Laboratorium 3b – Metody wykonywania zadań w tle (na przykładzie komunikacji sieciowej)

Usługi – wykonywanie długotrwałych zadań w tle

Usługi stanowią jeden z podstawowych składników aplikacji. Usługi pozwalają na wykonywanie długotrwałych czynności, które powinny być kontynuowane nawet jeżeli użytkownik przełączy się na inną aplikację.

Service to komponent aplikacji, który może wykonywać długotrwałe operacje w tle. Usługi są przydatne do zadań takich jak odtwarzanie muzyki, pobieranie danych przez sieć, czy synchronizacja danych. Ważne jest, aby pamiętać, że domyślnie usługa działa w głównym wątku procesu aplikacji, w którym została uruchomiona. Oznacza to, że intensywne operacje wykonywane bezpośrednio w metodach usługi (np. `onStartCommand()`, `onBind()`) mogą zablokować główny wątek, powodując zawieszenie interfejsu użytkownika (ANR - Application Not Responding). Aby wykonać zadania wymagające czasu lub zasobów, takie jak operacje sieciowe czy obliczenia, należy przenieść je do osobnego wątku. Do zarządzania takimi zadaniami w tle i komunikacji z głównym wątkiem można wykorzystać klasy takie jak `Handler` w połączeniu z `HandlerThread`.

```
public class MyService extends Service {

    public final static String TAG = MyService.class.getSimpleName();

    public final static String PACKAGE_NAME = "com.example.lab3_tlo.service";
    public final static String PROGRESS_INFO = PACKAGE_NAME + ".progress_info";
    private static final String CHANNEL_ID = PACKAGE_NAME + ".service_channel";

    private static final int NOTIFICATION_ID_PROGRESS = 1;
    private static final int NOTIFICATION_ID_COMPLETE = 2;

    private NotificationManager notificationManager;
    private HandlerThread handlerThread;
    private Handler handler;

    //metoda odpowiedzialna za zwrócenie Bindera, w przypadku usług niezwiązanych
    //(unbounded) musi zwracać null
    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        Log.d(TAG, "onBind");
        return null;
    }

    @Override
    public void onCreate() {
        super.onCreate();
        //dodatkowy wątek do wykonywania zadań asynchronicznie (usługa działa w głównym
        //wątku aplikacji)
        handlerThread = new HandlerThread("DownloadService");
        handlerThread.start();
        //utworzenie handlera umożliwiającego wykonanie zadań w osobnym wątku
        handler = new Handler(handlerThread.getLooper());
    }


    //wywoływana gdy usługa zostanie uruchomiona za pomocą startService(intent)
```

```
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        //przygotowanie kanału powiadomień (usługa musi wyświetlać powiadomienie)
        notificationManager =
            (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
        prepareNotificationChannel();

        //wykonanie zadania za pomocą handlera
        handler.post(() -> executeTask(params));

        return START_NOT_STICKY;
    }

    //zadanie do wykonania
    protected void executeTask(ParamsType params) {
        //usuwanie starych powiadomień
        //...

        //przejście usługi na pierwszy plan
        //...

        //wykonanie zadania
        try {
            //nawiązanie połączenia
            //...

            //jeżeli plik istnieje – skasować
            //…

            //utworzenie nowego pliku
            //...

            //aktualizacja powiadomienia
            //...

            //odczytywanie danych ze strumienia sieciowego i zapis do strumienia
            //związanego z plikiem
            //...

            //usługa schodzi z pierwszego planu (aby można było usunąć powiadomienie)
            //...
            //aktualizacja powiadomienia
            //...
        } catch (Exception e) {
            e.printStackTrace();
            //jeżeli wystąpił błąd kasujemy plik
            //...
            //usługa schodzi z pierwszego planu (aby powiadomienie zniknęło)
            //...
            //aktualizacja powiadomień i statusu postępu
            //...
        } finally {
            //zamykanie strumieni i zakończenie połączenia
            //...
        }
    }
}
```

Podobnie jak w przypadku aktywności i dostawców treści informację o usłudze należy umieścić w manifeście. Zadeklarowanie usługi polega na dodaniu elementu `<service>` wewnątrz elementu `<application>`. Atrybut `android:name` określa nazwę usługi. Atrybut `android:exported` określa, czy

dana usługa jest też dostępna dla innych aplikacji. Od Androida 14 (API Level 34) atrybut `android:foregroundServiceType` jest wymagany i musi określać czym zajmuje się usługa (istnieje kilkanaście typów: https://developer.android.com/about/versions/14/changes/fgs-types-required)

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

    <!-- uprawnienia -->

    <application>
        <!-- aktywności itp. -->
        <service
            android:name=".DownloadService"
            android:foregroundServiceType="dataSync"
            android:exported="false">
        </service>
    </application>
    <!-- -->
</manifest>
```

Uruchomienie usługi jest proste i polega na utworzeniu odpowiedniej intencji (z danymi niezbędnymi do wykonania zadania) i wywołaniu metody `startService()`.

```
Intent intent = new Intent(this, MyService.class);
intent.putExtra(KEY, params);
startService(intent);
```

Usługi pierwszoplanowe

W Androidzie od wersji 8.0 – API Level 26, jeżeli usługa ma się wykonywać nawet wtedy gdy aplikacja nie jest widoczna na ekranie, to musi być pierwszoplanowe i wyświetlać powiadomienie (tak aby użytkownik wiedział, że usługa działa). Uruchomienie usługi pierwszoplanowej wymaga uprawnienia „`android.permission.FOREGROUND_SERVICE`” (omówionego w dalszej części instrukcji).

Najprostszym rozwiązaniem jest przejście usługi na pierwszy plan na początku wykonywanego zadania (warto zauważyć, że nie da się przejść na pierwszy plan bez utworzenia powiadomienia):

```
//przejście usługi na pierwszy plan
//Android 10 (API Level 29)
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
    startForeground(NOTIFICATION_ID_PROGRESS, getNotification(),
                    ServiceInfo.FOREGROUND_SERVICE_TYPE_DATA_SYNC);
} else {
    startForeground(NOTIFICATION_ID_PROGRESS, getNotification());
}
```

Po zakończeniu wykonania zadania usługa powinna zejść z pierwszego planu (i usunąć powiadomienie:

```
//usługa schodzi z pierwszego planu (aby powiadomienie zniknęło)
Log.d(TAG, "downloadFile - going to background");
stopForeground(STOP_FOREGROUND_REMOVE);
```

Wyświetlanie powiadomień

Jak już wspomniano, od Androida 8.0, usługa pierwszoplanowa musi wyświetlać powiadomienie. Również począwszy od Androida 8.0 powiadomienia są przypisywane do kanałów,

którymi użytkownik może zarządzać. Poniżej pokazano sposób tworzenia kanału powiadomień.

```
private void prepareNotificationChannel() {
    notificationManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);

    //tylko na Androidzie 8.0 i późniejszych
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        //tworzymy kanał
        NotificationChannel mChannel = new NotificationChannel(CHANNEL_ID,
                getString(R.string.notification_channel_name),
                NotificationManager.IMPORTANCE_LOW);
        mChannel.setDescription(getString(R.string.notification_channel_description));
        notificationManager.createNotificationChannel(mChannel);
    }
}
```

Ponieważ powiadomienia będą tworzone dość często (za każdym razem gdy chcemy wyświetlić/zaktualizować powiadomienie) najlepiej jest utworzyć metodę pomocniczą, która będzie tworzyła powiadomienie na podstawie bieżącego stanu wykonania zadania przez usługę.

```
//tworzenie powiadomienia
private Notification getNotification() {
    //tworzenie powiadomienia
    NotificationCompat.Builder builder =
        new NotificationCompat.Builder(this, CHANNEL_ID);
    builder.setContentTitle(getString(R.string.notification_title_downloading))
            .setContentText(getString(R.string.notification_text_downloading))
            .setProgress(100, percentage(), false)
            .setSmallIcon(R.drawable.baseline_arrow_circle_down_24)
            .setWhen(System.currentTimeMillis())
            .setPriority(Notification.PRIORITY_LOW); // dla Android 7.1-

    //w zależności o stanu pobierania ustawiamy różny tytuł
        //...
        builder.setContentTitle(getString(R.string.notification_text_finished));
        //...
        builder.setContentTitle(getString(R.string.notification_text_finished_error));
        //jeżeli pobieranie się zakończył poprawnie albo z błędem to automatyczne
        //zamykanie powiadomienia
        builder.setAutoCancel(true);
        //...
        //jeżeli pobieranie trwa to powiadomienie ma być ustawione jako trwające
        builder.setOngoing(true);
    }

    return builder.build();
}
```

Do wyświetlania powiadomień niezbędny jest menadżer powiadomień (można go odczytać w np. metodzie `onCreate` i zapisać w polu)

```
notificationManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
```

Wyświetlać powiadomienia można za pomocą metody:

```
//aktualizacja powiadomień
void sendMessagesAndUpdateNotification() {
    //WAŻNE!!! nie należy wysyłać powiadomień jeżeli postęp się nie zmienił lub zmienił
    //się nieznacznie. Różne wersje Androida optymalizują wyświetlanie powiadomień
    //(jeżeli wysyłamy powiadomienia za często nie wszystkie zostaną wyświetlone)
```

```
    //sprawdzić jaki jest stan pobierania
    //...
        //wyświetlić powiadomienie
        notificationManager.notify(NOTIFICATION_ID_PROGRESS, getNotification());

        //jeżeli pobieranie się zakończyło zalecane jest użycie innego identyfikatora
        //powiadomienia wtedy na pewno się wyświetli
        notificationManager.notify(NOTIFICATION_ID_COMPLETE, getNotification());
    }
}
```

Aby zaktualizować powiadomienie należy, po prostu, wyświetlić je ponownie używając tego samego identyfikatora.

Aby usunąć powiadomienie wystarczy użyć metody `cancel()`:

```
notificationManager.cancel(NOTIFICATION_ID_PROGRESS);
```

Dostęp do plików

W starszych wersjach Androida (przed Androidem 10/API 29) aplikacje miały szeroki dostęp do „zewnętrznej” pamięci przy użyciu bezpośrednich ścieżek plików i standardowych klas Javy jak java.io.File.

W Androidzie 10 i nowszych (używających Scoped Storage) dostęp do plików w udostępnionych kolekcjach (jak katalog Downloads, Pictures itp.) jest znacznie bardziej ograniczony ze względów bezpieczeństwa i prywatności. Aplikacje nie mają już swobodnego dostępu do wszystkich ścieżek. Zamiast tego, dostęp do tych plików odbywa się za pośrednictwem dostawcy zawartości MediaStore i obiektu ContentResolver. Pliki nie jest tworzony bezpośrednio jako java.io.File pod konkretną ścieżką. Zamiast tego, aplikacja:

- Wyszukuje istniejący plik w bazie MediaStore za pomocą zapytania bazując na nazwie pliku i uzyskuje jego Uri
- Może usunąć wcześniej utworzony przez siebie plik używając ContentResolver.delete()
- Nowy plik tworzy poprzez utworzenie nowego wpisu w bazie MediaStore, używając ContentValues do zdefiniowania metadanych (jak nazwa pliku, typ MIME, status "oczekujący" - IS_PENDING) i ścieżki względnej (Environment.DIRECTORY_DOWNLOADS)
- Wstawia ten wpis do bazy za pomocą getContentResolver().insert() i otrzymuje Uri wskazujący na ten nowy wpis.
- Strumień do zapisu danych (fileOutputStream) jest otwierany za pomocą getContentResolver().openOutputStream(fileUri)
- Po zakończeniu zapisu, status w bazie MediaStore jest aktualizowany (getContentResolver().update()) (ustawienie IS_PENDING na 0).

Sprawdzenie czy plik istnieje (ewentualne skasowanie) i utworzenie pliku do zapisu można zrealizować następująco:

```
ContentValues values = new ContentValues();
//Android 10 (API Level 29) - używa scoped storage, mamy dostęp do katalogu Downloads
//ale trzeba dodać plik do bazy plików
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
    //jeżeli plik istnieje to go usuwamy
    Uri tmpUri = getFileUri(fileName);
    if (tmpUri != null) {
```

```
        getContentResolver().delete(tmpUri, null, null);
    }
    //tworzymy nowy wpis w bazie plików
    ContentResolver resolver = getContentResolver();
    values.put(MediaStore.Downloads.DISPLAY_NAME, fileName);
    values.put(MediaStore.Downloads.MIME_TYPE, mimeType);
    values.put(MediaStore.Downloads.IS_PENDING, 1);
    values.put(MediaStore.Downloads.RELATIVE_PATH, Environment.DIRECTORY_DOWNLOADS);
    fileUri = resolver.insert(MediaStore.Downloads.EXTERNAL_CONTENT_URI, values);
    fileOutputStream = resolver.openOutputStream(fileUri);
} else { //w przypadku starszych wersji używamy standardowych ścieżek
    File outFile = new File(Environment.getExternalStorageDirectory() + File.separator
                            + Environment.DIRECTORY_DOWNLOADS + File.separator
                            + fileName);
    if (outFile.exists()) outFile.delete();
    fileOutputStream = new FileOutputStream(outFile);
}
```

Pomocnicza metoda odczytująca Uri do istniejącego pliku (tylko dla Androida 10 i późniejszych)

```
//Android 10 (API Level 29) - pobieranie URI istniejącego pliku z bazy plików
@RequiresApi(api = Build.VERSION_CODES.Q)
private Uri getFileUri(String fileName) {
    ContentResolver resolver = getContentResolver();
    String[] projection = {MediaStore.Downloads._ID};
    String selection = MediaStore.Downloads.DISPLAY_NAME + " = ?";
    String[] selectionArgs = {fileName};
    try (Cursor cursor = resolver.query(MediaStore.Downloads.EXTERNAL_CONTENT_URI,
                                        projection, selection, selectionArgs, null)) {
        if (cursor != null && cursor.moveToFirst()) {
            int idColumn = cursor.getColumnIndexOrThrow(MediaStore.Downloads._ID);
            long id = cursor.getLong(idColumn);
            return Uri.withAppendedPath(MediaStore.Downloads.EXTERNAL_CONTENT_URI,
                                        String.valueOf(id));
        } else {
            return null;
        }
    } catch (Exception e) {
        return null;
    }
}
```

Po zakończeniu zapisywania pliku należy pamiętać o zamknięciu strumienia:

```
if (fileOutputStream != null) {
    try {
        fileOutputStream.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Komunikacja sieciowa

Otwieranie połączenia odbywa się dokładnie tak samo jak w przypadku pobierania informacji o pliku w poprzedniej instrukcji. Odbieranie danych z połączenia sieciowego sprowadza się do kopiowania danych między strumieniami. Strumień połączenia sieciowego można uzyskać za pomocą metody `getInputStream()` połączenia sieciowego. Odbieranie danych odbywa się porcjami o rozmiarze nie przekraczającym rozmiaru bufora.

```
//odczytywanie danych ze strumienia sieciowego i zapis do strumienia związanego z
```

```
//plikiem
DataInputStream reader = new DataInputStream(connection.getInputStream());

byte buffer[] = new byte[BLOCK_SIZE];
int downloaded = reader.read(buffer, 0, BLOCK_SIZE);
while (downloaded != -1) {
    fileOutputStream.write(buffer, 0, downloaded);
    bytesDownloaded += downloaded;
    //aktualizacja powiadomienia
    //...
    downloaded = reader.read(buffer, 0, BLOCK_SIZE);
}
```

Po odebraniu całego pliku należy zamknąć strumień połączenia sieciowego. Tak jak to pokazano poniżej. Warto zwrócić uwagę na obsługę wyjątków oraz sprawdzenie czy strumień w ogóle istnieje.

```
if (networkInputStream != null) {
    try {
        networkInputStream.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Po zakończeniu komunikacji należy zamknąć połączenie.

```
if (connection != null) connection.disconnect();
```

Uprawnienia aplikacji

Ćwiczenie wymaga dodatkowych uprawnień, które pokazano poniżej

```
<!-- uprawnienia -->
<uses-permission android:name="android.permission.POST_NOTIFICATIONS" />
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
<uses-permission android:name="android.permission.FOREGROUND_SERVICE_DATA_SYNC" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
 android:maxSdkVersion="29" />
```

Je również należy dodać do manifestu.

Uprawnienie `POST_NOTIFICATIONS` pozwala aplikacji na wysyłanie powiadomień. Wymagane jest w systemie Android 13 (API Level 33) i późniejszych. Uprawnienie wymaga zgody użytkownika.

Uprawnienie `FOREGROUND_SERVICE` umożliwia tworzenie usług pierwszoplanowych. Poza dodaniem do manifestu nie wymaga dodatkowych działań (ma normalny poziom ochrony).

Uprawnienie `FOREGROUND_SERVICE_DATA_SYNC` dodatkowo określa rodzaj zadania realizowanego przez usługę. Jest wymagane od Androida 14 (API Level 34). Nie wymaga zgody użytkownika.

Uprawnienie `WRITE_EXTERNAL_STORAGE` umożliwia zapis (i odczyt) danych z karty pamięci (lub emulowanej karty pamięci w urządzeniach, które nie mają takiego slotu). To uprawnienie ma poziom ochrony „niebezpieczny” i wymaga poproszenia użytkownika o przyznanie uprawnienia w trakcie wykonania programu. Atrybut `android:maxSdkVersion` mówi o tym, że aplikacja nie będzie używała tego uprawnienia na nowszym niż Android 10 (API Level 29)

Podsumowując – w aplikacji pobierającej pliki zgody wymagają – wyświetlanie

powiadomień (Android 13+) i dostęp do karty pamięci (Android 9-)

Poniżej przedstawiono sposób proszenia użytkownika o przyznanie uprawnień (kod należy umieścić w głównej aktywności). Sposób wykorzystuje bibliotekę *appcompat* dołączaną domyślnie do każdego projektu i jest kompatybilny z wersjami Androida, które nie wymagają proszenia o uprawnienia w trakcie wykonania programu (nie trzeba tworzyć wielu wersji kodu).

```
//o uprawnienia prosimy tylko gdy musimy (sprawdzić wersję Androida)
if (ActivityCompat.checkSelfPermission(this, requiredPermission) ==
    PackageManager.PERMISSION_GRANTED) {
    //już mamy uprawnienie możemy rozpocząć pobieranie
    //...
} else if (ActivityCompat.shouldShowRequestPermissionRationale(this,
                                                               requiredPermission)) {
    //musimy wyjaśnić użytkownikowi po co nam uprawnienie
    //...
} else {
    //nie mamy uprawnień - prosimy o uprawnienie
    //...
    ActivityCompat.requestPermissions(this, new String[]{requiredPermission},
                                      PERMISSIONS_REQUEST_CODE);
}
```

Aplikacja wyświetli użytkownikowi okienko (okienka) z pytaniem o zgodę. Po tym jak użytkownik podejmie decyzję zostanie wywołana metoda `onRequestPermissionsResults()`. Otrzymujemy w niej kod żądania, tablicę z uprawnieniami, o które prosiliśmy i tablicę z decyzjami użytkownika odnośnie uprawnień z pierwszej tablicy.

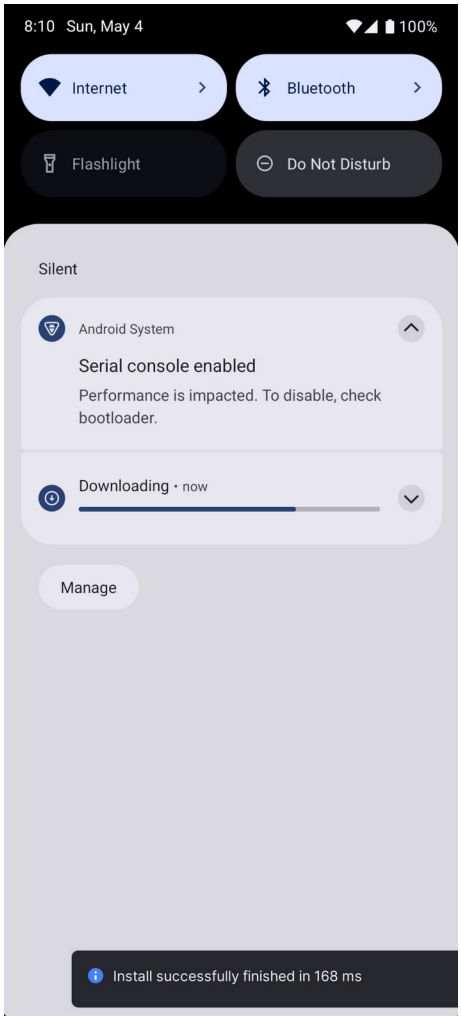
```
@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions,
                                       int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    String requiredPermission = getRequiredPermission();
    switch (requestCode) {
        case PERMISSIONS_REQUEST_CODE:
            if (permissions.length > 0 && permissions[0].equals(requiredPermission) &&
                grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                //użytkownik przyznał uprawnienie - można rozpocząć pobieranie
                //...
            } else
                //użytkownik się nie zgodził
                //...
            break;
        default:
            //nieznane uprawnienie
            //...
            break;
    }
}
```

Metoda pomocnicza, która sprawdza jakie uprawnienia są potrzebne w bieżącej wersji Androida:

```
private static String getRequiredPermission() {
    String requiredPermission = "";
    //Android 13+ (API 33+)
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
        requiredPermission = Manifest.permission.POST_NOTIFICATIONS;
    }
    //Android wcześniejszy niż 10 (API 29)
    else if (Build.VERSION.SDK_INT < Build.VERSION_CODES.Q) {
```

```
            requiredPermission = Manifest.permission.WRITE_EXTERNAL_STORAGE;
        }
        return requiredPermission;
    }
```

Ćwiczenie 3b



Dodaj do aplikacji możliwość pobierania wskazanego pliku. Pobierany plik powinien być zapisywany w domyślnej lokalizacji pobieranych plików. Aplikacja musi korzystać z usługi pierwszoplanowej, prosić o dostęp do karty pamięci/możliwość wyświetlania powiadomień i wyświetlać powiadomienie. Aplikacja musi działać na Androidzie 7+

Wskazówki:

- proszę pamiętać że wszystkie uprawnienia muszą być wymienione w manifeście (w sumie 5 uprawnień)
- usługa musi być wymieniona w manifeście
- aplikacja musi poprosić o odpowiednie uprawnienie w zależności od wersji Androida, na której aktualnie działa
- powiadomienie może być prostsze niż to przedstawione w przykładzie i nie musi pokazywać postępu pobierania; nie musi też umożliwiać powrotu do aplikacji po naciśnięciu powiadomienia. Musi jednak pokazywać czy pobieranie trwa czy już się zakończyło (czy pomyślnie czy z błędem)
- informacje o postępach pobierania należy wysyłać do *logcata* za pomocą klasy `Log`.

Pasek postępu i wskaźniki pobierania zostaną zaimplementowane w ramach kolejnego ćwiczenia.