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

Komunikacja składników aplikacji

W poprzedniej instrukcji pokazano usługę niezwiązaną (unbounded service) uruchamianą za pomocą metody `startService()`. Aby przekazywać dane z usługi można wykorzystać możliwość wiązania z usługą (i tworzenia bounded service). Wiązanie umożliwia bezpośrednie wywoływanie metod usługi np. z aktywności. O tym jakie metody usługi są dostępne z zewnątrz decyduje `Binder` definiowany i zwracany przez usługę. Użycie wiązania nie wyklucza uruchamiania wykonania zadań za pomocą `startService()`.

Do przekazywania danych można wykorzystać obiekty `LiveData`, które wykorzystywaliśmy w aplikacji wykorzystującej bibliotekę Room. W usłudze można stworzyć taki obiekt (a w zasadzie obiekt klasy `MutableLiveData`, który pozwala na modyfikowanie danych), za pomocą bindera przekazać go do aktywności, w usłudze zapisywać do niego dane a w aktywności obserwować zmiany danych.

Poniżej przedstawiono modyfikacje klasy usługi, które dodają możliwość wiązania.

```
public class MyService extends Service {

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

    //obiekt LiveData do przekazywana informacji o postępie do aktywności
    private MutableLiveData<ProgressEvent> progressLiveData =
        new MutableLiveData<>(null);

    //Binder definiuje metody, które można wywoływać z zewnątrz np. z aktywności
    public class MyServiceBinder extends Binder {
        LiveData<ProgressEvent> getProgressEvent() {
            return progressLiveData;
        }
    }

    private final IBinder binder = new MyServiceBinder();

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

    @Override
    public boolean onUnbind(Intent intent) {
        return super.onUnbind(intent);
    }

    //aktualizacja powiadomień i statusu postępu
    void sendMessagesAndUpdateNotification() {
        //sprawdzić jaki jest stan pobierania
        //...
            //i zaktualizwać stan pobierania (postValue a nie setValue bo ten kod
            //wykonuje się w dodatkowym wątku)
            progressLiveData.postValue(new ProgressEvent(progress, total, result));
            //...
    }
```

```
}
```

Wiązanie z usługą

Aby można było wywoływać metody usługi z poziomu aktywności należy aktywność związać z usługą. Służy do tego metoda `bindService()`. Wymaga ona przekazania wywołania zwrotnego, które reaguje na zdarzenia połączenia i odłączenia od usługi.

Wiązanie można wykorzystać do uruchomienia obserwowania obiektu `LiveData`, przez który będą przekazywane dane z usługi.

Poniżej przedstawiono modyfikacje klasy aktywności, które dodają wiązanie z usługą.

```
public class MainActivity extends AppCompatActivity {

    private static final String TAG = MainActivity.class.getSimpleName();

    //czy mamy połącznie z usługą
    private boolean serviceBound = false;
    //dane z usługi
    private LiveData<ProgressEvent> progressEventLiveData;

    //reagowanie na połączenie/rozłączenie z usługą
    private ServiceConnection serviceConnection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName componentName, IBinder iBinder) {
            serviceBound = true;
            MyService.MyServiceBinder downloadBinder =
                (MyService.MyServiceBinder) iBinder;
            //zapisanie i włączenie obserwowania obiektu LiveData z danymi z usługi
            progressEventLiveData = binder.getProgressEvent();
            progressEventLiveData.observe(MainActivity.this,
                                          MainActivity.this::updateProgress);
        }

        @Override
        public void onServiceDisconnected(ComponentName componentName) {
            //wyłączenie obserwowania
            progressEventLiveData.removeObservers(MainActivity.this);
            serviceBound = false;
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //...
        //powiązanie z usługą
        Intent myServiceIntent = new Intent(this, MyService.class);
        bindService(myServiceIntent, serviceConnection,
                    Context.BIND_AUTO_CREATE);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        //odłączenie usługi
        unbindService(serviceConnection);
        //...
    }
}
```

Interfejs Parcelable

Intencje i związane z nimi obiekty typu `Bundle` posiadają szereg metod do przekazywania wartości typów prostych takich jak wartości całkowite czy łańcuchy tekstowe. Czasami jednak zachodzi konieczność (albo po prostu jest wygodniej) przekazać cały obiekt. Można do tego zastosować interfejs `Parcelable`. Po jego zaimplementowaniu w danej klasie możliwe jest zapisanie całego obiektu w obiekcie klasy `Bundle`, a następnie jego odtworzenie na podstawie zapisanych danych (nie jest to przekazanie tej samej instancji obiektu, lecz utworzenie jego klona – innego obiektu z identycznymi wartościami).

Poniżej przedstawiono klasę `ProgressEvent`. Dzięki temu, że implementuje interfejs `Parcelable` można obiekty tej klasy zapisać w całości do intencji bądź obiektu `Bundle`. Najistotniejsze elementy klasy to:

- konstruktor z parametrem typu `Parcel` - jego zadaniem jest utworzenie obiektu na podstawie otrzymanej paczki (odczytując dane z paczki należy zwrócić uwagę na kolejność). Do odczytywania wartości, stosowane są metody `readNazwaTypu()` (nie tylko typy proste ale także tablice i wszystkie typy implementujące `Parcelable`).
- metoda `writeToParcel()` - jej zadaniem jest zapisanie stanu obiektu do paczki otrzymanej jako parametr. Dane do paczki zapisuje się za pomocą metod `writeNazwaTypu()` (nie tylko typy proste ale także tablice i wszystkie typy implementujące `Parcelable`).
- statyczne pole `CREATOR` typu `Parcelable.Creator<ProgressEvent>` - obowiązkowy element, który jest odpowiedzialny za tworzenie obiektów i tablic tego typu

```
public class ProgressEvent implements Parcelable
{
    //statusy pobierania
    public static final int OK =0;
    public static final int IN_PROGRESS =1;
    public static final int ERROR =2;

    public int progress;
    public int total;
    public int result;

    //inne potrzebne konstruktory...

    //interfejs Parcelable
    protected ProgressEvent(Parcel in) {
        progress = in.readInt();
        total = in.readInt();
        result = in.readInt();
    }

    public static final Creator<ProgressEvent> CREATOR = new Creator<ProgressEvent>() {
        @Override
        public ProgressEvent createFromParcel(Parcel in) {
            return new ProgressEvent(in);
        }

        @Override
        public ProgressEvent[] newArray(int size) {
            return new ProgressEvent[size];
        }
    };

    @Override
    public int describeContents() {
```

```
        return 0;
    }

    @Override
    public void writeToParcel(@NonNull Parcel dest, int flags) {
        dest.writeInt(progress);
        dest.writeInt(total);
        dest.writeInt(result);
    }
}
```

Tworzenie powiadomień wykonujących akcje

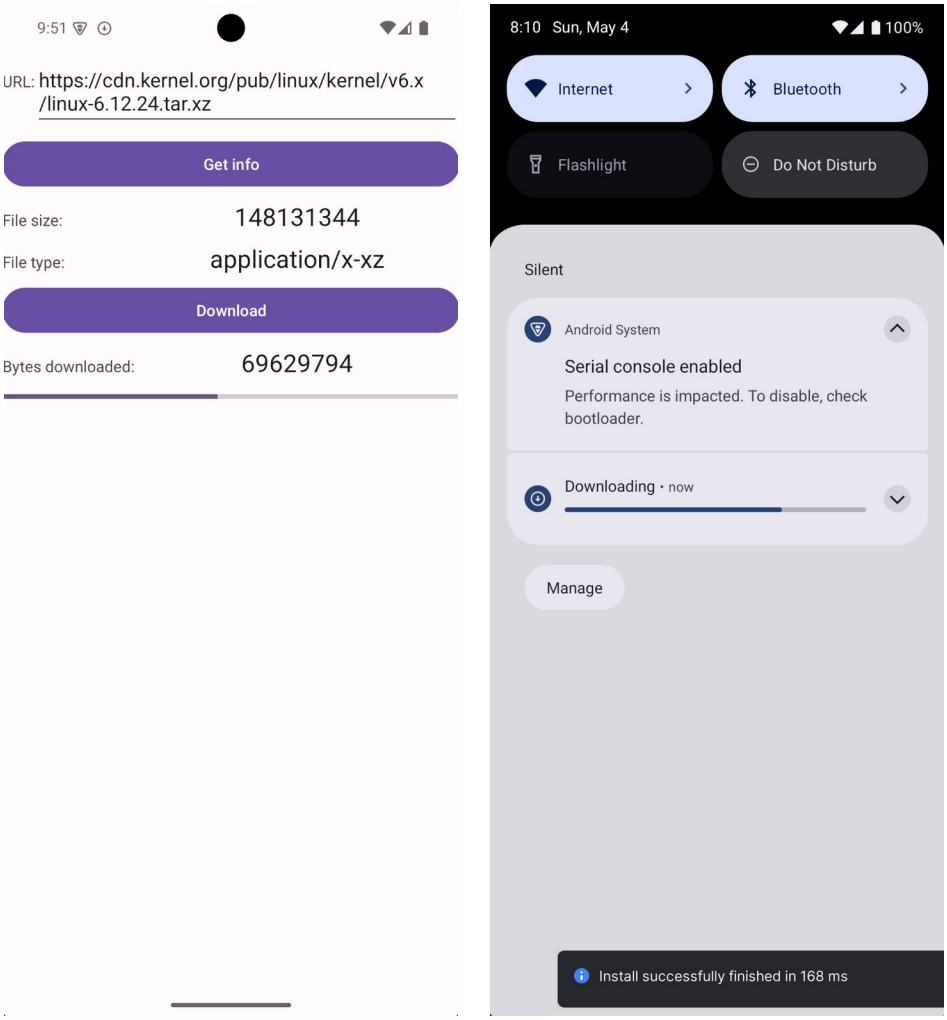
Powiadomienia mogą powodować przejście o aplikacji, która je wyświetliła. Realizuje się to poprzez stworzenie intencji oczekującej, która zawiera wszystkie potrzebne dane i uruchamia właściwą aktywność w stanie takim, w jakim aplikacja powinna być biorąc pod uwagę bieżący status wykonania zadania w tle.

Poniżej przedstawiono modyfikacje metody tworzącej powiadomienie dodającej tworzenie intencji oczekującej.

W aktywności uruchomionej przez dotknięcie powiadomienia zapisane dane można odczytać tak samo jak w przypadku przekazywania danych między aktywnościami (Aplikacja #1)

```
    //tworzenie powiadomienia
    private Notification getNotification() {
        //tworzenie intencji oczekującej uruchamiającej
        Intent notificationIntent = new Intent(this, MainActivity.class);
        //zapisanie potrzebnych danych (może być więcej niż jeden element)
        notificationIntent.putExtra(KEY, data);

        //utworzenie stosu aktywności
        TaskStackBuilder taskStackBuilder = TaskStackBuilder.create(this);
        taskStackBuilder.addParentStack(MainActivity.class);
        taskStackBuilder.addNextIntent(notificationIntent);

        //flaga FLAG_IMMUTABLE jest wymagaja dla Androida 12 (API Level 31) i
        //późniejszych
        PendingIntent pendingIntent = taskStackBuilder.getPendingIntent(0,
                PendingIntent.FLAG_UPDATE_CURRENT | PendingIntent.FLAG_IMMUTABLE);

        //tworzenie powiadomienia
        NotificationCompat.Builder builder =
            new NotificationCompat.Builder(this, CHANNEL_ID);
        //tworzenie powiadomienia
            //...
            builder.setContentIntent(pendingIntent)
            //...

        return builder.build();
    }
```

Ćwiczenie 3c



Ćwiczenie polega na dodaniu do aplikacji wyświetlania postępu pobierania pliku w głównej aktywności, rozbudowaniu powiadomienia wyświetlanego przez usługę, dodaniu zachowywania stanu aktywności przy obróceniu urządzenia.

Wskazówki:

- Dodaj do projektu klasę `ProgressEvent` zawierającą informacje o statusie pobierania. Będzie ona służyła do przechowywania liczby bajtów pobranych do danego momentu, rozmiaru pobieranego pliku, wyniku pobierania (pobieranie trwa, pobieranie zakończone lub błąd). Zaimplementuj w tej klasie interfejs `Parcelable`.
- Zaimplementuj w usłudze pobierającej plik mechanizm wiązania i dodaj zapisywanie danych o postępie do obiektu `LiveData`.
- Dodaj do aktywności mechanizm wiązania i dodaj obserwowanie obiektu `LiveData`. Dane o postępie mają być wykorzystywane do aktualizacji etykiety z liczbą pobranych bajtów oraz paska postępu.
- Rozbuduj wyświetlane powiadomienia tak aby:
  - kliknięcie powiadomienia powodowało powrót do głównej aktywności aplikacji (stan pobierania ma być aktualny – zapisz go w intencji oczekującej a w aktywności odczytaj stan z intencji w metodzie `onCreate`)
  - w powiadomieniu wyświetlany był pasek postępu pobierania z aktualnym postępem
- Dodaj zachowywanie stanu głównej aktywności (tak aby obrót urządzenia nie powodował

np. zniknięcia pobranych informacji o pliku albo nie przywracał paska postępu do wartości początkowej).