Kotlin & Android w AndroidStudio (3.5.1)

Aplikacja Kontakty. Wykonywanie połączeń i wysyłanie SMS.

RecyclerView i Adapter

Prezentowanie danych w postaci listy elementów to bardzo popularny wzorzec aplikacji mobilnych. Wystarczy otworzyć kilka aplikacji od Google, aby się o tym przekonać. Google Play wyświetla aplikacje w postaci listy. Gmail wyświetla e-maile w postaci listy. Google+ również wyświetla zawartość w postaci listy elementów.

Istnieją dwie klasy, które umożliwiają wyświetlanie list: **ListView** oraz **GridView**. Pierwsza klasa wyświetlała elementy pionowo, jeden pod drugim. Klasa druga prezentowała treść w postaci siatki, np. po trzy elementy w poziomie. Są to klasy, które w zasadzie wystarczają do najprostszych zastosowań.

Klasa **RecyclerView** to pojemnik do renderowania większego zestawu danych widoków, które można bardzo skutecznie przetwarzać i przewijać. RecyclerView przypomina tradycyjny ListView, ale z większą elastycznością w dostosowywaniu i optymalizowaniu do pracy z większymi zestawami danych.

Kolejnym elementem jest **Adapter**, czyli klasa, która przechowuje i zarządza danymi do wyświetlenia. Lista tylko wyświetla elementy, które dostaje właśnie od adaptera. Dla ListView oraz GridView możemy rozszerzyć klasę BaseAdapter. Dla RecyclerView musimy rozszerzyć klasę RecyclerView.Adapter

Tworzenie listy kontaktów

Naszym celem jest utworzenie aplikacji umożliwiającej wyświetlenie listy kontaktów oraz wykonywanie połączeń i wysyłania wiadomości SMS do wybranego kontaktu.

Zaczynamy od listy kontaktów.

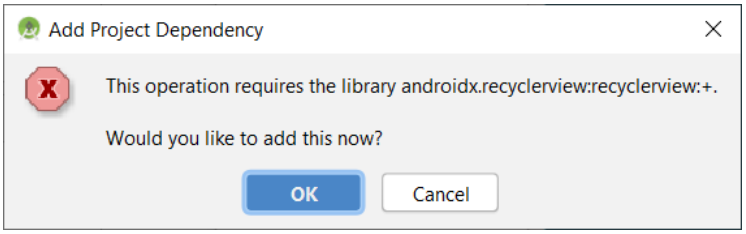
Lista kontaktów z pliku bazy kontaktów

Na początku utworzymy aplikację wyświetlającą kontakty, które wczytywane będą z pliku.

ZADANIE

Utwórz nowy projekt o nazwie **MyContactsApplication** zawierający **EmptyActivity**.

Do layoutu **ConstraintLayout** dodaj komponent **RecyclerView**. Dołącz potrzebną bibliotekę.



Ustaw id komponentu na **recyclerView**.

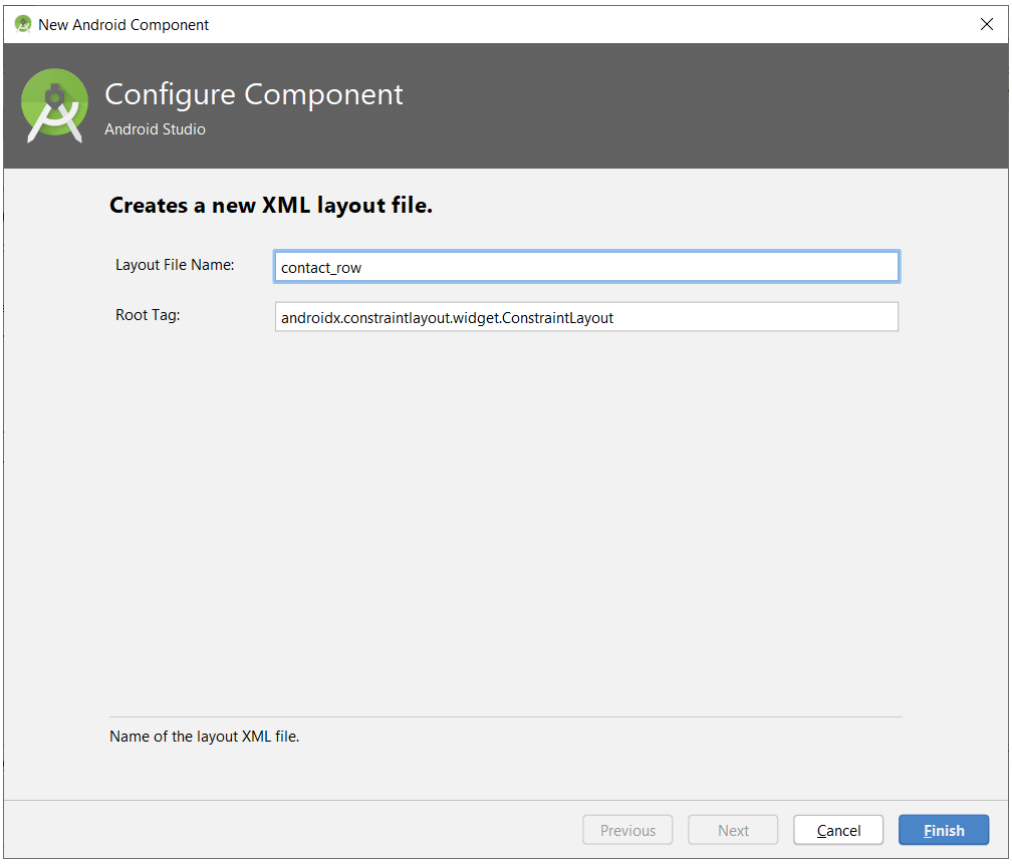
Pojedynczy kontakt w naszej aplikacji będzie reprezentowany przez 2 komponenty **TextView** odpowiednio do wyświetlenia nazwy kontaktu i numeru telefonu, a także przez przycisk umożliwiający wykonanie połączenia do wybranego kontaktu.

Na naszej liście kontaktów będziemy mieli wiele takich kontaktów, które reprezentowane są przez takie same komponenty zaś różnią się danymi. Mamy tutaj pewną powtarzalność - każdy kontakt wygląda identycznie. Ponadto lista kontaktów musi być dynamiczna, tzn. liczba wyświetlanych kontaktów musi się zgadzać z liczbą kontaktów w bazie i w razie zmian musi nastąpić aktualizacja. Dlatego idealnym rozwiązaniem jest RecyclerView.

Utworzymy teraz nowy plik layoutu, który reprezentował będzie pojedynczy kontakt.

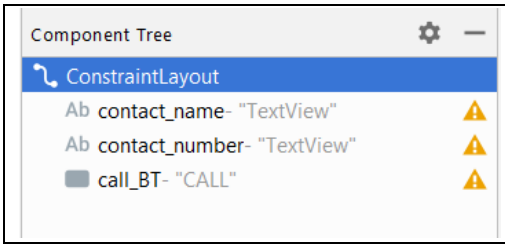
ZADANIE

Utwórz nowy plik layout o nazwie **contact_row**. W tym celu kliknij PPM na pakiet layout i wybierz: **New/XML/Layout XML File**. W polu **Root Tag** wpisz: **androidx.constraintlayout.widget.ConstraintLayout**.



Do layoutu **ConstraintLayout** dodaj 2 komponenty **TextView** oraz **Button**:

Zmień identyfikatory komponentów:



Mamy już utworzony widok pojedynczego kontaktu. Pora dodać widok(i) kontaktu do listy kontaktów.

ZADANIE

Przejdź do kodu **MainActivity**. Wprowadź następujące zmiany kodu:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    // Element odpowiedzialny za ustawienie widokow w formie listy
    recyclerView.layoutManager = LinearLayoutManager( context: this)
}
```

LinearLayoutManager będzie odpowiedzialny za ustawienie naszych widoków kontaktów w formie listy jeden pod drugim.

Teraz dla naszego **recyclerView** powinniśmy ustawić adapter, który zapewni połączenie między naszym widokiem a bazą danych z kontaktami. Musimy go najpierw utworzyć.

ZADANIE

Utwórz nową klasę kotlin i nazwij ją **MyAdapter**. W utworzonym pliku wprowadź kod:

```
1   package android.mycontactsapplication
2
3   import android.view.LayoutInflater
4   import android.view.View
5   import android.view.ViewGroup
6   import androidx.recyclerview.widget.RecyclerView
7
8   class MyAdapter: RecyclerView.Adapter<MyViewHlder>(){
9       override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyViewHlder {
10          val layoutInflater = LayoutInflater.from(parent.context)
11          val contactRow = layoutInflater.inflate(R.layout.contact_row, parent, attachToRoot: false)
12          return MyViewHlder(contactRow)
13      }
14
15      // ile layoutow mamy utworzyc
16      override fun getItemCount(): Int {
17          return 3
18      }
19
20      override fun onBindViewHolder(holder: MyViewHlder, position: Int) {
21
22      }
23  }
24
25  class MyViewHlder(view:View): RecyclerView.ViewHolder(view)
```

W linii 8 naszemu adapterowi musimy dostarczyć typ **ViewHolder**. Tworzymy zatem własną klasę **MyViewHolder**, która przekształca **View** na **ViewHolder**. Implementujemy też niezbędne metody.

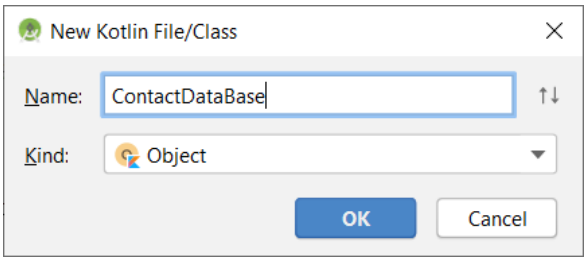
Dalej w metodzie **getItemCount()** „na sztywno” ustawiamy liczbę kontaktów (widoków kontaktu), które będą wyświetlane na liście kontaktów. Zajmiemy się tym nieco później, gdy utworzymy plik z kontaktami.

W metodzie **onCreateViewHolder()**, **LayoutInflater** tworzy z kodu XML layoutu obiekt. Zmienna **contactRow** jest typu **View** i reprezentuje widok pojedynczego kontaktu z pliku **contact_row.xml**. Zwracamy pojedynczy kontakt jako **ViewHolder**.

Teraz musimy utworzyć plik z kontaktami.

ZADANIE

Utwórz nową klasę kotlin typu **Object** i nazwij ją **ContactDataBase**.



Umieść w pliku listę kilku kontaktów, np:

```
object ContactDataBase {
    var contactList = arrayListOf<String>(
        "Jan Kowalski",
        "Janusz",
        "Grazyna",
        "Pjoter",
        "Brajan",
        "Dzesika",
        "Brajanusz",
        "Karyna"
    )
    var contactNumberList = arrayListOf<String>(
        "123456789",
        "987654321",
        "111222333",
        "444555666",
        "77888999",
        "135791357",
        "111000111",
        "999999999"
    )
}
```

W utworzonym pliku mamy 2 ArrayListy stringów odpowiednio z nazwami kontaktów i z numerami telefonu.

Teraz oprogramujemy metodę **onBindViewHolder()** oraz **getItemCount()** w **MyAdapter**.

ZADANIE

Dopisz poniższy kod metod klasy **MyAdapter**.

```
// ile layoutow mamy utworzyc
override fun getItemCount(): Int {
    return ContactDataBase.contactList.size // zwroc rozmiar listy kontaktow
}

// metoda aktualizujaca nasze widoki
override fun onBindViewHolder(holder: MyViewHlder, position: Int) {
    val name = holder.itemView.contact_name // TextView przechowujace nazwe kontaktu
    val number = holder.itemView.contact_number // TextView z numerem kontaktu

    // pobieranie danych z bazy kontaktow i ustawienie jako tekstu w TextView
    name.setText(ContactDataBase.contactList[position])
    number.setText(ContactDataBase.contactNumberList[position])
}
```

Kod adaptera jest już gotowy, wracamy do **MainActivity** aby ustawić adapter dla **RecyclerView**.

ZADANIE

Dopisz brakujący kod w **MainActivity**.

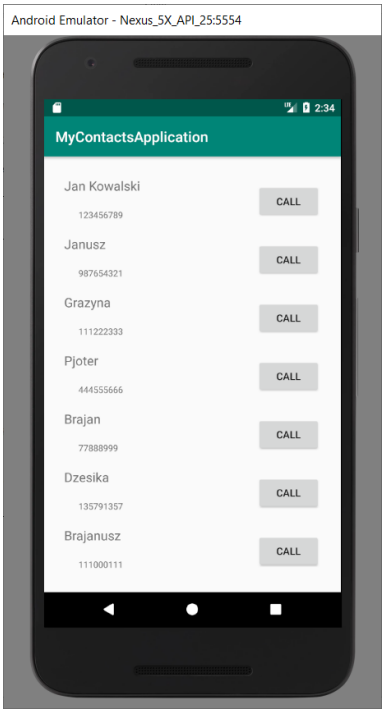
```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Element odpowiedzialny za ustawienie widokow w formie listy
        recyclerView.layoutManager = LinearLayoutManager( context: this)
        // Element odpowiedzialny za polaczenie miedzy danymi a widokami
        recyclerView.adapter = MyAdapter()
    }
}
```

Uruchom aplikację i sprawdź jej działanie.

Aplikacja działa poprawnie, tzn. wyświetla listę wszystkich kontaktów pobraną z naszej bazy.



ZADANIE

Dopisz kilka nowych kontaktów do naszej bazy i sprawdź czy aplikacja działa poprawnie.

Lista kontaktów z systemowej aplikacji Kontakty

Teraz ulepszymy naszą aplikację tak aby pokazywała prawdziwe kontakty z naszego telefonu, a nie te z prowizorycznej bazy.

Ważnymi pojęciami tutaj będą **content provider** i **content resolver**. Content provider udostępnia dane jednej aplikacji dla drugiej aplikacji. Content resolver z kolei daje nam możliwość pracy na dostarczonych danych.

ZADANIE

Zmodyfikuj kod **MainActivity**.

```
class MainActivity : AppCompatActivity() {

    companion object {
        var listaKontaktow = arrayListOf<String>()
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        recyclerView.layoutManager = LinearLayoutManager( context: this) // Element odpowiedzialny za ustawienie widokow w formie listy
        recyclerView.adapter = MyAdapter() // Element odpowiedzialny za polaczenie miedzy danymi a widokami

        val contentResolver = contentResolver
        val cursor = contentResolver.query(ContactsContract.Contacts.CONTENT_URI,
            projection: null, selection: null, selectionArgs: null, sortOrder: null, cancellationSignal: null)
        try{
            cursor.moveToFirst() // umieszczenie kursora na 1 rekordzie
            while(!cursor.isAfterLast){ // dopoki nie wskazuje na pozycje po ostatnim rekordzie ...
                // pobranie nazwy kontaktu i dodanie do listy
                val name = cursor.getString(cursor.getColumnIndexOrThrow(ContactsContract.Contacts.DISPLAY_NAME_PRIMARY))
                listaKontaktow.add(name)
                // przejscie do kolejnego rekordu
                cursor.moveToNext()
            }
        }
        finally {
            cursor.close() // zamkniecie kursora (dostepu do danych)
        }
    }
}
```

Zaznaczona linia nawiązuje trochę do języka MySQL. Wysyłamy zapytanie do bazy danych i ono zwróci nam tabelę z naszymi kontaktami. Widoczne wartości **null** odpowiadają klauzulom MySQL. Zmienna **cursor** jest „uchwytem” do tabeli z kontaktami i wskazuje na pojedynczy wiersz (rekord tabeli). Aby „wyciągnąć” dane z konkretnej komórki tabeli musimy dostarczyć naszemu kursorowi numer interesującego nas wiersza i kolumny w tym wierszu.

Operacje dostępu do danych wykonywane są w sekcji **try … catch** ponieważ mogą wystąpić nieoczekiwane sytuacje np. odmowa dostępu.

Kontakty zapisane w telefonie są chronione, dlatego nasza aplikacja potrzebować będzie uprawnień dostępu do nich. Odpowiednie uprawnienia definiujemy w pliku manifestu **AndroidManifest.xml**.

ZADANIE

Dodaj uprawnienia odczytu i zapisu kontaktów.

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

    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <uses-permission android:name="android.permission.WRITE_CONTACTS"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="MyContactsApplication"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <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>
```

Pozostaje jeszcze zmodyfikować kod adaptera odwołując się do listy kontaktów pobranej z telefonu.

ZADANIE

Zmodyfikuj kod klasy **MyAdapter**.

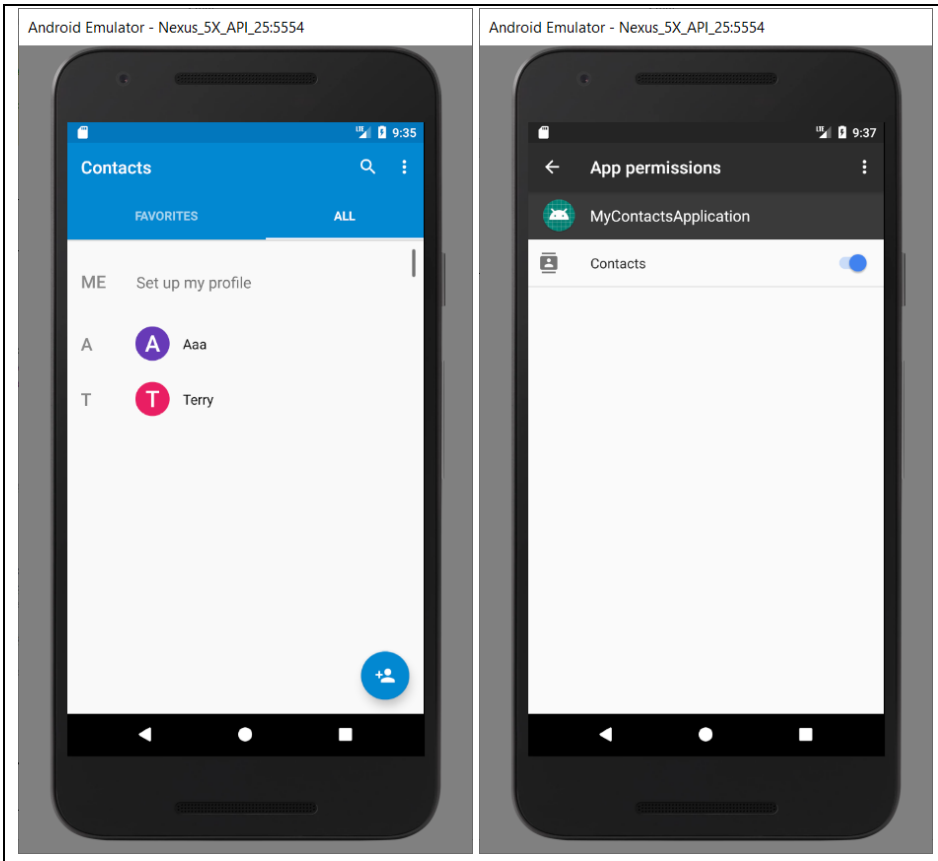
```
class MyAdapter: RecyclerView.Adapter<MyViewHlder>(){
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyViewHlder {
        val layoutInflater = LayoutInflater.from(parent.context)
        val contactRow = layoutInflater.inflate(R.layout.contact_row, parent, attachToRoot: false)
        return MyViewHlder(contactRow)
    }

    // ile layoutow mamy utworzyc
    override fun getItemCount(): Int {
        //return ContactDataBase.contactList.size // zwroc rozmiar listy kontaktow
        return MainActivity.listaKontaktow.size
    }

    // metoda aktualizujaca nasze widoki
    override fun onBindViewHolder(holder: MyViewHlder, position: Int) {
        val name = holder.itemView.contact_name // TextView przechowujace nazwe kontaktu
        val number = holder.itemView.contact_number // TextView z numerem kontaktu

        // pobieranie danych z bazy kontaktow i ustawienie jako tekstu w TextView
        //name.setText(ContactDataBase.contactList[position])
        //number.setText(ContactDataBase.contactNumberList[position])
        name.setText(MainActivity.listaKontaktow[position])
    }
}
```

Korzystając z emulatora, utwórz kilka kontaktów w telefonie. Wejdź w ustawienia telefonu, dalej w aplikacje i zezwól naszej aplikacji na dostęp do kontaktów.



Uruchom aplikację i przetestuj jej działanie.

Nasza aplikacja działa poprawnie:

Mamy skopiowane nazwy kontaktów, potrzebujemy jeszcze numerów telefonów.

ZADANIE

Dodaj do **MainActivity** ArrayListę do przechowywania numerów telefonów.

```
companion object {
    var listaKontaktow = arrayListOf<String>()
    var listaNumerow = arrayListOf<String>()
}
```

Przejdź do pętli odczytującej nazwy kontaktów i dopisz zaznaczoną linię kodu.

```
while(!cursor.isAfterLast){ // dopoki nie wskazuje na pozycje po ostatnim rekordzie ...
    // pobranie id kontaktu
    var id = cursor.getString(cursor.getColumnIndexOrThrow(ContactsContract.Contacts._ID))
    // pobranie nazwy kontaktu i dodanie do listy
    val name = cursor.getString(cursor.getColumnIndexOrThrow(ContactsContract.Contacts.DISPLAY_NAME_PRIMARY))
    listaKontaktow.add(name)
    // przejscie do kolejnego rekordu
    cursor.moveToNext()
}
```

Dodaj nową funkcję do **MainActivity**.

```
fun readPhoneNumber(resolver: ContentResolver, id: String){
    val cursorPhone = resolver.query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
        projection: null, selection: ContactsContract.CommonDataKinds.Phone.CONTACT_ID + "=?", arrayOf(id), sortOrder: null)

    if(cursorPhone.count > 0) {
        var numbers = "" // String do zbierania numerow danego kontaktu
        while (cursorPhone.moveToNext()){ // sprawdz czy jest kolejny nr tel i jesli tak, to przejdz do niego
            val phoneNumValue = cursorPhone.getString(
                cursorPhone.getColumnIndex(ContactsContract.CommonDataKinds.Phone.NUMBER))
            numbers += phoneNumValue + "; "
        }
        listaNumerow.add(numbers)
    }
    else
        listaNumerow.add("Brak numeru")
    cursorPhone.close()
}
```

Wywołaj utworzoną funkcję w pętli while poniżej kodu do odczytu nazw kontaktów.

```
while(!cursor.isAfterLast){ // dopoki nie wskazuje na pozycje po ostatnim rekordzie ...
    // pobranie id kontaktu
    val id = cursor.getString(cursor.getColumnIndexOrThrow(ContactsContract.Contacts._ID))
    // pobranie nazwy kontaktu i dodanie do listy
    val name = cursor.getString(cursor.getColumnIndexOrThrow(ContactsContract.Contacts.DISPLAY_NAME_PRIMARY))
    listaKontaktow.add(name)
    // odczyt numerow telefonu danego kontaktu
    readPhoneNumber(contentResolver, id)
    // przejscie do kolejnego rekordu
    cursor.moveToNext()
}
```

Omówmy ważniejsze kwestie funkcji **readPhoneNumber()**. W utworzonym kursorze klauzula **selection** działa jak warunek **WHERE** w MySQL. Ustalamy tutaj aby pobrane zostały tylko te rekordy, które zawierają **CONTACT_ID = ?**. Zaś w miejsce znaku zapytania wędruje parametr **arrayOf(id)**. Tym samym otrzymujemy tylko te numery telefonów, które są przypisane do id danego kontaktu. Zakładamy, że dany kontakt może posiadać kilka numerów.

Dalej sprawdzamy czy kursor zawiera jakieś rekordy, czyli czy w ogóle dany kontakt posiada przypisany numer telefonu. Jeśli tak, to dodajemy numery danego kontaktu do zmiennej **numbers**, a

później wartość tej zmiennej do ArrayListy numerów. Jeśli kontakt nie posiada numeru to jako numer ustawiamy komentarz „Brak numeru”.

Przechodzimy teraz do **MyAdapter** i wprowadzamy zmiany aby móc wyświetlać numery telefonów dla kontaktów.

ZADANIE

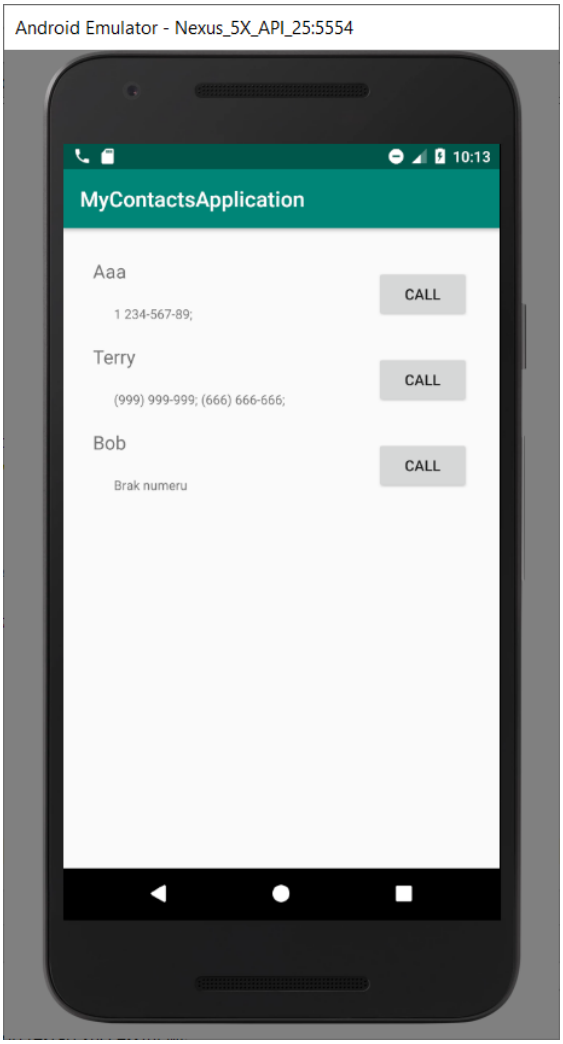
Wprowadź poniższe zmiany w **MyAdapter**.

```
override fun onBindViewHolder(holder: MyViewHlder, position: Int) {
    val name = holder.itemView.contact_name // TextView przechowujace nazwe kontaktu
    val number = holder.itemView.contact_number // TextView z numerem kontaktu

    // pobieranie danych z bazy kontaktow i ustawienie jako tekstu w TextView
    //name.setText(ContactDataBase.contactList[position])
    //number.setText(ContactDataBase.contactNumberList[position])
    name.setText(MainActivity.listaKontaktow[position])
    number.setText(MainActivity.listaNumerow[position])
}
```

Sprawdź działanie aplikacji.

Aplikacja działa poprawnie, wyświetlane są wszystkie kontakty i ich numery telefonów.



Wykonywanie połączeń

W tej sekcji zajmiemy się dodaniem do naszej aplikacji możliwości wykonywania połączeń. Zacznijmy od dodania uprawnień naszej aplikacji do wykonywania połączeń.

ZADANIE

Wprowadź poniższe zmiany w pliku manifestu.

```
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission.WRITE_CONTACTS"/>
<uses-permission android:name="android.permission.CALL_PHONE"/>
```

Teraz zajmijmy się przyciskiem **CALL**. Musimy ustawić dla niego listenera, który wywoła intencję wybierania numeru i/lub wykonywania połączenia.

ZADANIE

Wprowadź poniższe zmiany w **MyAdapter**.

```
override fun onBindViewHolder(holder: MyViewHlder, position: Int) {
    val name = holder.itemView.contact_name // TextView przechowujace nazwe kontaktu
    val number = holder.itemView.contact_number // TextView z numerem kontaktu
    val callBT = holder.itemView.call_BT

    // pobieranie danych z bazy kontaktow i ustawienie jako tekstu w TextView
    //name.setText(ContactDataBase.contactList[position])
    //number.setText(ContactDataBase.contactNumberList[position])
    name.setText(MainActivity.listaKontaktow[position])
    number.setText(MainActivity.listaNumerow[position])

    // obsluga przycisku CALL
    callBT.setOnClickListener { it: View!
        // utworzenie intencji wybierania numeru
        var intent = Intent()
                            // obowiazkowo 'tel:' aby Android zrozumial, ze chodzi o nr tel
        intent.data = Uri.parse( uriString: "tel:" + MainActivity.listaNumerow[position])
        //intent.action = Intent.ACTION_CALL // wykonywanie polaczenia od razu
        intent.action = Intent.ACTION_DIAL // wywolanie "wybierania numerow" z podanym numerem
        startActivity(holder.itemView.context, intent, options: null)
    }
}
```

Przetestuj działanie aplikacji zarówno dla **ACTION_CALL** jak i **ACTION_DIAL**. Pamiętaj o włączeniu uprawnień do wykonywania połączeń dla naszej aplikacji w ustawieniach telefonu.

Zróbmy jeszcze małe zabezpieczenia aby w przypadku kontaktów bez numerów akcja wykonywania połączenia nie uruchamiała się. Zamiast tego pojawiał się będzie stosowny komunikat.

ZADANIE

Zmodyfikuj kod obsługi przycisku.

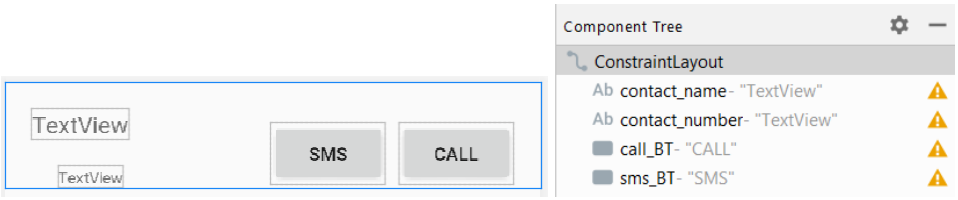
```
// obsluga przycisku CALL
callBT.setOnClickListener { it: View!
    // utworzenie intencji wybierania numeru
    var intent = Intent()
    if(MainActivity.listaNumerow[position]=="Brak numeru"){
        Toast.makeText(holder.itemView.context,  text: "Nieprawidłowy numer lub brak numeru!", Toast.LENGTH_SHORT)
            .show()
    }
    else{
                            // obowiązkowo 'tel:' aby Android zrozumiał, że chodzi o nr tel
        intent.data = Uri.parse( uriString: "tel:" + MainActivity.listaNumerow[position])
        //intent.action = Intent.ACTION_CALL // wykonywanie połączenia od razu
        intent.action = Intent.ACTION_DIAL // wywołanie "wybierania numerow" z podanym numerem
        startActivity(holder.itemView.context, intent,  options: null)
    }
}
```

Wysyłanie wiadomości SMS

Mamy już możliwość wykonywania połączeń, pora na wiadomości SMS. Na początek dodajmy do layoutu kontaktu przycisk do SMS-owania.

ZADANIE

Dodaj nowy przycisk do layoutu kontaktu i nazwij go **sms_BT**. Umieść tekst „SMS” na przycisku.



Przechodzimy do oprogramowania akcji przycisku **sms_BT**.

ZADANIE

Wprowadź zmiany w **MyAdapter** / **onBindViewHolder()**.

```
override fun onBindViewHolder(holder: MyViewHlder, position: Int) {
    val name = holder.itemView.contact_name // TextView przechowujace nazwe kontaktu
    val number = holder.itemView.contact_number // TextView z numerem kontaktu
    val callBT = holder.itemView.call_BT
    val smsBT = holder.itemView.sms_BT

    // pobieranie danych z bazy kontaktow i ustawienie jako tekstu w TextView
    //name.setText(ContactDataBase.contactList[position])
    //number.setText(ContactDataBase.contactNumberList[position])
    name.setText(MainActivity.listaKontaktow[position])
    number.setText(MainActivity.listaNumerow[position])

    // obsluga przycisku SMS
    smsBT.setOnClickListener { it: View!
        var sms_intent = Intent()
        if(MainActivity.listaNumerow[position]=="Brak numeru"){
            Toast.makeText(holder.itemView.context,  text: "Nieprawidłowy numer lub brak numeru!", Toast.LENGTH_SHORT)
                .show()
        }
        else{
                                    // obowiązkowo 'sms:'
            sms_intent.data = Uri.parse( uriString: "sms:" + MainActivity.listaNumerow[position])
            sms_intent.action = Intent.ACTION_VIEW // wywołanie aplikacji do smsowania
            startActivity(holder.itemView.context, sms_intent,  options: null)
        }
    }

    // obsluga przycisku CALL
    callBT.setOnClickListener { it: View!
```

Jak łatwo zauważyć, kod przycisku **SMS** jest bardzo podobny do kodu przycisku **CALL**. Właściwie jedyną różnicą jest druga linijka w **else**, gdzie wywołujemy intencję SMS.

Nasza aplikacja posiada już możliwość wyświetlania kontaktów, wykonywania połączeń i SMS-owania. Zauważmy jednak, że w przypadku dłuższych nazw kontaktów nasza aplikacja wygląda nieelegancko.

Wprowadźmy zmianę polegającą na skróceniu nazwy kontaktu.

ZADANIE

Wprowadź zmiany w **MyAdapter** / **onBindViewHolder()**.

```
override fun onBindViewHolder(holder: MyViewHlder, position: Int) {
    val name = holder.itemView.contact_name // TextView przechowujace nazwe kontaktu
    val number = holder.itemView.contact_number // TextView z numerem kontaktu
    val callBT = holder.itemView.call_BT
    val smsBT = holder.itemView.sms_BT

    if(MainActivity.listaKontaktow[holder.adapterPosition].length > 13){
        name.setText("")
        var x=0
        while(x<13){
            var znak = MainActivity.listaKontaktow[holder.adapterPosition].get(x)
            if (x>=10)
                name.append(".")
            else
                name.append(znak.toString())
            x++
        }
    }
    else
        name.setText(MainActivity.listaKontaktow[holder.adapterPosition])
    number.setText(MainActivity.listaNumerow[holder.adapterPosition])
```

Sprawdź działanie aplikacji.

Zaimplementowany kod sprawdza czy nazwa kontaktu jest dłuższa niż 13 znaków i jeśli tak, to skraca ją do 10 znaków oraz dodaje trzy kropki na końcu. Jeśli natomiast nazwa kontaktu jest krótsza, to wyświetla ją w całości.