Kotlin & Android w Android Studio Giraffe (2022.3.1)

Cykl życia aktywności. Formularz logowania

Cykl życia aktywności

Aktywności w systemie Android mają określony cykl życia. Na stronie dokumentacji Androida znaleźć możemy poniższy diagram:

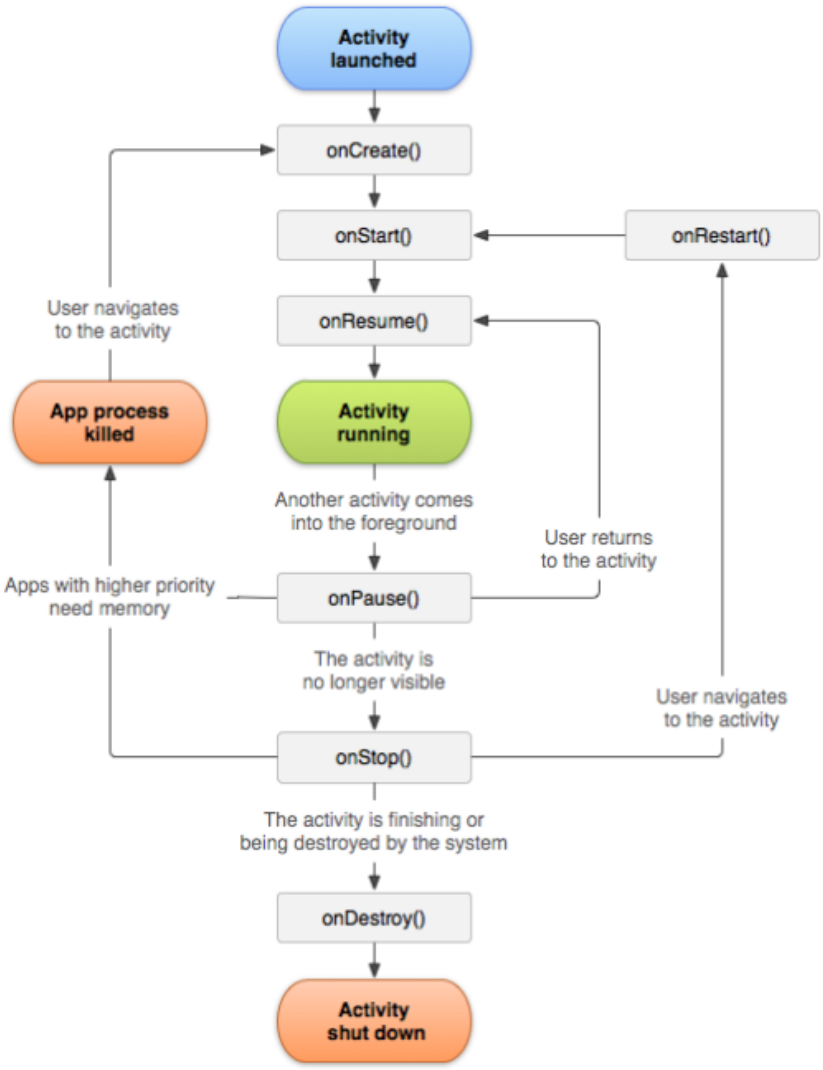


Figure 1. A simplified illustration of the activity lifecycle.

Metoda **onCreate()** została utworzona domyślnie przy tworzeniu nowego projektu w Android Studio. Metoda ta wywoływana jest jednokrotnie przy uruchamianiu aplikacji androidowej. Odpowiada za przygotowanie do startu aplikacji. Tutaj zamieszczamy np. kod odpowiedzialny za ustawienia layoutu.

Metoda **onStart()** odpowiada za właściwe uruchomienie aplikacji.

Metoda **onResume()** odpowiada za wznowienie aplikacji po jej zapauzowaniu.

Metoda **onPause()** odpowiada za zapauzowanie aplikacji („zapamiętanie” aktualnego jej stanu) w sytuacji przesłonięcia jej przez inną aplikację. Wówczas nasza aplikacja „czeka” aż do niej powrócimy.

Metoda **onStop()** odpowiada za zatrzymanie aplikacji (jej „zminimalizowanie”). Aktywność nie zostaje tutaj jeszcze zniszczona ale znajduje się w ostatnich aplikacjach i możemy do niej wrócić poprzez metodę **onRestart()**.

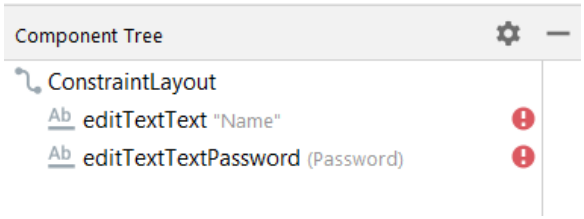
Metoda **onDestroy()** „niszczy” naszą aktywność i kończy jej działanie zwalniając przydzieloną pamięć.

Ponadto aplikacja androidowa może zostać zniszczona po jej zapauzowaniu. Taka sytuacja może się zdarzyć np. gdy mamy mocno obciążony telefon i nasza aktywność musi zwolnić pamięć dla innej aktywności działającej na pierwszym planie.

Tworzenie formularza

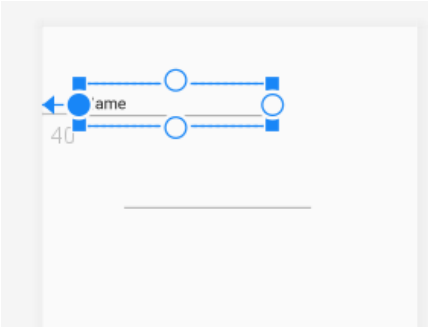
ZADANIE

Korzystając z projektu z poprzedniego laboratorium zmodyfikuj wygląd drugiej aktywności (*activity_second.xml*) dodając do *ConstraintLayout* komponenty *PlainText* i *Password*.

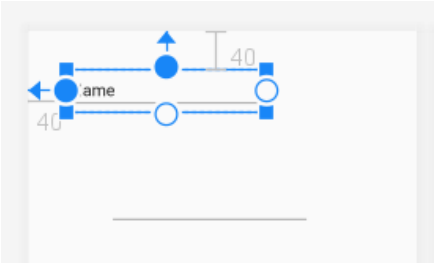


Przy dodanych kontrolkach widnieją ikony błędów mówiące o tym, że w momencie uruchomienia aplikacji wskazane komponenty „uciekną” w położenie (0,0). Aby temu zapobiec musimy określić odległości naszych komponentów od innych.

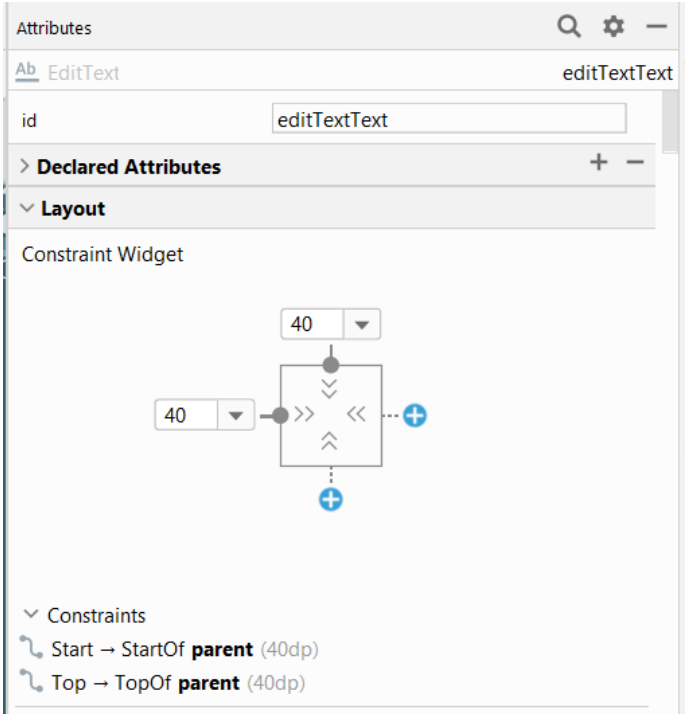
W tym celu „łapiemy” np. za lewą krawędź naszej kontrolki *PlainText* i ustalamy jej odległość od lewej krawędzi layoutu.



Podobnie musimy uczynić dla górnej (lub dolnej) części kontrolki.



Ustawienia odległości możemy dokładnie sprecyzować w sekcji *Attributes* naszego komponentu:



ZADANIE

Ustaw dodane kontrolki w odległości 20 jednostek od lewej krawędzi layoutu. Ponadto *PlainText* ustaw 20 jednostek poniżej górnej krawędzi layoutu, zaś *Password* w odległości 20 jednostek poniżej *PlainText*.

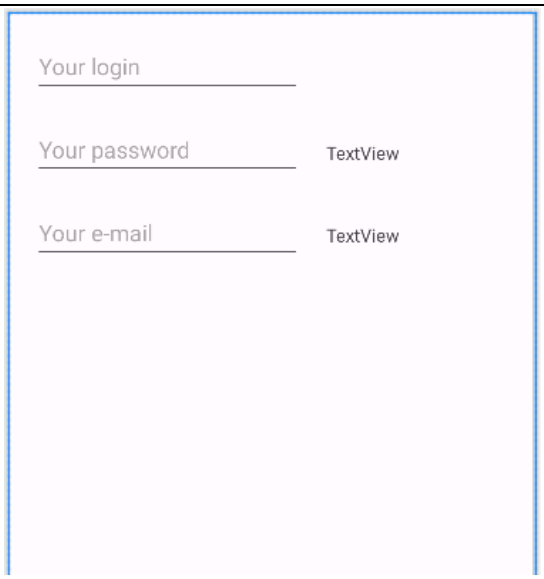
Dalej dla *PlainText* usuń tekst „Name” i ustaw *tekst hint* na „Your login” i dla *Password* na „Your password”.

Tekst *hint* widoczny jest w momencie, gdy nasza kontrolka jest nieaktywna. W sytuacji kliknięcia na nią i rozpoczęcia wprowadzania tekstu domyślny tekst znika. Za pomocą pola *hint* możemy powiedzieć użytkownikowi co powinien zrobić z daną kontrolką, np. wpisać hasło.

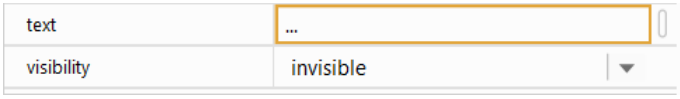
ZADANIE

Dodaj komponent *E-mail* poniżej *Password* zachowując odległości 20 jednostek. Ustaw *hint* dla pola *E-mail* na „Your e-mail”.

Ponadto po prawej stronie *Password* i *E-mail* dodaj kontrolki *TextView*.

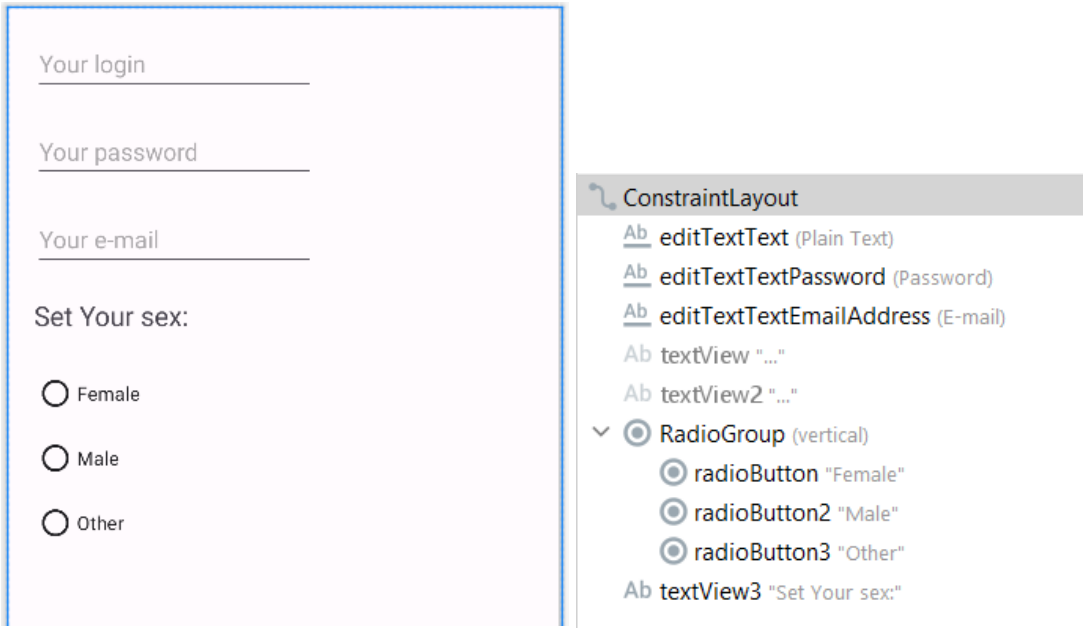


Dla obydwu *TextView* ustaw tekst „…” oraz *visibility* na „invisible”.

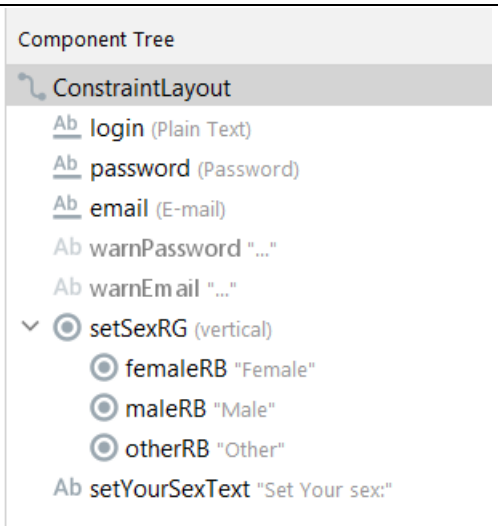


Dalej dodaj do layoutu kontrolkę *RadioGroup*, a do niej 3 przyciski *RadioButton*. Ustaw przyciski w odpowiedniej pozycji i ustaw na nich tekst wg. poniższego rysunku.

Powyżej przycisków radio dodaj *TextView* z napisem „Set your sex:”. Ustaw rozmiar tekstu dla tego komponentu na 20sp.



Zmień identyfikatory kontrolek w następujący sposób:



Uruchom aplikację i sprawdź jej działanie.

Wykorzystanie metody onUserInteraction()

Zarys naszego formularza jest już gotowy. Przystępujemy do jego oprogramowania.

Zaczynamy od pola do wprowadzania hasła. Ustalmy, że dobre hasło powinno mieć co najmniej 8 znaków. Ponadto nasz formularz powinien być niejako cały czas aktualizowany ponieważ np. po wprowadzeniu zbyt krótkiego hasła powinniśmy zobaczyć komunikat o niedostatecznej jego długości, ale po wpisaniu 8 znaków komunikat dotyczący hasła powinien zniknąć. Dlatego nasz kod będziemy tworzyć w specjalnej metodzie *onUserInteraction()*. Metod ta będzie wywoływana za każdym razem gdy użytkownik wykona jakąś interakcję z naszą aktywnością.

ZADANIE

Zaimplementuj poniższą metodę w *SecondActivity*.

```
override fun onUserInteraction() {
    super.onUserInteraction()

    val password = findViewById<EditText>(R.id.password)
    val warnPassword = findViewById<TextView>(R.id.warnPassword)

    if( password.isFocused){ // jesli pole password "ma uwage"
        if(password.length()<8) { // dlugosc hasla < 8 znakow
            warnPassword.setText("Hasło za krótkie!") // tekst ostrzeżenia
            warnPassword.visibility = TextView.VISIBLE // ustawienie widoczności ostrzeżenia
        }
        else{ // dlugosc hasla >= 6 znakow
            warnPassword.setText("Super hasło!") // tekst ostrzeżenia
            warnPassword.visibility = TextView.VISIBLE // ustawienie widoczności ostrzeżenia
        }
    }
    else {
        warnPassword.visibility = TextView.INVISIBLE // ustawienie braku widoczności ostrzeżenia
    }
}
```

Przetestuj działanie pola *Password* w naszym formularzu.

Przystępujemy do oprogramowania pola e-mail. Załóżmy, że poprawny e-mail powinien zawierać znak „@”.

ZADANIE

Do metody *onUserInteraction()* dopisz następujący kod:

```
val email = findViewById<EditText>(R.id.email)
val warnEmail = findViewById<TextView>(R.id.warnEmail)

if(email.isFocused){ // jesli pole email "ma uwage"
    warnEmail.setText("Niepoprawny e-mail!")
    for(i in email.text){ // przesledź znaki zawarte w e-mailu
        if(i == '@') {
            warnEmail.setText("Poprawny e-mail!")
            warnEmail.visibility = TextView.VISIBLE // ustawienie widoczności ostrzeżenia
        }
        else{
            warnEmail.visibility = TextView.VISIBLE // ustawienie widoczności ostrzeżenia
        }
    }
}
else{
    warnEmail.visibility = TextView.INVISIBLE // ustawienie braku widoczności ostrzeżenia
}
```

Przetestuj działanie pola *E-mail*.

Nasz formularz posiada pewien błąd działania. Gdy wpisujemy jakiś ciąg znaków w polu hasła pojawiał się komunikat „Hasło za krótkie!”. Po jednokrotnym kliknięciu na inną kontrolkę, np. pole email – komunikat o zbyt krótkim haśle pozostaje, zaś w momencie wpisywania znaków e-maila komunikat dotyczący hasła znika.

Wykorzystanie metody onTextChanged() oraz afterTextChanged()

Do bardziej profesjonalnego oprogramowania naszego formularza użyjemy metody *onTextChange()*, która sprawdza na bieżąco długość znaków oraz metody *afterTextChange()*.

ZADANIE

Zmodyfikuj zawartość metody *onCreate()* w *SecondActivity*:

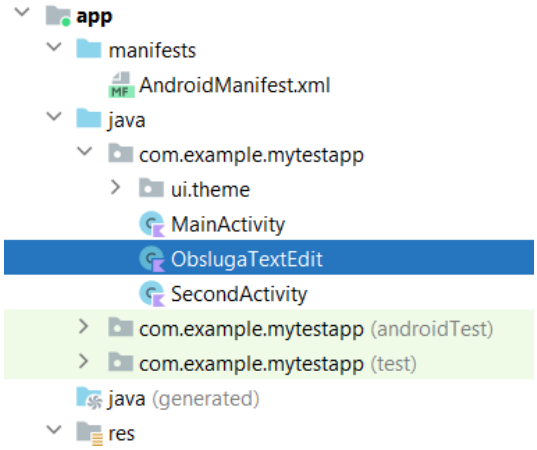
```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_second)

    val password = findViewById<EditText>(R.id.password)
    val warnPassword = findViewById<TextView>(R.id.warnPassword)
    // dodanie "sluchacza" do pola password
    password.addTextChangedListener(object : TextWatcher { // wykorzystujemy interfejs TextWatcher
        override fun beforeTextChanged(p0: CharSequence?, p1: Int, p2: Int, p3: Int) {

        }
        override fun onTextChanged(p0: CharSequence?, p1: Int, p2: Int, p3: Int) {
            if(password.length()<=6) { // dlugosc hasla <= 6 znakow
                warnPassword.setText("Hasło za krótkie!") // tekst ostrzeżenia
                warnPassword.visibility = TextView.VISIBLE // ustawienie widoczności ostrzeżenia
            }
            else{ // dlugosc hasla > 6 znakow
                warnPassword.setText("Super hasło!") // tekst ostrzeżenia
                warnPassword.visibility = TextView.VISIBLE // ustawienie widoczności ostrzeżenia
            }
        }
        override fun afterTextChanged(p0: Editable?) {
            if(password.length()==0)
                warnPassword.visibility = TextView.INVISIBLE // ustawienie braku widoczności ostrzeżenia
        }
    })

    val email = findViewById<EditText>(R.id.email)
    val warnEmail = findViewById<TextView>(R.id.warnEmail)
    // dodanie "sluchacza" do pola email
    email.addTextChangedListener(object : TextWatcher{ // wykorzystujemy interfejs TextWatcher
        override fun beforeTextChanged(p0: CharSequence?, p1: Int, p2: Int, p3: Int) {

        }
        override fun onTextChanged(p0: CharSequence?, p1: Int, p2: Int, p3: Int) {
            warnEmail.setText("Niepoprawny e-mail!")
            for(i in email.text){ // przesledz znaki zawarte w e-mailu
                if(i == '@') {
                    warnEmail.setText("Poprawny e-mail!")
                    warnEmail.visibility = TextView.VISIBLE // ustawienie widoczności ostrzeżenia
                }
                else{
                    warnEmail.visibility = TextView.VISIBLE // ustawienie widoczności ostrzeżenia
                }
            }
        }
        override fun afterTextChanged(p0: Editable?) {
            if(email.length()==0)
                warnEmail.visibility = TextView.INVISIBLE // ustawienie braku widoczności ostrzeżenia
        }
    })
}
```

Zakomentuj kod metody *onUserInteraction()*. Sprawdź działanie pól *password* i *email*.

Nasza aplikacja działa lepiej, jednak długi kod w *onCreate()* wygląda nieelegancko. Uporządkujmy nieco nasza aplikację.

ZADANIE

Utwórz nową klasę Kotlin w naszej aplikacji i nazwij ją *ObslugaTextEdit*:



W nowo utworzonym pliku tworzymy funkcję odpowiedzialną za obsługę hasła:

```
class ObslugaTextEdit {
    fun obslugaEditPassword(password: EditText, warnPassword: TextView){

    }
}
```

Jako zawartość metody ustaw kod z metody *onCreate()* z *SecondActivity*, który wcześniej odpowiadał za obsługę hasła:

```
fun obslugaEditPassword(password: EditText, warnPassword: TextView){
    // dodanie "sluchacza" do pola password
    password.addTextChangedListener(object : TextWatcher { // wykorzystujemy interfejs TextWatcher
        override fun beforeTextChanged(p0: CharSequence?, p1: Int, p2: Int, p3: Int) {

        }
        override fun onTextChanged(p0: CharSequence?, p1: Int, p2: Int, p3: Int) {
            if(password.length()<=6) { // dlugosc hasla <= 6 znakow
                warnPassword.setText("Hasło za krótkie!") // tekst ostrzeżenia
                warnPassword.visibility = TextView.VISIBLE // ustawienie widoczności ostrzeżenia
            }
            else{ // dlugosc hasla > 6 znakow
                warnPassword.setText("Super hasło!") // tekst ostrzeżenia
                warnPassword.visibility = TextView.VISIBLE // ustawienie widoczności ostrzeżenia
            }
        }
        override fun afterTextChanged(p0: Editable?) {
            if(password.length()==0)
                warnPassword.visibility = TextView.INVISIBLE // ustawienie braku widoczności ostrzeżenia
        }
    })
}
```

Podobnie utwórz metodę do obsługi pola email:

```
fun obslugaEditEmail(email: EditText, warnEmail: TextView){
    // dodanie "sluchacza" do pola email
    email.addTextChangedListener(object : TextWatcher{ // wykorzystujemy interfejs TextWatcher
        override fun beforeTextChanged(p0: CharSequence?, p1: Int, p2: Int, p3: Int) {

        }
        override fun onTextChanged(p0: CharSequence?, p1: Int, p2: Int, p3: Int) {
            warnEmail.setText("Niepoprawny e-mail!")
            for(i in email.text){ // przesledz znaki zawarte w e-mailu
                if(i == '@') {
                    warnEmail.setText("Poprawny e-mail!")
                    warnEmail.visibility = TextView.VISIBLE // ustawienie widoczności ostrzeżenia
                }
                else{
                    warnEmail.visibility = TextView.VISIBLE // ustawienie widoczności ostrzeżenia
                }
            }
        }
        override fun afterTextChanged(p0: Editable?) {
            if(email.length()==0)
                warnEmail.visibility = TextView.INVISIBLE // ustawienie braku widoczności ostrzeżenia
        }
    })
}
```

Pozostaje wywołać nasze metody w *onCreate()* w *SecondActivity*:

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

    val obslugaTextEdit = ObslugaTextEdit()

    val password = findViewById<EditText>(R.id.password)
    val warnPassword = findViewById<TextView>(R.id.warnPassword)
    obslugaTextEdit.obslugaEditPassword(password, warnPassword)

    val email = findViewById<EditText>(R.id.email)
    val warnEmail = findViewById<TextView>(R.id.warnEmail)
    obslugaTextEdit.obslugaEditEmail(email, warnEmail)
}
```

Sprawdź działanie aplikacji.

Nasza aplikacja działa poprawnie, a struktura projektu i kod są bardziej przejrzyste.

Obsługa RadioButtonów, metoda setOnCheckedChangeListener()

W naszej aktywności potrzebujemy jeszcze oprogramować przyciski *RadioButton*. Przyciski te są zgrupowane w *RadioGroup*, dzięki czemu są ze sobą niejako powiązane, tzn. możemy zaznaczyć tylko jeden z przycisków – i o to nam chodzi.

Dodamy teraz „słuchacza zmiany wyboru” *setOnCheckedChangeListener()* do naszej grupy przycisków, aby móc obsłużyć akcję zmiany zaznaczenia w grupie.

ZADANIE

W metodzie *onCreate()* w *SecondActivity* dodaj kod odpowiedzialny za obsługę przycisków radio.

```
val setSexRG = findViewById<RadioGroup>(R.id.setSexRG)
setSexRG.setOnCheckedChangeListener{group, checkedId->
    run{ this: SecondActivity

    }
}
```

W powyższym kodzie *group* oznacza naszą grupę, na rzecz której tworzymy listenera, zaś *checkedId* jest identyfikatorem zaznaczonego przycisku radio. Wewnątrz *run* umieścimy kod akcji, która ma się wykonać po dokonaniu wyboru przycisku radio.

Naszym celem jest wyświetlenie w *MainActivity* tekstu odpowiadającego płci, która została wybrana w *SecondActivity*. W tym celu musimy utworzyć nową intencję i poinformować niejako *MainActivity*, że w aktywności *SecondActivity* wybraliśmy dany przycisk radio. Posłużą nam tutaj tzw. *extras*.

ZADANIE

Uzupełnij kod odpowiedzialny za obsługę przycisków radio.

```
val setSexRG = findViewById<RadioGroup>(R.id.setSexRG)
setSexRG.setOnCheckedChangeListener{group, checkedId->
    run{ this: SecondActivity
        val intencjaAktywujaca: Intent = Intent(applicationContext, MainActivity::class.java)
        var RB: RadioButton = findViewById(checkedId) // pobranie id zaznaczonego RadioButtona

        // wyslanie extra do MainActivity, wysylamy pare: id=plec, text=tekst{_zaznaczonego_przycisku_radio
        intencjaAktywujaca.putExtra( name: "plec", RB.text)
        startActivity(intencjaAktywujaca) // startujemy nasza intencje
    }
}
```

Pozostaje nam teraz zmodyfikować wygląd i kod *MainActivity*.

ZADANIE

Dodaj do głównej aktywności *TextView* poniżej wszystkich kontrolek. Ustaw szerokość na *wrap_content*, margines górny na 40dp, rozmiar czcionki na 18sp, usuń domyślny tekst kontrolki. Ustaw identyfikator kontrolki na *plec_TextView*.

Przechodzimy do *MainActivity*. Nasz kod osadzimy w metodzie *onResume()*.

ZADANIE

Dodaj do *MainActivity* treść metody *onResume()*.

```
override fun onResume() {
    super.onResume()

    val plec_TextView = findViewById<TextView>(R.id.plec_TextView)
    // sprawdzenie czy intencja uruchamiajaca MainActivity ma dodatek Extras o kluczu=plec
    if(intent.hasExtra( name: "plec")){
        // ustawienie tekstu pobranego z Extras
        plec_TextView.setText(intent.getCharSequenceExtra( name: "plec"))
    }
}
```

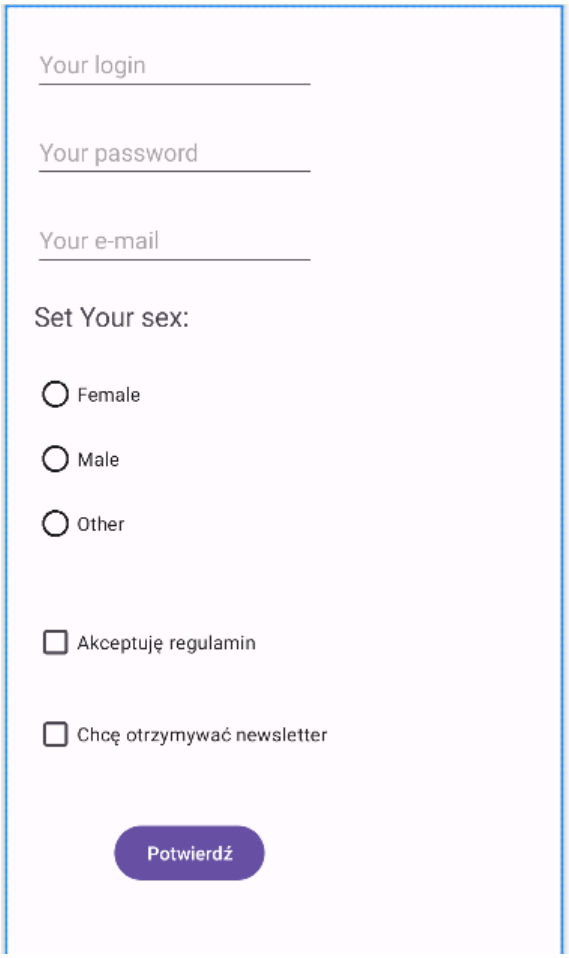
Przetestuj działanie aplikacji.

Obsługa CheckBoxów

Do naszego formularza dodamy jeszcze dwa pola wielokrotnego wyboru – *CheckBox* i jeden przycisk *Button*.

ZADANIE

Dodaj do formularza w drugiej aktywności 2 *CheckBoxy* oraz *Button*. Zmień domyślne teksty dodanych kontrolek wg rysunku:



Ustaw identyfikatory dla dodanych kontrolek:

- pole akceptacji regulaminu na „regulations_CHB”,

- pole newslettera na „newsletter_CHB”,

- przycisk na „submit_BT”.

Docelowo chcemy, aby po wypełnieniu formularza i akceptacji regulaminu dane w nim zawarte zostały przekazane do nowej aktywności. W przypadku braku akceptacji regulaminu i kliknięciu na przycisk *Potwierdź* powinno zostać wyświetlone tylko powiadomienie o konieczności akceptacji regulaminu.

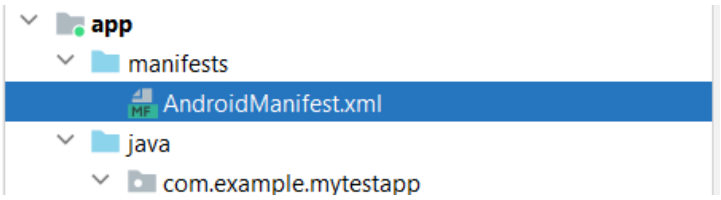
Dodatkowo chcemy, aby *SecondActivity* była aktywnością główną, tzn. aby była widoczna po uruchomieniu aplikacji. Dlatego musimy dokonać stosownych zmian w pliku *AndroidManifest.xml*. W tym momencie jego zawartość wygląda następująco:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools">
4
5      <application
6          android:allowBackup="true"
7          android:dataExtractionRules="@xml/data_extraction_rules"
8          android:fullBackupContent="@xml/backup_rules"
9          android:icon="@mipmap/ic_launcher"
10         android:label="@string/app_name"
11         android:roundIcon="@mipmap/ic_launcher_round"
12         android:supportsRtl="true"
13         android:theme="@style/Theme.MyTestApp"
14         tools:targetApi="31">
15         <activity
16             android:name=".SecondActivity"
17             android:exported="false" />
18         <activity
19             android:name=".MainActivity"
20             android:exported="true"
21             android:label="MyTestApp"
22             android:theme="@style/Theme.MyTestApp">
23             <intent-filter>
24                 <action android:name="android.intent.action.MAIN" />
25                 <category android:name="android.intent.category.LAUNCHER" />
26             </intent-filter>
27         </activity>
28     </application>
29
30 </manifest>
```

W linii 10 mamy odwołanie do pliku *strings.xml* zawierającego m.in. wyświetlaną nazwę naszej aplikacji. W liniach 15 i 18 mamy informację o tym, że nasza aplikacja składa się z 2 aktywności. Linie 24-25 wskazują, że *MainActivity* jest aktywnością główną oraz że od wyświetlania tej aktywności będzie startować nasza aplikacja.

ZADANIE

Przejdź do edycji pliku *AndroidManifest.xml*



Zmodyfikuj go w następujący sposób:

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

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.MyTestApp"
        tools:targetApi="31">
        <activity
            android:name=".SecondActivity"
            android:exported="true" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".MainActivity"
            android:exported="false"
            android:label="MyTestApp"
            android:theme="@style/Theme.MyTestApp">
        </activity>
    </application>

</manifest>
```

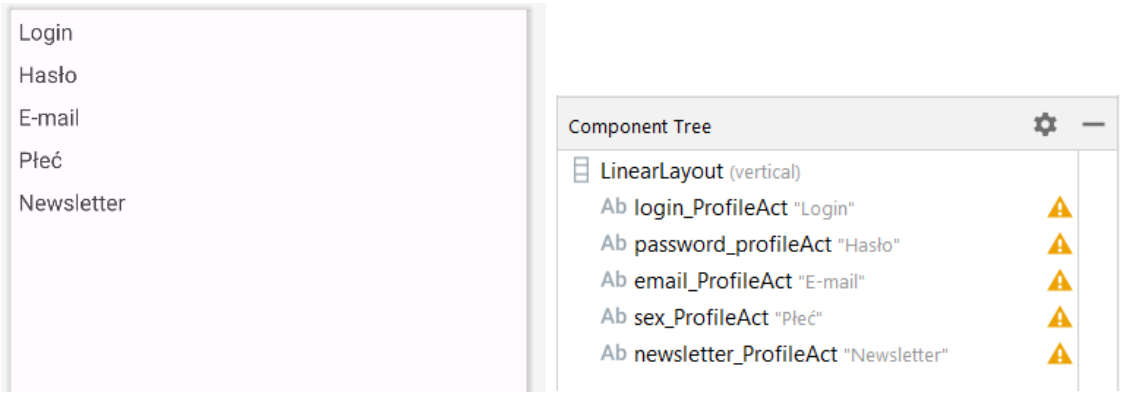
Uruchom aplikację i sprawdź co się zmieniło.

Po dokonanych zmianach i uruchomieniu aplikacji jako pierwsza aktywność startuje *SecondActivity*.

Teraz dodamy nową aktywność, która wyświetlać będzie dane przekazane z formularza.

ZADANIE

Utwórz w projekcie nową aktywność (Empty View Activity) i nazwij ją *ProfileActivity*. Przejdź do designera utworzonej aktywności i dodaj do niej *LinearLayout*, a w nim pięć kontrolek *TextView*. Ustaw położenie, teksty na kontrolkach i ich identyfikatory wg poniższych rysunków.



Teraz zajmiemy się oprogramowaniem kontrolek dodanych do formularza.

ZADANIE

Przejdź do pliku *SecondActivity*. Zakomentuj kod odpowiedzialny za akcję wyboru płci.

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

    val obslugaTextEdit = ObslugaTextEdit()

    val password = findViewById<EditText>(R.id.password)
    val warnPassword = findViewById<TextView>(R.id.warnPassword)
    obslugaTextEdit.obslugaEditPassword(password, warnPassword)

    val email = findViewById<EditText>(R.id.email)
    val warnEmail = findViewById<TextView>(R.id.warnEmail)
    obslugaTextEdit.obslugaEditEmail(email, warnEmail)

      val setSexRG = findViewById<RadioGroup>(R.id.setSexRG)
      setSexRG.setOnCheckedChangeListener{group, checkedId->
          run{
              val intencjaAktywujaca: Intent = Intent(applicationContext, MainActivity::class.java)
              var RB: RadioButton = findViewById(checkedId) // pobranie id zaznaczonego RadioButtona

              // wyslanie extra do MainActivity, wysylamy pare: id=plec, text=tekst_zaznaczonego_przycisku_radio
              intencjaAktywujaca.putExtra("plec", RB.text)
              startActivity(intencjaAktywujaca) // startujemy nasza intencje
          }
      }
}
```

Następnie utwórz nową funkcję:

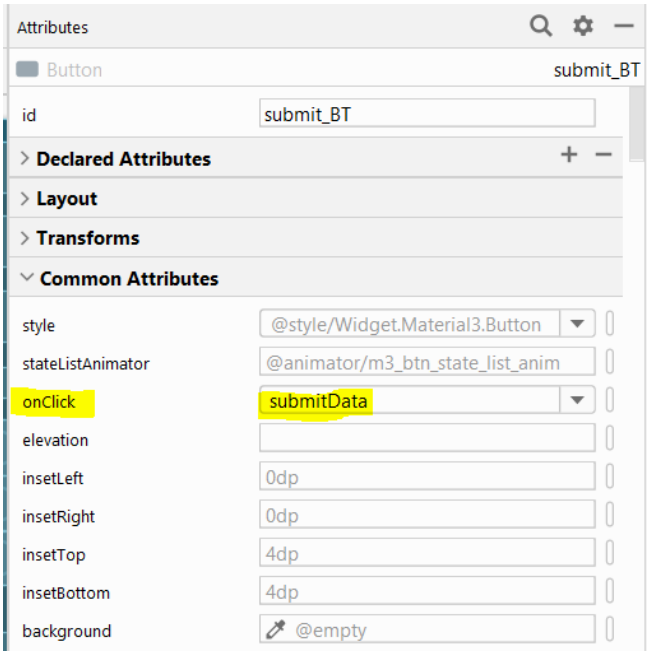
```
fun submitData(view: View){
    val regulation_CHB = findViewById<CheckBox>(R.id.regulation_CHB)
    if(!regulation_CHB.isChecked){ // jeśli regulamin jest niezaznaczony ...
        Toast.makeText(applicationContext, text: "Proszę potwierdzić regulamin!", Toast.LENGTH_SHORT)
            .show()
    }
    else{ // regulamin zaakceptowany
        val login = findViewById<EditText>(R.id.login)
        val password = findViewById<EditText>(R.id.password)
        val email = findViewById<EditText>(R.id.email)
        val setSexRG = findViewById<RadioGroup>(R.id.setSexRG)
        val newsletter_CHB = findViewById<CheckBox>(R.id.newsletter_CHB)
        val intent = Intent(applicationContext, ProfileActivity::class.java)
        intent.putExtra( name: "login", login.text)
        intent.putExtra( name: "password", password.text)
        intent.putExtra( name: "email", email.text)
        intent.putExtra( name: "plec", findViewById<RadioButton>(setSexRG.checkedRadioButtonId).text)
        if(newsletter_CHB.isChecked)
            intent.putExtra( name: "newsletter",  value: "TAK")
        else
            intent.putExtra( name: "newsletter",  value: "NIE")

        startActivity(intent)
    }
}
```

Mamy utworzoną funkcję, która przekaże dane z formularza do aktywności *ProfileActivity*. Trzeba ją teraz „podpiąć” do przycisku w formularzu. Można to zrobić na 2 sposoby:

- „ręcznie” okodowując przycisk w *activity_second.xml*:

```
<Button
    android:id="@+id/submit_BT"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="80dp"
    android:layout_marginTop="40dp"
    android:text="Potwierdź"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/newsletter_CHB"
    android:onClick="submitData"/>
```

- ustawiając odpowiednią funkcję w atrybutach przycisku w designerze:



ZADANIE

Ustaw funkcję *submitData()* jako akcję kliknięcia na przycisk *Prześlij*.

Pozostało nam teraz „odebrać” informacje przekazane w formularzu i wyświetlić je w *ProfileActivity*.

ZADANIE

Przejdź do *ProfileActivity* i dodaj kod odpowiedzialny za wyświetlenie informacji przekazanych z formularza *SecondActivity*.

```
override fun onResume() {
    super.onResume()
    val login_ProfileAct = findViewById<TextView>(R.id.login_ProfileAct)
    val password_profileAct = findViewById<TextView>(R.id.password_profileAct)
    val email_ProfileAct = findViewById<TextView>(R.id.email_ProfileAct)
    val sex_ProfileAct = findViewById<TextView>(R.id.sex_ProfileAct)
    val newsletter_ProfileAct = findViewById<TextView>(R.id.newsletter_ProfileAct)
    if(intent.hasExtra( name: "login"))
        login_ProfileAct.setText("Login: "+intent.getCharSequenceExtra( name: "login"))
    if(intent.hasExtra( name: "password"))
        password_profileAct.setText("Hasło: "+intent.getCharSequenceExtra( name: "password"))
    if(intent.hasExtra( name: "email"))
        email_ProfileAct.setText("E-mail: "+intent.getCharSequenceExtra( name: "email"))
    if(intent.hasExtra( name: "plec"))
        sex_ProfileAct.setText("Płeć: "+intent.getCharSequenceExtra( name: "plec"))
    if(intent.hasExtra( name: "newsletter"))
        newsletter_ProfileAct.setText("Newsletter: "+intent.getCharSequenceExtra( name: "newsletter"))
}
```

Uruchom aplikację i przetestuj jej działanie.

Nasz formularz jest gotowy.