Panel logowania cz.2 – inne podejście

Skorzystamy z panelu logowania z poprzedniego projektu (można też obsługę panelu logowania umieścić w komentarzu i kontynuować programowanie). Tworzę bliźniaczy projekt pod nazwą logowanie2. Wszystkie zmiany wprowadzone w plik cofamy – logowanieActivity.kt

activity_main.xml | MainActivity.kt | LogowanieActivity.kt

```
1   package com.example.logowanie2
2
3   import androidx.appcompat.app.AppCompatActivity
4   import android.os.Bundle
5   import android.widget.EditText
6   import android.widget.TextView
7
8   class LogowanieActivity : AppCompatActivity() {
9       override fun onCreate(savedInstanceState: Bundle?) {
10          super.onCreate(savedInstanceState)
11          setContentView(R.layout.activity_logowanie)
12      }
13      / override fun onUserInteraction(){//metoda interaction .../
47  }
```

Tworzymy nową obsługę. Zamiast '***onUserInteraction()***' użyjemy metody '***onTextChanged()***'. Reakcja na wprowadzane znaki będzie natychmiastowa, a nie po kliknięciu w inny element. Wprowadzamy metodę **`addTextChangedListener()`** – metoda ta nasłuchuje zdarzenie zmiany tekstu. Tworzymy referencję do tekstu hasła i obsługę zmiany tekstu za pomocą wspomnianej metody:

```
10  class LogowanieActivity : AppCompatActivity() {
11      override fun onCreate(savedInstanceState: Bundle?) {
12          super.onCreate(savedInstanceState)
13          setContentView(R.layout.activity_logowanie)
14          var haslo = findViewById<TextView>(R.id.poleHasla)
15          haslo.addTextChangedListener(object : TextWatcher{})
```

Podczas wpisywania w klamerkach może pojawić się chmurka informyjąca konieczność użycia trzech zdarzeń: Przed, po i w trakcie zmian tekstu)

```
watcher: TextWatcher!
beforeTextChanged: (text: CharSequence?, start: Int, count: Int, after: Int) -> Unit = { _, _, _, _ -> }, onTextChanged: (text: CharSequence?, start: Int, before: Int, count: Int) -> Unit = { _, _, _, _ -> },
afterTextChanged: (text: Editable?) -> Unit = {}
```

Dla nas najważniejsze jest zdarzenie w trakcie wpisywania. Jednak TextWatcher musi posiadać wszystkie trzy zdarzenia.

Aby nie wpisywać wszystkiego ręcznie można najechać myszą na napis TextWatcher, nacisnąć klawisz Ctrl (na MacBooku klawisz command) i kliknąć w napis. Pojawi się opis, z którego możemy skopiować zawartość i wkleić do naszego kodu.

object : TextWatcher

```
13      var haslo = findViewById<TextView>(R.id.poleHasla)
14      haslo.addTextChangedListener(object : TextWatcher{
15          override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {}
16          override fun afterTextChanged(s: Editable?){}
17          override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int){}
18      })
```

W linii 17 wewnątrz klamer umieszczamy kod obsługi zdarzenia. Nie zapominajmy o referencji do tekstu pod polem wprowadzania hasła oraz emaila.

```
10  class LogowanieActivity : AppCompatActivity() {
11      override fun onCreate(savedInstanceState: Bundle?) {
12          super.onCreate(savedInstanceState)
13          setContentView(R.layout.activity_logowanie)
14          var haslo = findViewById<EditText>(R.id.poleHasla)
15          val hasloInfo = findViewById<TextView>(R.id.infoHaslo)
16          var email = findViewById<EditText>(R.id.E_MAIL)
17          var infoEmail = findViewById<TextView>(R.id.infoEmail)
18          haslo.addTextChangedListener(object : TextWatcher{// nasłuch zmiany tekstu
19              override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {}
20              override fun afterTextChanged(s: Editable?){}
21              override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int){
22                  if(haslo.length() == 0){
23                      hasloInfo.setText("Wprowadź hasło")
24                      hasloInfo.visibility = TextView.VISIBLE
25                  } else if(haslo.length()<8){
26                      hasloInfo.setText("Twoje hasło jest za krótkie")
27                      hasloInfo.visibility = TextView.VISIBLE
28                  } else {
29                      hasloInfo.setText("Hasło prawidłowe")
30                      hasloInfo.visibility = TextView.VISIBLE
31                  }
32              }
33          })
34      }
```

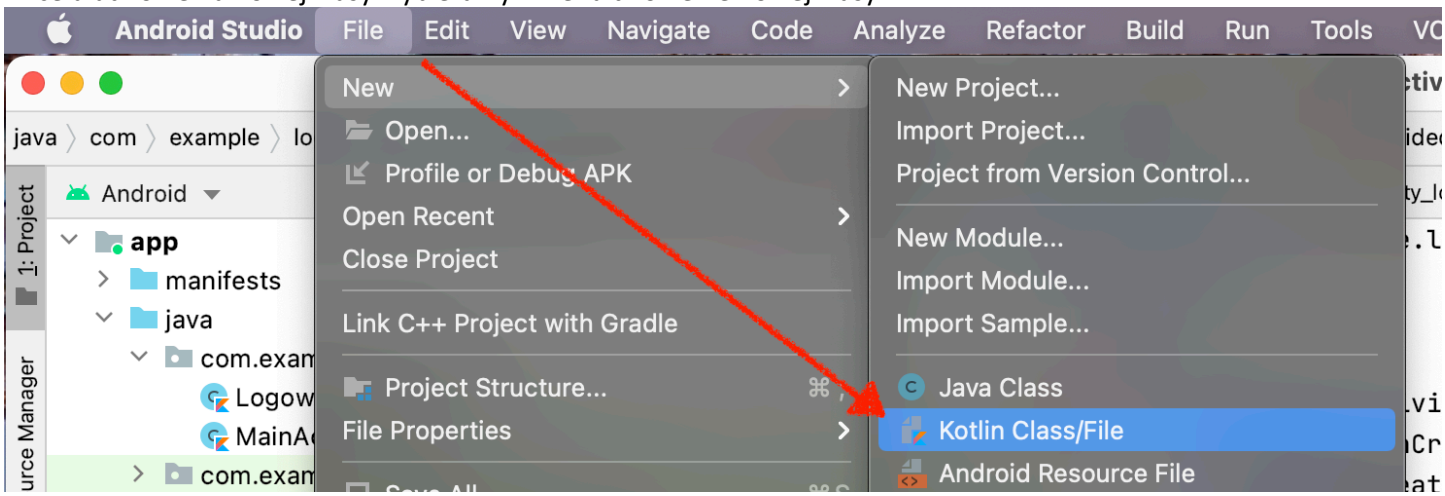
W taki sam sposób dopisujemy zdarzenie e-maila

```
34          email.addTextChangedListener(object : TextWatcher{// nasłuch zmiany tekstu
35          override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {}
36              override fun afterTextChanged(s: Editable?){}
37              override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int){
38                  infoEmail.setText("Proszę wpiać e-mail")
39                  infoEmail.visibility = TextView.VISIBLE
40                  for (i:Char in email.text){
41                      if(i == '@'){
42                          infoEmail.setText("Twoje e-mail jest prawidłowy")
43                          infoEmail.visibility = TextView.VISIBLE
44                      } else {
45                          infoEmail.setText("E-mail jest nie prawidłowy")
46                          infoEmail.visibility = TextView.VISIBLE
47                      }
48                  }
49              }
50          })
```

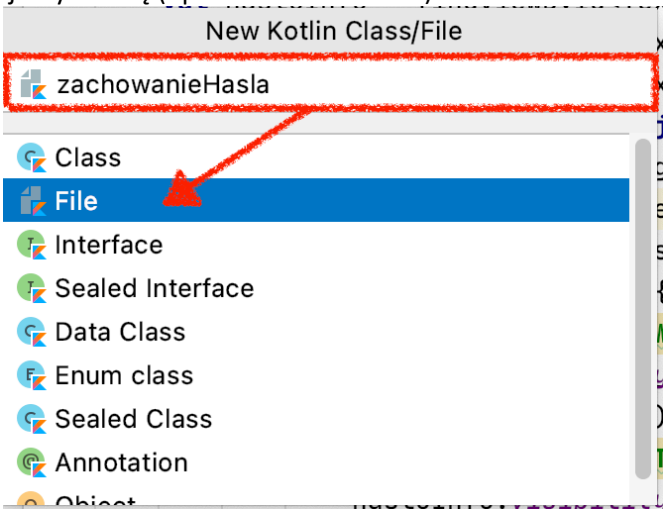
Funkcje takie można umieścić w oddzielnych klasach a w aktywności je tylko wywołać. Dzięki temu będzie można wielokrotnie używać tego samego kodu w innych miejscach.

Utwórzmy nową klasę obsługującą funkcje i ją wywołajmy. Funkcję umieścimy w nowej klasie.

W celu utworzenia nowej klasy wybieramy z menu tworzenie nowej klasy



i nadajemy nazwę (np.: zachowanieHasla):



2-krotnie klikamy w nazwę tworzonego obiekltu (File)

Zostanie utworzony pusty plik do którego przekopiujemy zawartość z pliku ‘LogowanieActivity.kt:

```
haslo.addTextChangedListener(object : TextWatcher{// nasłuch zmiany tekstu
    override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {}
    override fun afterTextChanged(s: Editable?){}
    override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int){
        if(haslo.length() == 0){
            hasloInfo.setText("Wprowadź hasło")
            hasloInfo.visibility = TextView.VISIBLE
        } else if(haslo.length()<8){
            hasloInfo.setText("Twoje hasło jest za krótkie")
            hasloInfo.visibility = TextView.VISIBLE
        } else {
            hasloInfo.setText("Hasło prawidłowe")
            hasloInfo.visibility = TextView.VISIBLE
        }
    }
})
```

(na niebiesko zaznaczony jest tekst kopiowany)

i wklejamy do utworzonej funkcji:

zachowanieHasla.kt | LogowanieActivity.kt | activity_logowanie.xml

```
package com.example.logowanie2

import android.text.Editable
import android.text.TextWatcher
import android.widget.EditText
import android.widget.TextView

fun zachowanieHasla(haslo: EditText , hasloInfo: TextView) {

}
```

Kod po wklejeniu:

```
fun zachowanieHasla(haslo: EditText , hasloInfo: TextView) {
    haslo.addTextChangedListener(object : TextWatcher {
        // nasłuch zmiany tekstu
        override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {}
        override fun afterTextChanged(s: Editable?) {}
        override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {
            if (haslo.length() == 0) {
                hasloInfo.setText("Wprowadź hasło")
                hasloInfo.visibility = TextView.VISIBLE
            } else if (haslo.length() < 8) {
                hasloInfo.setText("Twoje hasło jest za krótkie")
                hasloInfo.visibility = TextView.VISIBLE
            } else {
                hasloInfo.setText("Hasło prawidłowe")
                hasloInfo.visibility = TextView.VISIBLE
            }
        }
    })
}
```

Pozostało jeszcze wywołać funkcję z przekazywanymi argumentami:

```
var haslo = findViewById<EditText>(R.id.poleHasla)
val hasloInfo = findViewById<TextView>(R.id.infoHaslo)
zachowanieHasla(haslo,hasloInfo)
```

W podobny sposób możemy utworzyć plik z naszą funkcją, którą możemy nazwać zachowanieEmaila. (tworzymy plik a w nim funkcję oraz wywołujemy tą funkcję. (możemy także utworzyć nową funkcję w już utworzonym pliku o nazwie zachowanie hasła – nazwa funkcji nie musi być zbieżna z nazwą pliku).

Uzyskamy nową postać plików:

LogowanieActivity.kt

zachowanieHasla.kt | LogowanieActivity.kt | activity_logowanie.xml

```
package com.example.logowanie2

import android.os.Bundle
import android.text.Editable
import android.text.TextWatcher
import android.widget.EditText
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity

class LogowanieActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_logowanie)
        var haslo = findViewById<EditText>(R.id.poleHasla)
        val hasloInfo = findViewById<TextView>(R.id.infoHaslo)
        zachowanieHasla(haslo,hasloInfo)
        var email = findViewById<EditText>(R.id.E_MAIL)
        var infoEmail = findViewById<TextView>(R.id.infoEmail)
        zachowanierEmaila(email,infoEmail)
    }
}
```

zachowanieHasla.kt

zachowanieHasla.kt | LogowanieActivity.kt | activity_logowanie.xml

```kotlin
package com.example.logowanie2

import android.provider.ContactsContract
import android.text.Editable
import android.text.TextWatcher
import android.widget.EditText
import android.widget.TextView

fun zachowanieHasla(haslo: EditText , hasloInfo: TextView) {
    haslo.addTextChangedListener(object : TextWatcher {
        // nasłuch zmiany tekstu
        override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {}
        override fun afterTextChanged(s: Editable?) {}
        override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {
            if (haslo.length() == 0) {
                hasloInfo.setText("Wprowadź hasło")
                hasloInfo.visibility = TextView.VISIBLE
            } else if (haslo.length() < 8) {
                hasloInfo.setText("Twoje hasło jest za krótkie")
                hasloInfo.visibility = TextView.VISIBLE
            } else {
                hasloInfo.setText("Hasło prawidłowe")
                hasloInfo.visibility = TextView.VISIBLE
            }
        }
    })
}
fun zachowanierEmaila(email: EditText, infoEmail: TextView){
    email.addTextChangedListener(object : TextWatcher{// nasłuch zmiany tekstu
    override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {}
        override fun afterTextChanged(s: Editable?){}
        override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int){
            infoEmail.setText("Proszę wpiać e-mail")
            infoEmail.visibility = TextView.VISIBLE
            for (i:Char in email.text){
                if(i == '@'){
                    infoEmail.setText("Twoje e-mail jest prawidłowy")
                    infoEmail.visibility = TextView.VISIBLE
                } else {
                    infoEmail.setText("E-mail jest nie prawidłowy")
                    infoEmail.visibility = TextView.VISIBLE
                }
            }
        }
    })
}
```