

## ЗАНЯТИЕ 2.16

### ПРАКТИЧЕСКОЕ ЗАНЯТИЕ

**Тема:** Отображение уведомлений. Компоненты Android-приложений: Services и Broadcast Receivers

#### Упражнение №1

В данном упражнении будет изучена работа с файловой системой Android (внутренним хранилищем).

Создайте новое Android-приложение для телефона с использованием шаблона Empty Activity (или Empty View Activity, в зависимости от версии Android Studio).

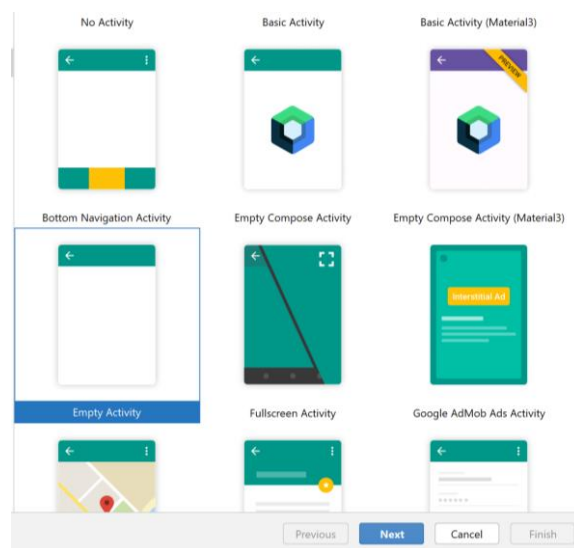


Рисунок 1

Установите минимальную версию API 21. Язык разработки – Java (не Kotlin). Имя приложения – ServiceApplication.

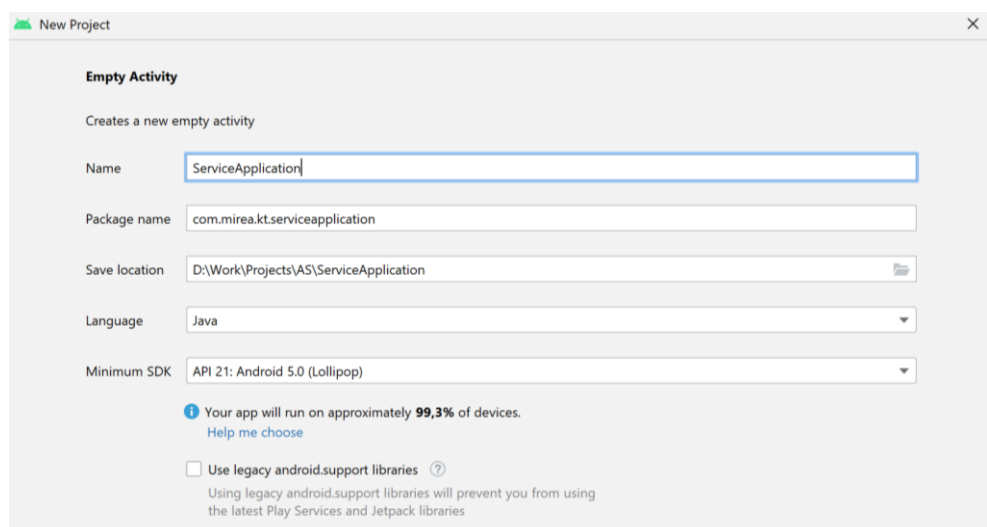


Рисунок 2

Дождитесь создания приложения и корректного обновления плагина Gradle. Перейдите в файл макета (в режим Code) **activity\_main.xml**. Замените контейнер **ConstraintLayout** на **RelativeLayout**. Добавьте внутрь контейнера четыре кнопки. Обратите внимание, что значения текста и цвета указываются из файлов ресурсов **strings.xml** и **colors.xml** соответственно:

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <Button
        android:id="@+id/btnShow"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:text="@string/show_notification" />
    <Button
        android:id="@+id/btnCancel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/btnShow"
        android:layout_centerInParent="true"
        android:text="@string/cancel_notification" />
    <Button
        android:id="@+id/btnStart"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/btnCancel"
        android:layout_centerInParent="true"
        android:text="@string/start_service" />
    <Button
        android:id="@+id/btnStop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/stop_service"
        android:layout_centerInParent="true"
        android:layout_below="@id/btnStart"/>
</RelativeLayout>
```

Рисунок 3

```

<string name="app_name">ServiceApplication</string>
<string name="show_notification">Показать уведомление</string>
<string name="start_service">Запуск сервиса</string>
<string name="stop_service">Остановка сервиса</string>
<string name="cancel_notification">Убрать уведомление</string>
</resources>

```

Рисунок 4 – Ссылки на строки в файле strings.xml

Макет в итоге должен выглядеть примерно следующим образом:

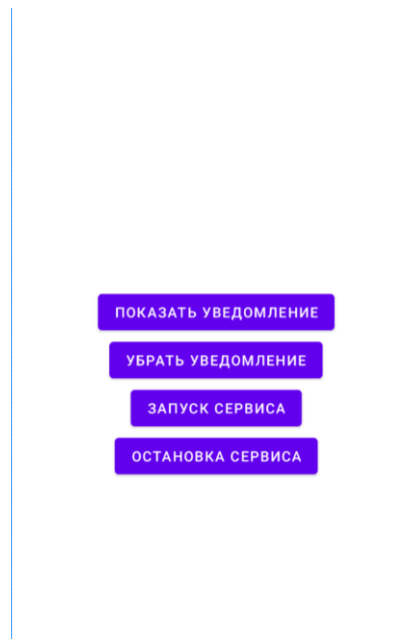


Рисунок 5

В приложении есть главный класс – Application. Это стандартный класс из Android SDK, и он используется неявно (вы с ним не работаете напрямую). Но можно создать собственный класс Application, с которого будет стартовать приложение. Как правило, он используется для хранения глобальных данных и состояний, доступных во всем приложении. Класс Application наследуется от Context, что дает ему доступ к ресурсам и сервисам Android.

Создайте новый класс. Назовите его MyCustomApp (или как-то иначе). Сделайте данный класс наследником класса Application.

```

activity_main_old.xml × activity_main.xml × strings.xml × MainActivity.java × MyCustomApp
package com.mirea.kt.serviceapplication;

import android.app.Application;

public class MyCustomApp extends Application {

}

```

Рисунок 6

Для того, чтобы работа приложения начиналась именно с этого класса Application, необходимо добавить запись об этом в файле манифеста:

```
<application
    android:name=".MyCustomApp"
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@mipmap/ic_launcher"
```

Рисунок 7

Вернитесь в класс MyCustomApp и переопределите в нем метод onCreate родительского класса (используйте средства генерации кода). Также создайте публичную тестовую константу, которая будет хранить тэг логирования, и к которой можно будет обращаться из любой точки приложения.

```
public class MyCustomApp extends Application {

    public static final String LOG_TAG = "my_app_tag";

    @Override
    public void onCreate() {
        super.onCreate();
        Log.i(LOG_TAG, msg: "Application created!");
    }
}
```

Рисунок 8

Перейдите в класс вашей activity и реализуйте обработчики нажатия для кнопок «Показать уведомление» и «Убрать уведомление». В качестве слушателя используйте саму activity. По нажатию на первую кнопку будет показываться уведомление, которое вы создадите.

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btnShowNotif = findViewById(R.id.btnShow);
        Button btnCancelNotif = findViewById(R.id.btnCancel);
        btnShowNotif.setOnClickListener(this);
        btnCancelNotif.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        if(v.getId() == R.id.btnShow){

        }else if(v.getId() == R.id.btnCancel){

        }
    }
}
```

Рисунок 9

Создайте новый класс – NotificationHelper.

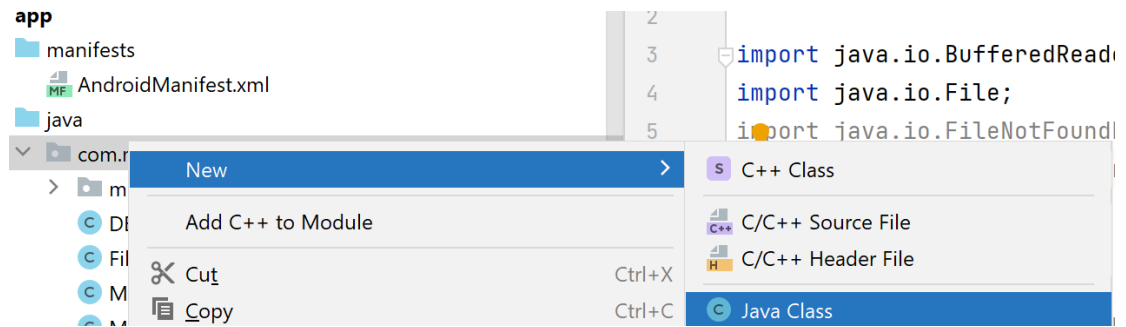


Рисунок 10

В данном классе будут созданы методы для работы с уведомлениями, чтобы не загружать лишним кодом activity. Старайтесь придерживаться такой практики. Скопируйте код класса в свой проект.

```
public class NotificationHelper {

    private final NotificationManagerCompat notificationManager;
    private static final String CHANNEL_ID = "my_channel_id";
    private final Context context;

    public NotificationHelper(Context context) {
        this.context = context;
        this.notificationManager = NotificationManagerCompat.from(context);
    }

    public void showNotification(String title, String message, int notificationId) {
        Notification notification = createNotification(title, message,
notificationId);
        notificationManager.notify(notificationId, notification);
    }

    public Notification createNotification(String title, String message, int
notificationId) {
        if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.O) {
            NotificationChannel channel = new NotificationChannel(CHANNEL_ID, "My
channel", NotificationManager.IMPORTANCE_HIGH);
            channel.setDescription("My channel description");
            channel.enableLights(true);
            channel.setLightColor(Color.RED);
            channel.enableVibration(false);
            notificationManager.createNotificationChannel(channel);
        }
        NotificationCompat.Builder builder =
            new NotificationCompat.Builder(context, CHANNEL_ID)
                .setSmallIcon(R.mipmap.ic_launcher)
                .setContentTitle(title)
                .setContentText(message)
                .setPriority(NotificationCompat.PRIORITY_HIGH);
        Notification notification = builder.build();
        return notification;
    }

    public void cancelNotification(int notificationId) {
        if(notificationManager != null){
            notificationManager.cancel(notificationId);
        }
    }

}
```

Конструктор класса принимает Context для того, чтобы внутри класса можно было получить экземпляр класса NotificationManagerCompat, необходимый для работы с уведомлениями. Также Context используется при формировании внешнего вида уведомления.

Созданный метод **showNotification** будет отвечать за показ уведомления, созданного методом **createNotification** по входным параметрам. В качестве входных параметров метод принимает заголовок уведомления, содержимое и идентификатор уведомления. Идентификатор нужен, чтобы можно было отличать уведомления между собой (ведь их может быть много). При необходимости можно изменить список аргументов этого метода, чтобы сделать настройку уведомления еще более гибкой (иконка, цвет светового индикатора и пр.). В методе **createNotification** также происходит создание канала уведомлений (но только для устройств с Android 8 и выше). Каналы позволяют определять разные типы уведомлений в одном приложении, имея возможность настраивать несколько уровней приоритета и разные методы уведомления. Таким образом, пользователь может лучше контролировать информацию, которую он хочет получать. Разработчик приложения создает канал и указывает его ID при создании уведомлений. Пользователь в системных настройках приложения видит этот канал и может настроить его: важность, звук, вибрацию и пр. В итоге все уведомления, которые принадлежат этому каналу, будут отображаться с этими настройками. Т.е. создавая канал, разработчик дает пользователю возможность настроить поведение определенной группы уведомлений.

В рамках тренировочного упражнения будет создан один канал (в приложении можно создать сколько угодно каналов, если это требуется).

Созданный метод **cancelNotification** будет отвечать за отмену показа уведомления по его идентификатору.

При этом, если приложение работает на устройстве с Android 13 и выше необходимо сначала реализовать запрос на показ уведомлений.

Перейдите в файл манифеста и добавьте permission для показа уведомлений и на разрешение работы в фоне:



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

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

Рисунок 11

Вернитесь в класс вашей activity и реализуйте код запроса разрешения на показ уведомлений, если у пользователя устройство с Android 13. Если разрешение получено или у пользователя более старая версия операционной

системы, то вызовете метод показа уведомлений (заголовок, текст и идентификатор уведомления передайте в аргументах любые).

Скопируйте код ниже в класс вашей activity. Изучите как он работает. **При необходимости задайте вопрос преподавателю (или преподаватель задаст вопрос вам).**

```
@Override
public void onClick(View v) {
    if(v.getId() == R.id.btnShow){
        if(askNotificationPermission()){
            NotificationHelper notificationHelper = new
NotificationHelper(this);
            notificationHelper.showNotification("Внимание",
                "Это текст тестового уведомления",
                123);
        }
    }else if(v.getId() == R.id.btnCancel){
        NotificationHelper notificationHelper = new
NotificationHelper(this);
        notificationHelper.cancelNotification(123);
    }
}

// сюда возвращается результат того, что нажал пользователь в диалоге
private final ActivityResultLauncher<String> requestPermissionLauncher =
    registerForActivityResult(new
ActivityResultContracts.RequestPermission(), isGranted -> {
        if (isGranted) {
            Log.d(LOG_TAG, "Permission granted!");
        } else {
            Log.d(LOG_TAG, "Permission not granted");
            Toast.makeText(this, "Приложение может работать
некорректно!", Toast.LENGTH_LONG).show();
        }
    });

private boolean askNotificationPermission() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
        //проверяем есть ли у приложения разрешение на показ уведомлений
        if (ContextCompat.checkSelfPermission(this,
"android.permission.POST_NOTIFICATIONS") ==
PackageManager.PERMISSION_GRANTED) {
            return true;
        }
        //проверяем отказался ли пользователь от показа уведомлений
    } else if
(shouldShowRequestPermissionRationale("android.permission.POST_NOTIFICATIONS"
)) {
        Toast.makeText(this, "Вы не предоставили приложению разрешение для
получения уведомлений!", Toast.LENGTH_LONG).show();
        return false;
    } else {
        //спрашиваем у пользователя разрешение на показ уведомлений
        requestPermissionLauncher.launch("android.permission.POST_NOTIFICATIONS");
        return false;
    }
}
return true;
}
```



Запустите приложение. Убедитесь, что уведомление отображается и отменяется при нажатии на соответствующие кнопки. Самостоятельно сохраните используемые строки в файл ресурсов strings.xml.

## Упражнение №2

Продолжайте работу в текущем приложении.

В этом упражнении будут изучены особенности работы сервиса. В рамках упражнения будет реализован фоновый сервис, который с некоторой периодичностью будет проверять заблокировано на данный момент устройство или нет.

Сначала необходимо убедиться, что блокировка экрана поддерживается вашим устройством. Если вы работаете с эмулятором Android Studio, то ее необходимо включить. Перейдите в приложение настройки (Settings), далее Security->Screen Lock. Выберите Swipe.

Создайте класс сервиса. Назовите его как считаете нужным. В данном примере он будет называться LockMonitorService.

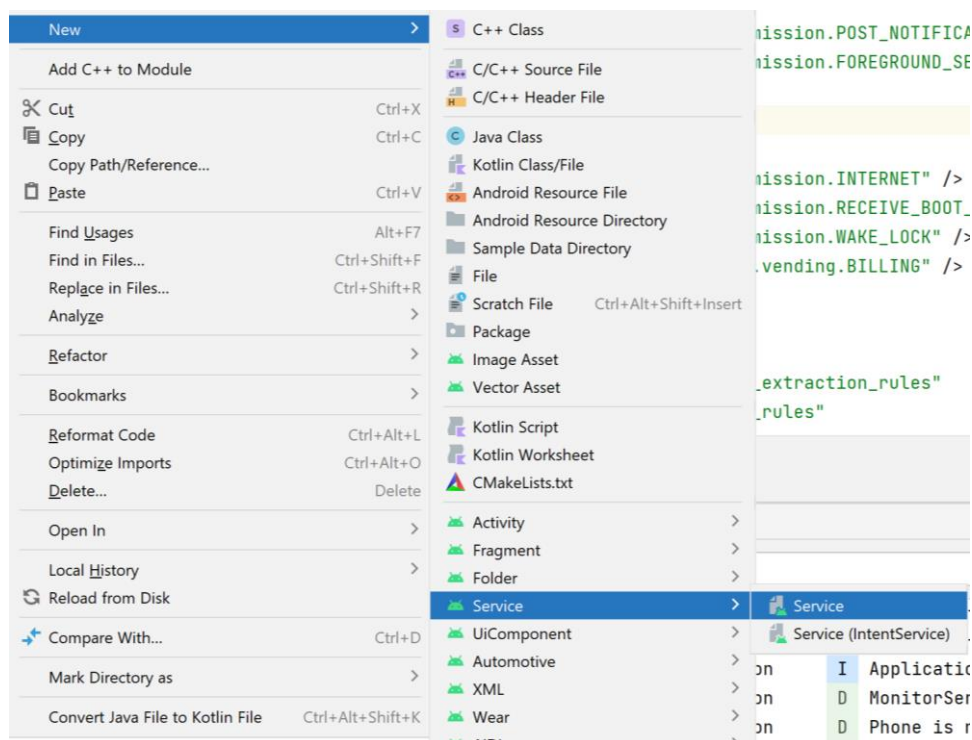


Рисунок 12

Перейдите в файл манифеста AndroidManifest.xml и убедитесь, что информация о созданном сервисе там отображается.

Вернитесь в класс вашей activity и зарегистрируйте слушателей для двух оставшихся кнопок. Также можно закомментировать код внутри метода OnClick. По нажатию на кнопку «Запуск сервиса» сервис будет стартовать и выполнять фоновую работу. Следует обратить внимание, что методы запуска сервиса отличаются для версий Android (в Android 8 появились



дополнительные ограничения по работе фоновых сервисов, см. лекцию №2.13).

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btnShowNotif = findViewById(R.id.btnShow);
        Button btnCancelNotif = findViewById(R.id.btnCancel);
        Button btnStart = findViewById(R.id.btnStart);
        Button btnStop = findViewById(R.id.btnStop);
        btnStart.setOnClickListener(this);
        btnStop.setOnClickListener(this);
        btnShowNotif.setOnClickListener(this);
        btnCancelNotif.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        Intent servIntent = new Intent( packageContext: this, LockMonitorService.class);
        if(v.getId() == R.id.btnStart){
            if(Build.VERSION.SDK_INT < Build.VERSION_CODES.O){
                startService(servIntent);
            }else{
                startForegroundService(servIntent);
            }
        }else if(v.getId() == R.id.btnStop){
            stopService(servIntent);
        }
    }
}
```

Рисунок 13

Перейдите в класс сервиса. Переопределите метод **onStartCommand**, который выполняет основную работу сервиса. Для устройств с Android 8 необходимо отобразить уведомление о работе фонового сервиса, выполнив после старта сервиса метод **startForeground**. Метод **startForeground** в качестве аргументов принимает на вход ваш идентификатор уведомления и сам объект уведомления. Объект уведомления можно получить методом **createNotification**, который ранее был создан вами в вашем вспомогательном классе **NotificationHelper**.

```

public class LockMonitorService extends Service {

    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.d(LOG_TAG, msg: "MonitorService started");
        NotificationHelper notificationHelper = new NotificationHelper( context: this);
        Notification n = notificationHelper.createNotification( title: "Мониторинг", message: "Сервис наблюдения запущен", notificationId: 12345);
        startForeground( id: 12345, n);
        return START_STICKY;
    }

    @Override
    public void onDestroy() {
        Log.d(LOG_TAG, msg: "MonitorService destroyed");
        super.onDestroy();
    }
}

```

Рисунок 14

Нужно создать метод, с помощью которого получится детектировать состояние экрана устройства. Скопируйте в класс сервиса метод **checkPhoneScreenLocked**. Информацию о блокировке можно получить с помощью keyguard-сервиса (не обязательно понимать, как этот метод работает).

```

private void checkPhoneScreenLocked(){
    KeyguardManager km = (KeyguardManager)
getApplicationContext().getSystemService(Context.KEYGUARD_SERVICE);
    if(km.isKeyguardLocked()) {
        Log.d(LOG_TAG,"Phone is locked");
    } else {
        Log.d(LOG_TAG,"Phone is not locked");
    }
}

```

Метод выводит в logcat статус блокировки экрана.

Чтобы реализовать периодическое выполнение данного метода можно использовать Handler (см. лекцию №2.15). Создайте в классе сервиса объект Handler и «свяжите» его с «лупером» главного потока.

```

public class LockMonitorService extends Service {

    private Handler handler = new Handler(Looper.getMainLooper());

```

Рисунок 15

В итоге реализация выполнения проверки статуса блокировки с периодичностью 2 секунды будет выглядеть следующим образом (не забудьте в onDestroy очистить все отправленные и запланированные хэндлером сообщения):

```

public class LockMonitorService extends Service {

    private Handler handler = new Handler(Looper.getMainLooper());

    class MonitoringRunnable implements Runnable{
        @Override
        public void run() {
            checkPhoneScreenLocked();
            handler.postDelayed( r: this, delayMillis: 2000);
        }
    }

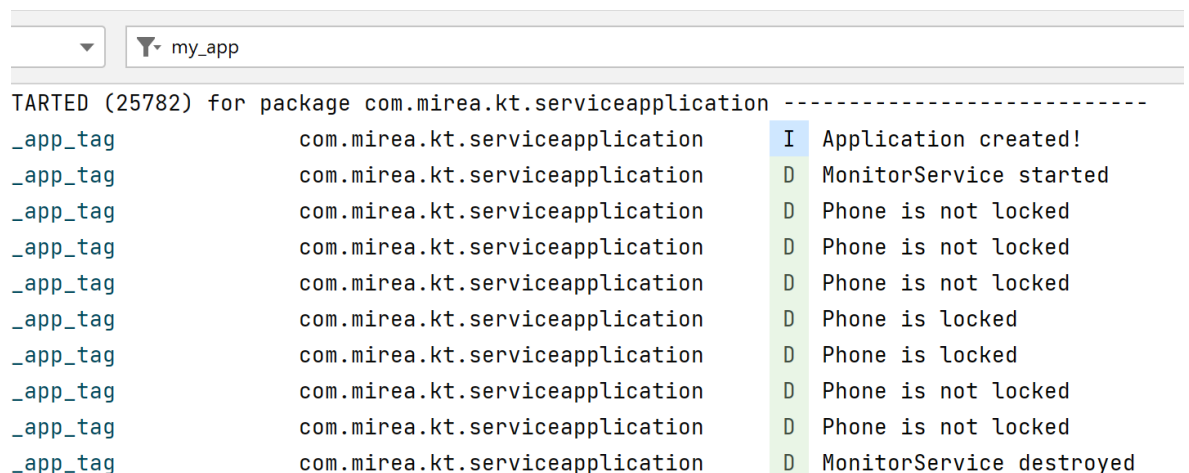
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.d(LOG_TAG, msg: "MonitorService started");
        NotificationHelper notificationHelper = new NotificationHelper( context: this);
        Notification n = notificationHelper.createNotification( title: "Мониторинг",
            message: "Сервис наблюдения запущен",
            notificationId: 12345);
        startForeground( id: 12345, n);
        handler.post(new MonitoringRunnable());
        return START_STICKY;
    }

    @Override
    public void onDestroy() {
        Log.d(LOG_TAG, msg: "MonitorService destroyed");
        // обнуляем очередь сообщений от хэндлера
        handler.removeCallbacksAndMessages( token: null);
        super.onDestroy();
    }
}

```

Рисунок 16

Запустите приложение, нажмите на кнопку запуска сервиса. Сверните приложение и заблокируйте устройство. Затем разблокируйте устройство, вернитесь в приложение и остановите сервис. Убедитесь, что в логах отображаются корректные значения.



```

TARTED (25782) for package com.mirea.kt.serviceapplication -----
_app_tag      com.mirea.kt.serviceapplication  I  Application created!
_app_tag      com.mirea.kt.serviceapplication  D  MonitorService started
_app_tag      com.mirea.kt.serviceapplication  D  Phone is not locked
_app_tag      com.mirea.kt.serviceapplication  D  Phone is not locked
_app_tag      com.mirea.kt.serviceapplication  D  Phone is not locked
_app_tag      com.mirea.kt.serviceapplication  D  Phone is locked
_app_tag      com.mirea.kt.serviceapplication  D  Phone is locked
_app_tag      com.mirea.kt.serviceapplication  D  Phone is not locked
_app_tag      com.mirea.kt.serviceapplication  D  Phone is not locked
_app_tag      com.mirea.kt.serviceapplication  D  MonitorService destroyed

```

Рисунок 17

## Упражнение №3

Продолжайте работу в текущем приложении.

В этом упражнении будет реализован приемник широковещательных сообщений – `BroadcastReceiver`. Создайте в проекте класс-приемника:

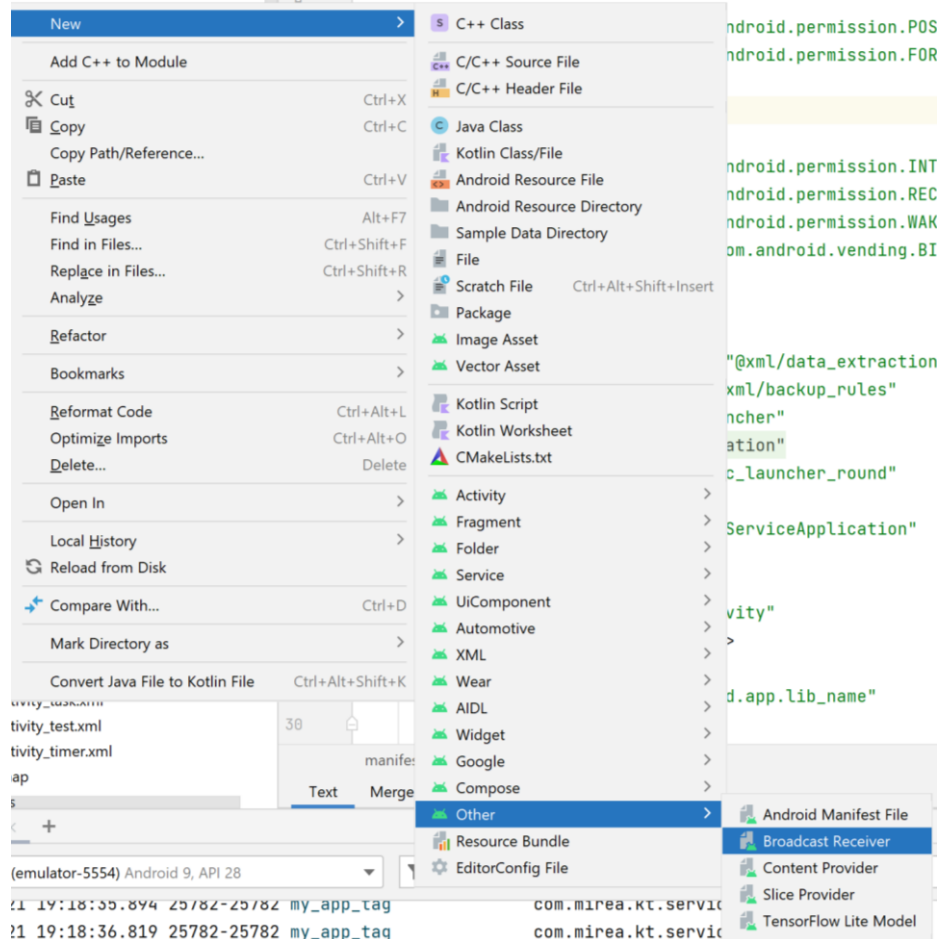


Рисунок 18

Название класса может быть любым удобным для вас. В данном случае будет присвоено имя `LanguageReceiver`, так как этот приемник будет «слушать» системное событие смены языка интерфейса устройства.

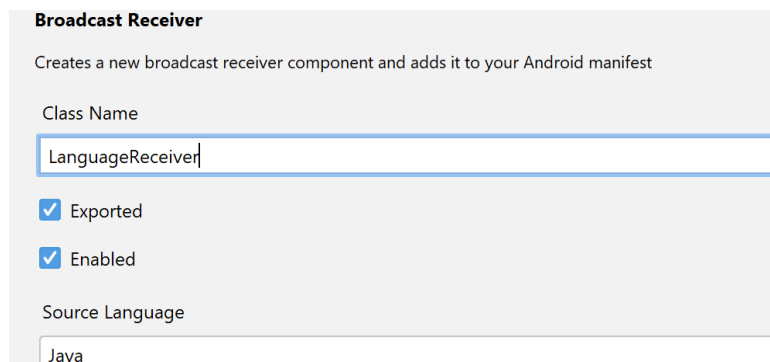


Рисунок 19

Регистрация этого приемника сообщений будет осуществлена статически в файле манифеста, так как событие смены языка одно из немногих, которое не попадает под ограничения введенных для Broadcast Receivers в Android 8 (см. лекцию №2.13). Перейдите в файл манифеста и в intentFilter ресивера добавьте информацию об action того события, которое приемник будет «слушать» (android.intent.action.LOCALE\_CHANGED).

```
<receiver
    android:name=".LanguageReceiver"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.LOCALE_CHANGED" />
    </intent-filter>
</receiver>
```

Рисунок 20

Теперь, устройстве произойдет смена языка, система отправит широковещательное сообщение с action равным android.intent.action.LOCALE\_CHANGED и будет выполнен код метода **onReceive** созданного вами приемника. Перейдите в класс LanguageReceiver и измените код метода onReceive (покажите какое-нибудь всплывающее сообщение при смене языка).

```
public class LanguageReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Log.d(LOG_TAG, msg: "onReceive message!");
        Toast.makeText(context.getApplicationContext(),
            context.getString(R.string.lang_change),
            Toast.LENGTH_LONG).show();
    }
}
```

Рисунок 21

Запустите приложение. Перейдите в настройки устройства и смените язык на любой другой. Убедитесь, что сообщение появилось и в логах отобразилась информация о том, ресивер сработал.