomDialogFragment**, затем у него вызывается метод show(). В этот метод передается менеджер фрагментов **FragmentManager** и строка - произвольный тег.

И поле нажатия на кнопку мы сможем ввести данные в диалоговое окно:

DialogFragment и AlertDialog в Android и Java

Теперь немного кастомизируем диалоговое окно:

```
public Dialog onCreateDialog(Bundle savedInstanceState) {

    AlertDialog.Builder builder=new AlertDialog.Builder(getActivity());
    return builder
            .setTitle("Диалоговое окно")
            .setIcon(android.R.drawable.ic_dialog_alert)
            .setMessage("Для закрытия окна нажмите ОК")
            .setPositiveButton("OK", null)
            .setNegativeButton("Отмена", null)
            .create();
}
```



Диалоговые окна в Android и Java

Здесь добавляется иконка, которая в качестве изображения использует встроенный ресурс android.R.drawable.ic_dialog_alert и устанавливаются две кнопки. Для каждой кнопки можно установить текст и обработчик нажатия. В данном случае для обработчика нажатия передается null, то есть обработчик не установлен.

Теперь добавим в папку **res/layout** новый файл **dialog.xml**:

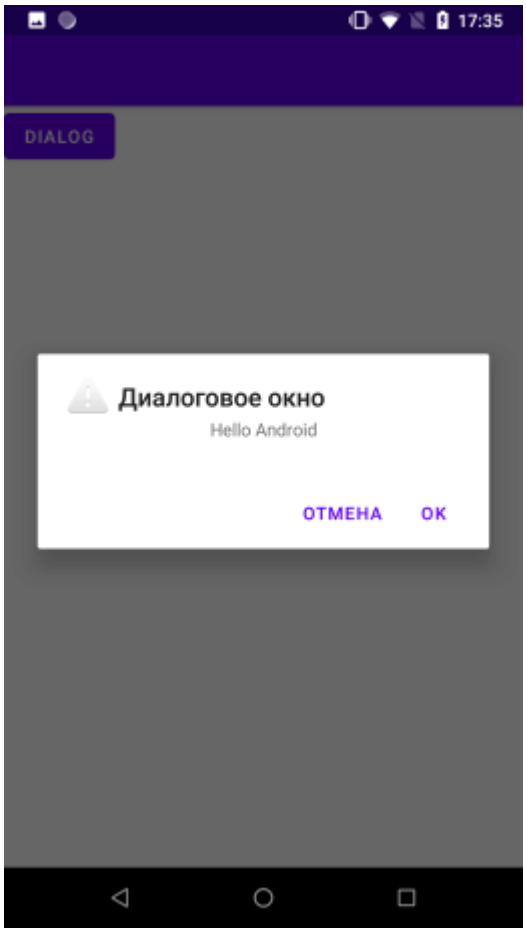
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:gravity="center"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Hello Android"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintBottom_toBottomOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

И изменим создание диалогового окна:

```
public Dialog onCreateDialog(Bundle savedInstanceState) {

    AlertDialog.Builder builder=new AlertDialog.Builder(getActivity());
    return builder
            .setTitle("Диалоговое окно")
```

```
            .setIcon(android.R.drawable.ic_dialog_alert)
            .setView(R.layout.dialog)
            .setPositiveButton("OK", null)
            .setNegativeButton("Отмена", null)
            .create();
}
```

Метод setView() устанавливает в качестве интерфейса окна ранее добавленный ресурс layout **dialog.xml**.



Создание диалоговых окон в Android

При этом, как можно увидеть на скриншоте, кнопки и заголовок с иконкой не входят в разметку.

Передача данных в диалоговое окно

Передача данных в диаговое окно, как и в любой фрагмент, осуществляется с помощью объекта Bundle.

Так, определим в файле activity_main.xml список ListView:

```
<?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">

    <ListView
        android:id="@+id/phonesList"
        android:layout_width="match_parent"
        android:layout_height="match_parent"

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

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

В классе **MainActivity** определим для этого списка данные:

```
package com.example.dialogsapp;
```

```
import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;

import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {

    @Override
    public void onCreate(Bundle savedInstance) {
        super.onCreate(savedInstance);
        setContentView(R.layout.activity_main);

        ListView phonesList = findViewById(R.id.phonesList);
        ArrayList<String> phones = new ArrayList<>();
        phones.add("Google Pixel");
        phones.add("Huawei P9");
        phones.add("LG G5");
        phones.add("Samsung Galaxy S8");

        ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
android.R.layout.simple_list_item_1, phones);
        phonesList.setAdapter(adapter);
```

```
        phonesList.setOnItemClickListener(new AdapterView.OnItemClickListener()
{
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position,
long id) {


                String selectedPhone = adapter.getItem(position);
                CustomDialogFragment dialog = new CustomDialogFragment();
                Bundle args = new Bundle();
                args.putString("phone", selectedPhone);
                dialog.setArguments(args);
                dialog.show(getSupportFragmentManager(), "custom");
            }
        });
    }
}
```

В обработчике нажатия на элемент в списке получаем выбранный элемент и добавляем его в объект Bundle. Далее через метод dialog.setArguments() передаем данные из Bundle во фрагмент.

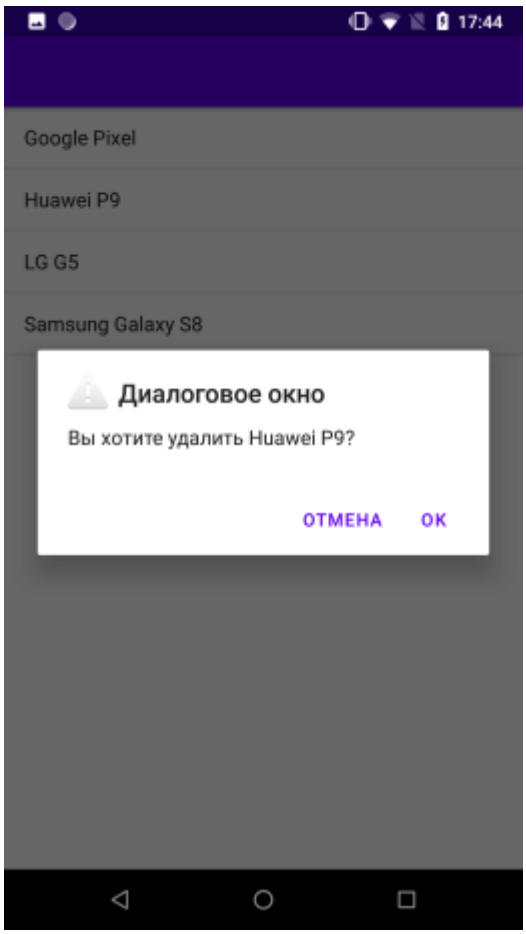
Теперь определим следующий класс фрагмента **CustomDialogFragment**:

```
package com.example.dialogsapp;


import android.app.AlertDialog;
import android.app.Dialog;
import android.os.Bundle;


import androidx.annotation.NonNull;
import androidx.fragment.app.DialogFragment;
```

```
public class CustomDialogFragment extends DialogFragment {

    @NonNull
    public Dialog onCreateDialog(Bundle savedInstanceState) {

        String phone = getArguments().getString("phone");
        AlertDialog.Builder builder=new AlertDialog.Builder(getActivity());
        return builder
                .setTitle("Диалоговое окно")
                .setIcon(android.R.drawable.ic_dialog_alert)
                .setMessage("Вы хотите удалить " + phone + "?")
                .setPositiveButton("OK", null)
                .setNegativeButton("Отмена", null)
                .create();
    }
}
```

С помощью метода getArguments() получаем переданный в MainActivity объект Bundle. И так как была передана строка, то для ее извлечения применяется метод getString().

И при нажатии элемент списка будет передаваться в диалоговое окно:



Передача данных в диалоговое окно AlertDialog и DialogFragment в Android и Java

В данном случае реального удаления не происходит, и в следующей статье рассмотрим, как добавить логику удаления и взаимодействия с Activity

Взаимодействие диалогового окна с Activity

Взаимодействие между Activity и фрагментом производится, как правило, через интерфейс. К примеру, в прошлой теме MainActivity выводила список объектов, и теперь определим удаление из этого списка через диалоговое окно.

Для этого добавим в проект интерфейс **Removable**:

```
package com.example.dialogsapp;

public interface Removable {

    void remove(String name);

}
```

Единственный метод интерфейса remove получает удаляемый объект в виде параметра name.

Теперь реализуем этот интерфейс в коде **MainActivity**:

```
package com.example.dialogsapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;

import java.util.ArrayList;

public class MainActivity extends AppCompatActivity  implements  Removable{

    private ArrayAdapter<String> adapter;
    @Override
    public void onCreate(Bundle savedInstance) {
        super.onCreate(savedInstance);
        setContentView(R.layout.activity_main);

        ListView phonesList = findViewById(R.id.phonesList);
        ArrayList<String> phones = new ArrayList<>();
        phones.add("Google Pixel");
        phones.add("Huawei P9");
        phones.add("LG G5");
        phones.add("Samsung Galaxy S8");
```

```
        adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1,
phones);
        phonesList.setAdapter(adapter);


        phonesList.setOnItemClickListener(new AdapterView.OnItemClickListener()
{
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position,
long id) {


                String selectedPhone = adapter.getItem(position);
                CustomDialogFragment dialog = new CustomDialogFragment();
                Bundle args = new Bundle();
                args.putString("phone", selectedPhone);
                dialog.setArguments(args);
                dialog.show(getSupportFragmentManager(), "custom");
            }
        });
    }
    @Override
    public void remove(String name) {
        adapter.remove(name);
    }
}
```

Метод remove просто удаляет из адаптера переданный элемент.

Файл **activity_main.xml** по прежнему определяет только элемент ListView:

```
<?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">

    <ListView
        android:id="@+id/phonesList"
        android:layout_width="match_parent"
        android:layout_height="match_parent"

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

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

И в конце определим фрагмент **CustomDialogFragment**:

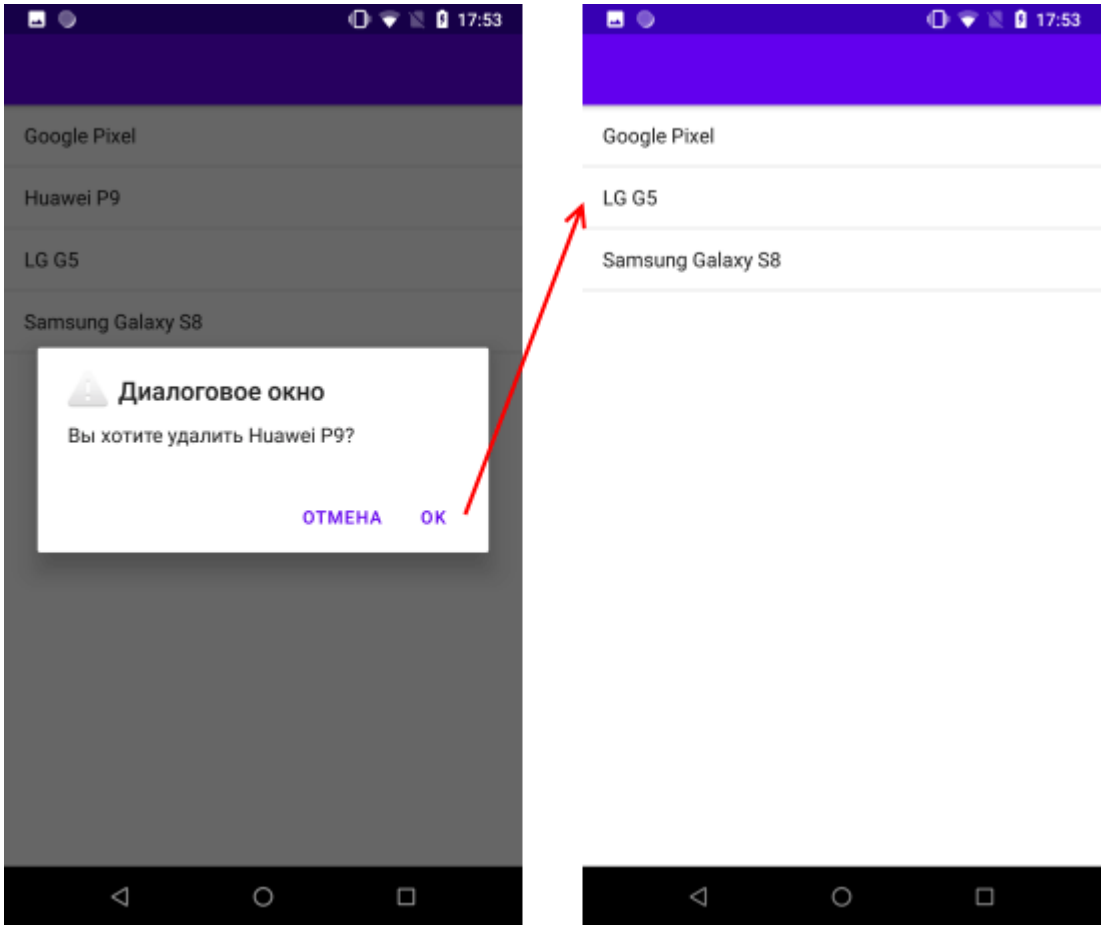
```
package com.example.dialogsapp;

import android.app.AlertDialog;
import android.app.Dialog;
import android.content.Context;
import android.content.DialogInterface;
import android.os.Bundle;

import androidx.annotation.NonNull;
import androidx.fragment.app.DialogFragment;
```

```
public class CustomDialogFragment extends DialogFragment {

    private Removable removable;

    @Override
    public void onAttach(Context context){
        super.onAttach(context);
        removable = (Removable) context;
    }

    @NonNull
    public Dialog onCreateDialog(Bundle savedInstanceState) {

        final String phone = getArguments().getString("phone");
        AlertDialog.Builder builder=new AlertDialog.Builder(getActivity());
        return builder
                .setTitle("Диалоговое окно")
                .setIcon(android.R.drawable.ic_dialog_alert)
                .setMessage("Вы хотите удалить " + phone + "?")
                .setPositiveButton("OK", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        removable.remove(phone);
                    }
                })
                .setNegativeButton("Отмена", null)
                .create();
    }
```

}

Метод **onAttach()** вызывается в начале жизненного цикла фрагмента, и именно здесь мы можем получить контекст фрагмента, в качестве которого выступает класс MainActivity. Так как MainActivity реализует интерфейс Removable, то мы можем преобразовать контекст к данному интерфейсу.

Затем в обработчике кнопки OK вызывается метод remove объекта Removable, который удаляет переданный во фрагмент объект phone.



Взаимодействие с MainActivity в диалоговом окне AlertDialog и DialogFragment в Android и Java