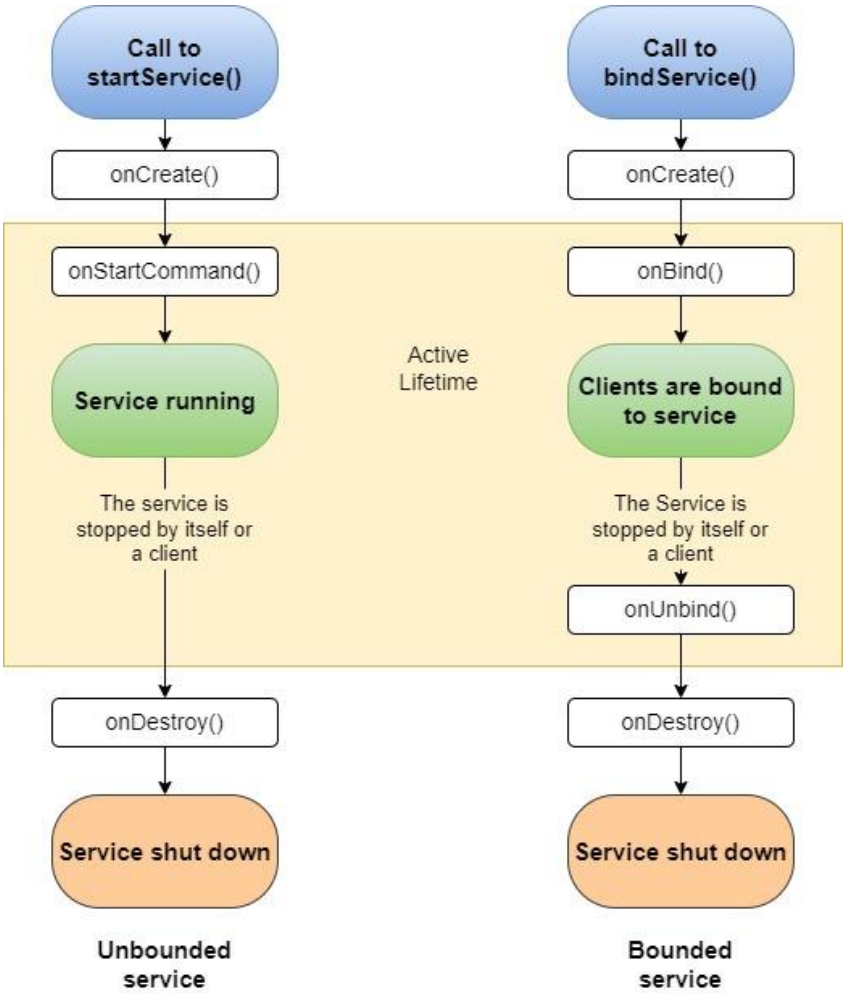
Часть 1. Сервисы

Сервис в Android — это компонент приложения, предназначенный для выполнения длительных или ресурсоемких операций в фоновом режиме без предоставления пользовательского интерфейса. Сервисы могут работать в фоне даже тогда, когда пользователь не взаимодействует с приложением, что делает их идеальными для таких задач, как воспроизведение музыки, выполнение сетевых операций, выполнение вычислений и мониторинг данных.

Все сервисы наследуются от класса **Service** и по аналогии с активностью, сервис имеет свой жизненный цикл и методы:

1. onCreate().
2. onStartCommand(Intent intent, int flags, int startId).
3. onBind(Intent intent).
4. onUnbind(Intent intent).
5. onDestroy().



Метод **onCreate()** вызывается при создании сервиса. Это первый вызов, который получает сервис, и он используется для однократной инициализации, такой как создание ресурсов, которые будут использоваться в течение всего времени существования сервиса. Метод **onCreate()** вызывается только один раз перед вызовом onStartCommand() или onBind().

Метод **onStartCommand()** вызывается каждый раз, когда компонент (например, активность) запрашивает запуск сервиса через **startService().** В этом методе сервис может выполнять любые операции, включая запуск потока для выполнения сложной задачи в фоне. Метод возвращает константу, указывающую, как система должна вести себя, если сервис уничтожается до того, как он завершит выполнение своей работы.

Метод **onBind()** вызывается, когда другой компонент хочет привязаться к сервису через **bindService().** Если сервис не предоставляет интерфейс для клиентов, то он должен возвращать null.

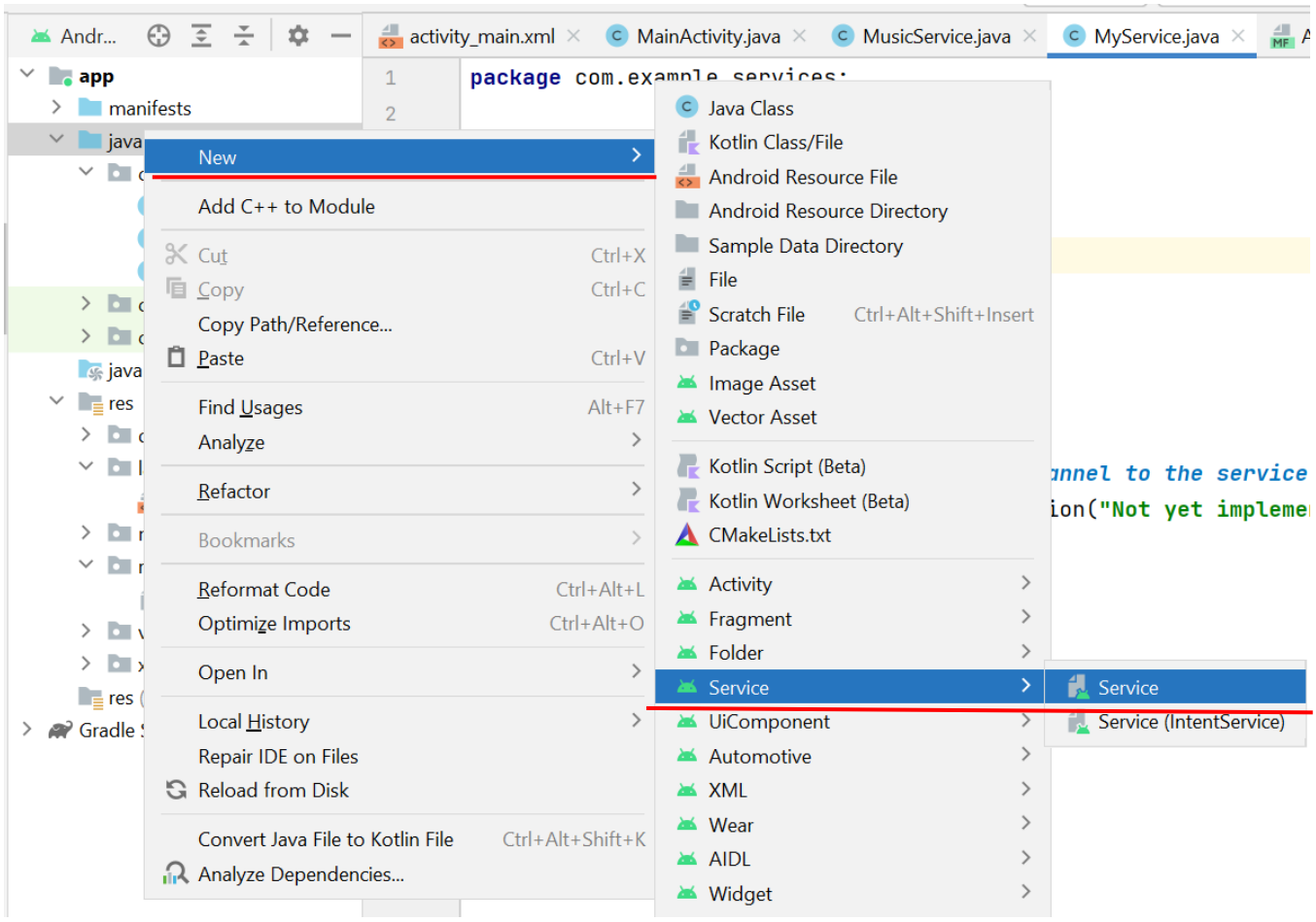
Метод **onUnbind()** вызывается, когда все клиенты отсоединились от определенного интерфейса сервиса. После этого вызова, если необходимо, сервис может остановить себя через **stopSelf().**

Метод **onDestroy()** вызывается, когда сервис больше не используется и собирается быть уничтоженным. Это последний вызов, который получает сервис, и он используется для освобождения ресурсов, таких как потоки, зарегистрированные приемники, обработчики и т.д.

В качестве примера, создадим сервис, который будет воспроизводить музыку. Предварительно загрузим медиафайл формата mp3.

Чтобы это сделать необходимо создать в папке res создать папку **raw**. Именно в этой папке хранятся различные файлы, которые сохраняются в исходном виде. После этого нужно загрузить медиафайл в папку **res/raw** таким же способом, как добавляем изображения.

После того, как файл добавлен нужно создать новый класс сервиса. Это можно сделать, нажав правой кнопкой мыши на **"java"→ "New" → "Service" → "Service".**



Теперь нужно прописать логику работы сервиса.

```
    public class MusicService extends Service {
        private static final String TAG = "MusicService";
        private MediaPlayer mediaPlayer;

        @Override
        public void onCreate() {
            super.onCreate();
            // Инициализация медиаплеера
            mediaPlayer = MediaPlayer.create(this, R.raw.music);
            mediaPlayer.setLooping(true); // Зацикливание
воспроизведения
            mediaPlayer.setVolume(100, 100);
        }

        @Override
        public int onStartCommand(Intent intent, int flags, int
startId) {
            if (!mediaPlayer.isPlaying()) {
```

```
                mediaPlayer.start();
                Log.d(TAG, "Музыка начала играть");
            }
            return START_STICKY;
        }

        @Override
        public void onDestroy() {
            super.onDestroy();
            if (mediaPlayer.isPlaying()) {
                mediaPlayer.stop();
                mediaPlayer.release();
                Log.d(TAG, "Музыка остановлена и ресурсы
освобождены");
            }
        }

        @Override
        public IBinder onBind(Intent intent) {
            return null; // Для сервисов без привязки возвращаем null
        }
    }
```

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.

Далее, объявим сервис в файле **AndroidManifest.xml**. При создании сервиса изменения в файле манифеста произойдут автоматически, но если этого не произошло, то необходимо прописать вручную объявление сервиса.

```
<application
    ...>
    <service android:name=".MusicService" />
    ...
</application>
```

```
<application
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@mipmap/ic_launcher"
    android:label="Services"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.Services"
    tools:targetApi="31" >
    <service
        android:name=".MusicService"
        android:enabled="true"
        android:exported="true" />
    <activity
        android:name=".MainActivity"
```

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.

Теперь можно управлять сервисом (запускать и останавливать воспроизведение) из активности.

```
public class MainActivity extends AppCompatActivity {

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

        // Запуск сервиса для воспроизведения музыки
        Intent startIntent = new Intent(this, MusicService.class);
        startService(startIntent);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        // Остановка сервиса и музыки при уничтожении активности
        Intent stopIntent = new Intent(this, MusicService.class);
        stopService(stopIntent);
    }
}
```

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 в классе Activity и принимает объект Intent. Он останавливает работу сервиса, вызывая его метод **onDestroy().**

Теперь если запустить созданный проект, то начнет играть музыка, при этом на экране ничего не будет отображаться, только если мы сами не зададим.

Часть 2. Диалоговые окна

Диалоговые окна в Android представляют собой небольшие окна, которые отображаются поверх основного содержимого приложения для передачи важного сообщения пользователю или запроса действия от пользователя. Эти окна могут использоваться для подтверждения действий, информирования о событиях, ввода данных и других задач, требующих внимания пользователя.

Среди основных типов диалоговых окон можно выделить:

1. AlertDialog: Используется для отображения предупреждений и предложения пользователю совершить выбор. Может включать кнопки для обработки действий пользователя, такие как «Да», «Нет» или «Отмена».
2. DatePickerDialog и TimePickerDialog: Позволяют пользователю выбрать дату или время соответственно.
3. Custom Dialog: Позволяет создавать диалоговые окна с пользовательским макетом, что может включать любые элементы управления.

В создаваемых диалоговых окнах можно задавать следующие элементы:

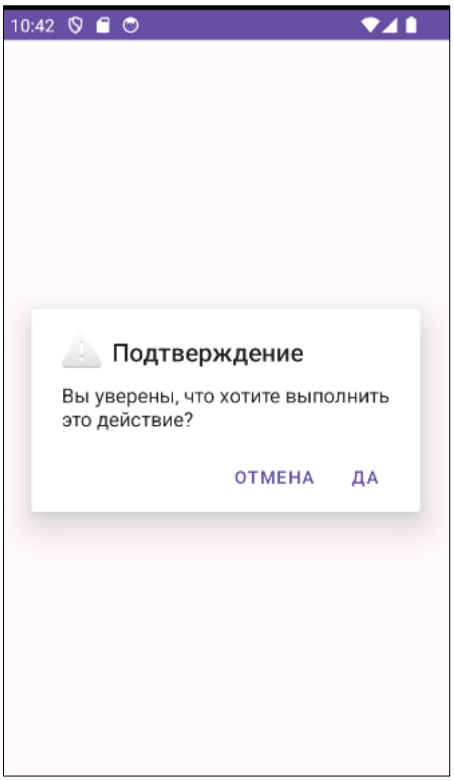
- заголовок
- текстовое сообщение
- кнопки: от одной до трёх
- список
- флажки
- переключатели

Разберем созданием каждого типа диалоговых окон.

Начнём с AlertDialog с простого примера – покажем на экране диалоговое окно с вопросом и парой кнопок. Для этого в файле MainActivity пропишем следующий код.

```
    // Создание строителя диалоговых окон
    AlertDialog.Builder builder = new
AlertDialog.Builder(MainActivity.this);

    // Установка заголовка и сообщения диалогового окна
    builder.setTitle("Подтверждение");
    builder.setMessage("Вы уверены, что хотите выполнить это
действие?");
    builder.setIcon(android.R.drawable.ic_dialog_alert);

    // Установка кнопки "Да" и ее обработчика
    builder.setPositiveButton("Да", new
DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            // Обработка подтверждения
        }
    });

    // Установка кнопки "Отмена" и ее обработчика
    builder.setNegativeButton("Отмена", new
DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            // Обработка отмены действия
            dialog.dismiss();
        }
    });

    // Создание и отображение AlertDialog
    AlertDialog dialog = builder.create();
    dialog.show();
```

Таким образом будет создано диалоговое окно с кнопками «Да» и «Отмена», которое может обрабатывать действия пользователя.



Класс **AlertDialog.Builder** с помощью своих методов 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

Аналогичным образом создаются диалоговые окна с выбором времени или даты. Создание диалогового окна с выбором времени:

```
    // Создание и отображение TimePickerDialog
            TimePickerDialog timePickerDialog = new TimePickerDialog(
```

```
                    this,
                    new TimePickerDialog.OnTimeSetListener() {
                        @Override
                        public void onTimeSet(TimePicker view, int
hourOfDay, int minute) {
                            // Обработка выбранного времени
                            // Пример: установка времени в TextView
                            // textView.setText(hourOfDay + ":" +
minute);
                        }
                    }, hour, minute, true); // Использование 24-
часового формата

            timePickerDialog.show();
```

```
1 usage
private int hour;
1 usage
private int minute;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // Создание и отображение TimePickerDialog
    TimePickerDialog timePickerDialog = new TimePickerDialog(
            context: this,
            new TimePickerDialog.OnTimeSetListener() {
                @Override
                public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
                    // Обработка выбранного времени
                    // Пример: установка времени в TextView
                    // textView.setText(hourOfDay + ":" + minute);
                }
            }, hour, minute, is24HourView: true); // Использование 24-часового формата
    timePickerDialog.show();
}
```

Не забыть создать переменные

Создание диалогового окна с выбором даты:

```
    // Создание обработчика выбора даты
    DatePickerDialog.OnDateSetListener dateSetListener = new
DatePickerDialog.OnDateSetListener() {
        @Override
        public void onDateSet(DatePicker view, int year, int month,
int dayOfMonth) {
            // Обработка выбора даты
        }
    };

    // Создание и отображение DatePickerDialog
    DatePickerDialog datePickerDialog = new DatePickerDialog(
            MainActivity.this,
            dateSetListener,
            year, // текущий год
            month, // текущий месяц
            day); // текущий день
    datePickerDialog.show();
```

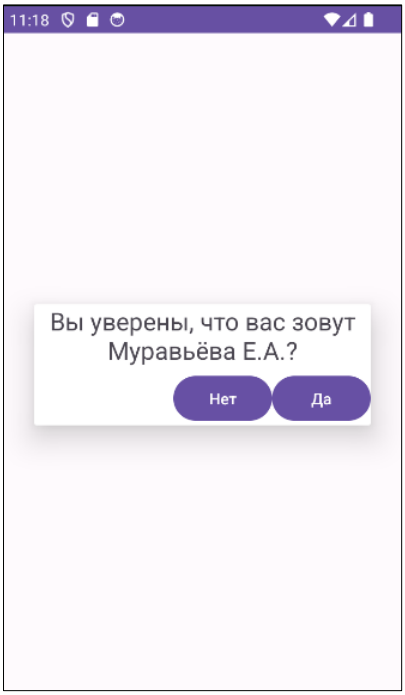
```
private int year;
1 usage
private int month;
1 usage
private int day;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // Создание обработчика выбора даты
    DatePickerDialog.OnDateSetListener dateSetListener = new DatePickerDialog.OnDateSetListener() {
        @Override
        public void onDateSet(DatePicker view, int year, int month, int dayOfMonth) {
            // Обработка выбора даты
        }
    };
    // Создание и отображение DatePickerDialog
    DatePickerDialog datePickerDialog = new DatePickerDialog(
            context: MainActivity.this,
            dateSetListener,
            year, // текущий год
            month, // текущий месяц
            day); // текущий день
    datePickerDialog.show();
}
```

Не забываем создать переменные

Создание пользовательского диалогового окна отличатся установкой макета того, как будет выглядеть окно и какие элементы пользовательского интерфейса будет иметь. Для этого необходимо создать отдельный макет, например custom_dialog и наполнить его элементами пользовательского интерфейса. Например, добавим текстовой поле с вопросом и две кнопки для выбора.

```
// Создание диалога
Dialog dialog = new Dialog(MyActivity.this);

// Установка макета для диалогового окна
dialog.setContentView(R.layout.custom_dialog);

// Настройка элементов в макете
Button button = dialog.findViewById(R.id.button);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Обработка нажатия на кнопку
    }
});

// Отображение диалогового окна
dialog.show();
```

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Создание диалога
        Dialog dialog = new Dialog( context: MainActivity.this);
        // Установка макета для диалогового окна
        dialog.setContentView(R.layout.custom_dialog);
        // Настройка элементов в макете
        Button button = dialog.findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Обработка нажатия на кнопку
            }
        });
        // Отображение диалогового окна
        dialog.show();
    }
}
```



Кроме того, что диалоговое окно можно вызвать напрямую в MainActivity, его можно выделить в отдельный класс фрагмента **DialogFragment**.

Использование фрагментов для диалоговых окон в силу своей архитектуры является удобным вариантом в приложениях, который лучше справляется с поворотами устройства, нажатием кнопки "Назад", лучше подходит под разные размеры экранов и т.д.

Для создания диалога следует наследоваться от класса **DialogFragment**.

Вначале добавляем в проект новый класс фрагмента, который назовем MyDialogFragmen.

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, также как и в MainActivity.

```
public class MyDialogFragment extends DialogFragment {
    3 usages
    @NonNull
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {

        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        // Установка заголовка,сообщения, иконки диалогового окна
        builder.setTitle("Подтверждение");
        builder.setMessage("Вы уверены, что хотите выполнить это действие?");
        builder.setIcon(android.R.drawable.ic_dialog_alert);
        builder.setNegativeButton( text: "Отмена",  listener: null);
        builder.setPositiveButton( text: "Ок",  listener: null);
        return builder.create();
    }
}
```

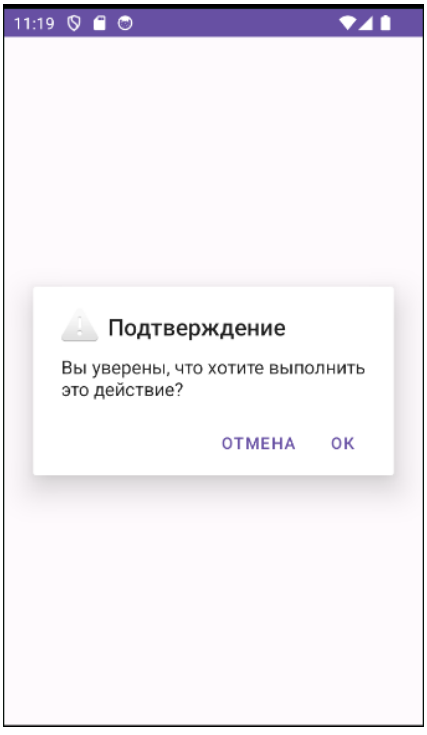
В данном случае для обработчика нажатия передается null, то есть обработчик не установлен.

Теперь диалоговое окно можно вызвать в классе MainActivity.

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

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

    MyDialogFragment dialogFragment = new MyDialogFragment();
    dialogFragment.show(getSupportFragmentManager(),  tag: "Проверка");
}
```

Результат запуска будет таким же, как и ранее.



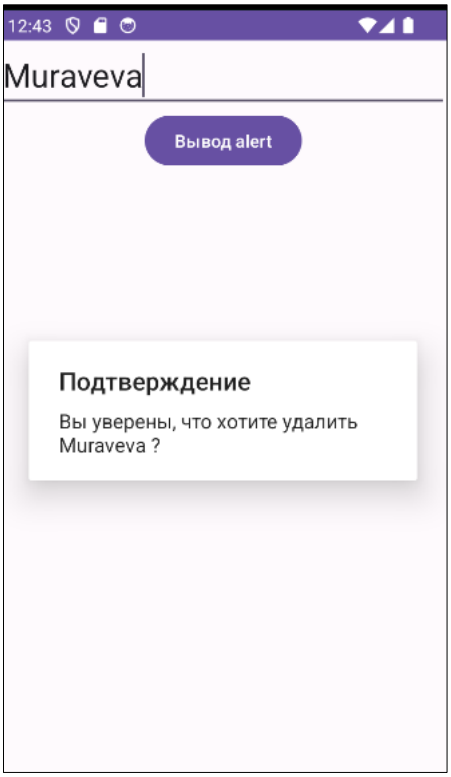
Что делать, если в диалоговое окно необходимо передать данные для уточнения информации? В таком случае в сообщение необходимо передавать текстовое значение.

Например, создадим поле для ввода фамилии и кнопку для обработки нажатия. После нажатия на кнопку, у нас будет выводиться предупреждение.

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

    Button alertButton = findViewById(R.id.buttonAlert);
    AlertDialog.Builder builder = new AlertDialog.Builder( context: MainActivity.this);
    alertButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            EditText nameText = findViewById(R.id.textName);
            String name = nameText.getText().toString();
            builder.setTitle("Подтверждение");
            builder.setMessage("Вы уверены, что хотите удалить " + name + " ?");
            builder.create().show();
        }
    });
}
```

Получает введенный текст

Передаем текст в Alert



Задание

1. Создать и запустить сервис с любым функционал (не с логами).
2. Создать диалоговые окна каждого вида (AlertDialog, DatePickerDialog, TimePickerDialog, Custom Dialog). Реализовать передачу данных с окон во фрагменты или активности. Например, устанавливать в текстовое поле полученные значения времени и даты, реализовать переход на другую activity или изменить фрагменты при нажатии на кнопки диалогового окна.

Источники

1) https://developer.android.com/reference/android/app/Service
2) https://www.geeksforgeeks.org/services-in-android-with-example/
3) https://developer.alexanderklimov.ru/android/theory/services-theory.php?ysclid=ltvlwjthxu857666576
4) https://developer.alexanderklimov.ru/android/dialogfragment_alertdialog.php?ysclid=ltvlxt5dm1933741362
5) https://metanit.com/java/android/18.2.php?ysclid=ltvlxu0u8p822972346
6) https://www.geeksforgeeks.org/datepickerdialog-in-android/