PROGRAMOWANIE APLIKACJI MOBILNYCH

Słuchacze zdarzeń

dr Artur Bartoszewski

Słuchacze zdarzeń

Programowanie sterowane zdarzeniami to paradygmat programowania, w którym program otrzymuje informacje o zdarzeniach (ang. event), na które musi odpowiednio zareagować.

Zdarzenia generowane są przez system operacyjny na podstawie aktywności użytkownika (układów wejściowych HID) oraz aktywności innych programów i procesów systemowych, a następnie rozsyłane do odpowiednich aplikacji.

Programowanie sterowane zdarzeniami charakteryzuje się tym, że zamiast zasadniczego nurtu sterowania istnieje wiele niewielkich procedur obsługi zdarzeń, które uruchamiane są w chwili wystąpienia odpowiedniego zdarzenia.

W aplikacjach pisanych w języku Java dla systemu Android zastosowano koncepcję delegacyjnego modelu obsługi zdarzeń. Umożliwia on przekazanie obsługi zdarzenia, związanego z jakimś obiektem (źródłem zdarzenia) do innego obiektu (słuchacza zdarzenia). Zdarzenia reprezentowane są tu przez obiekty odpowiednich klas, określających ich rodzaj.

Słuchacze zdarzeń

Słuchacze zdarzeń (ang. events listeners) są obiektami klas implementujących interfejsy nasłuchu (ang. listener interface).

Interfejsy nasłuchu to klasy których zadaniem jest przechwycenie zdarzenia, rozpoznanie go i odpowiednia reakcja. Posiadają one zestawy metod obsługi danego rodzaju zdarzeń.

W momencie zajścia zdarzenia uruchamiana jest odpowiednia metoda lub dla zdarzeń o bardziej rozbudowanej obsłudze, sekwencja kilku metod.

Aby oprogramować obsługę zdarzenia kod umieścić należy w odpowiednich metodach słuchacza zdarzeń

Słuchacze zdarzeń

Od strony praktycznej:

- Słuchacze zdarzeń to klasy posiadające metody obsługi poszczególnych zdarzeń. Metody są wywoływane, gdy nastąpi odpowiadające im zdarzenie (np. rozpoczęcie przesuwania suwaka, zakończenie przesuwania suwaka itp.)

- Słuchacze są powiązane z określonymi kontrolkami. Np. domyślnym słuchaczem dla kontrolki Button jes OnClickListener, a dla kontrolki SeekBar - ocSeekListener

Obsługa komponentu Button za pomocą słuchacza zdarzeń

View.OnClickListener. Listenr

Istnieją trzy najczęściej stosowane metody obsługi kliknięcia na przycisk:

1. Dodanie do jego opisu w pliku XML parametru onClick informującego jaką metodę należy wywołać po kliknięciu (metoda aktualnie uznana za przestarzałą)
2. Utworzenie dla każdego przycisku jego własnego słuchacza zdarzeń,
3. Zarejestrowanie przycisku do „zbiorczego" słuchacza zdarzeń obsługującego kilka przycisków.

Utworzenie dla przycisku jego własnego słuchacza zdarzeń.

```
public class MainActivity extends AppCompatActivity {

    Button przycisk;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        przycisk = findViewById(R.id.button01);
        przycisk.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // akcja po kliknięciu na przycisk
            }
        });
    }
}
```

Tworzymy pustą referencję do przycisku.

Uzyskujemy referencję przycisku.

Do przycisku dodajmy słuchacza – metoda **setOnClickListener()** której parametrem jest sam listener (klasa anonimowa – cała opisana w i od razu użyta jako parametr wywołania metody)

Ten typ słuchacza posiada tylko jedna metodą **onClick()**, która wywoływana jest w chwili kliknięcia na przycisk – tu wstawiamy kod

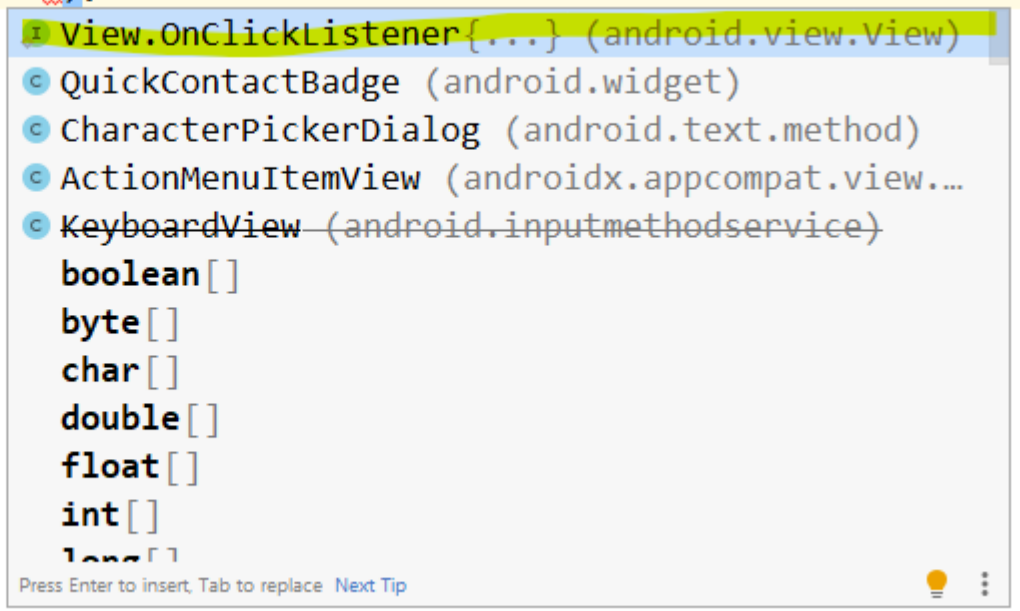
Utworzenie dla przycisku jego własnego słuchacza zdarzeń.

PODPOWIEDŹ: **Tworząc słuchacza należy użyć kreatora**

```
przycisk.setOnClickListener(new );
```

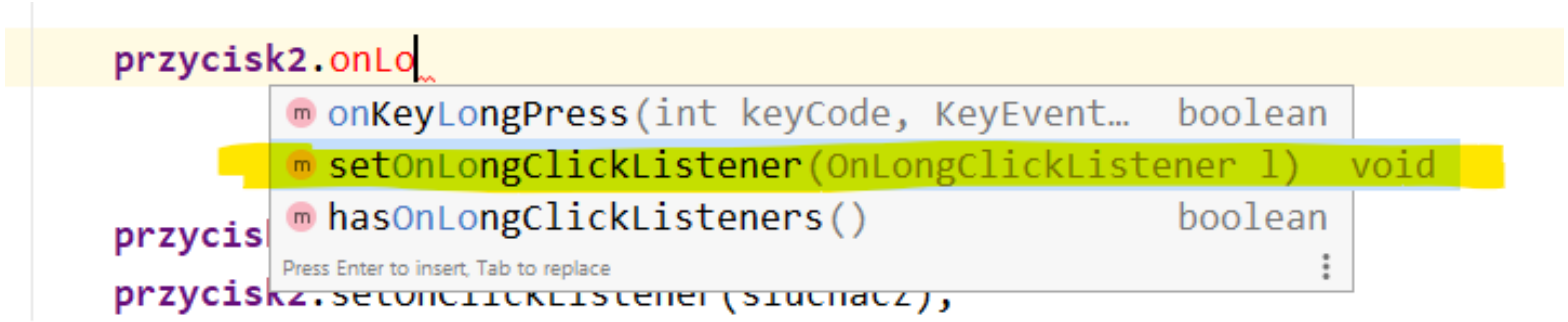
Ctrl + Spacja

```
przycisk.setOnClickListener(new );
```



Obsługa komponentu Button – słuchacz „długiego kliknięcia”

Słuchacz „długiego kliknięcia” OnLongClickListener. Listener



Drugim, często używanym słuchaczem zdarzeń dla przycisku (i nie tylko) jest słuchacz **OnLongClickListener()** który reaguje na „długie kliknięcie” czyli przytrzymanie elementu

Obsługa komponentu Button – słuchacz „długiego kliknięcia”

Słuchacz „długiego kliknięcia” OnLongClickListener. Listener

```
przycisk2.setOnLongClickListener(new View.OnLongClickListener() {
    @Override
    public boolean onLongClick(View view) {
        // tu wpisz kod
        return false;
    }
});
```

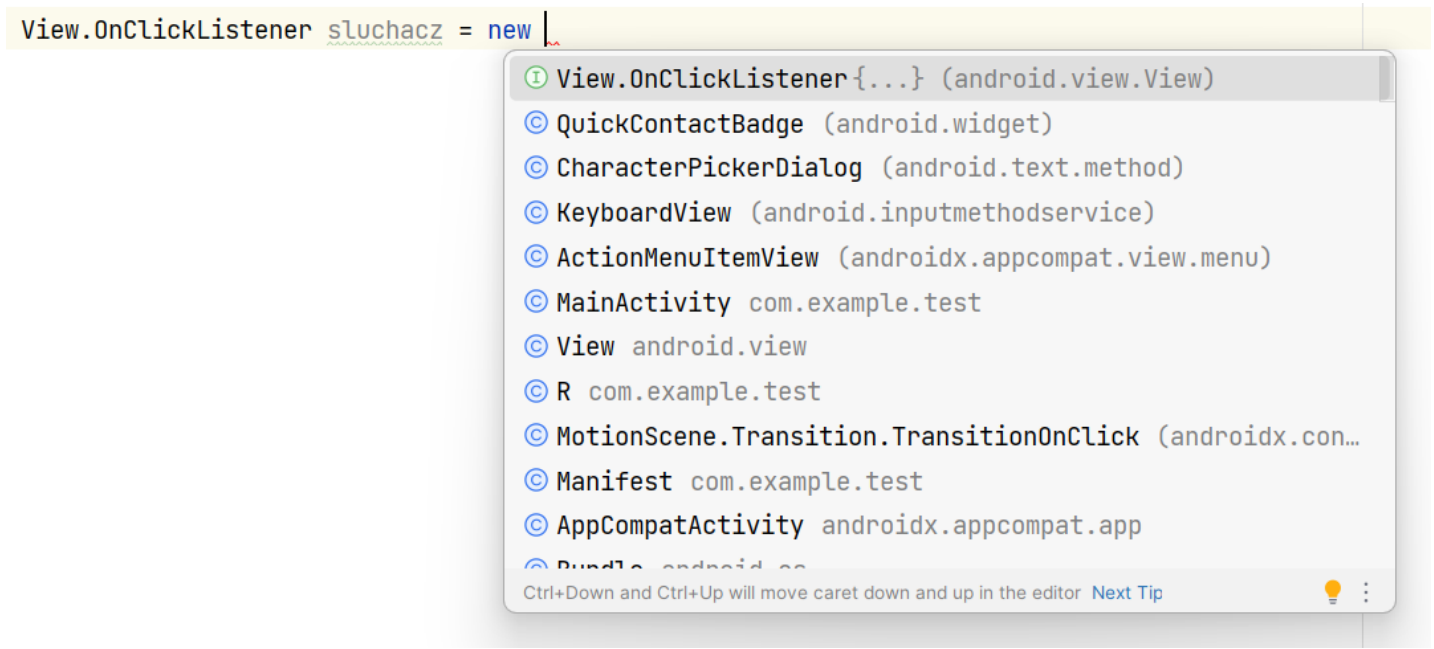
Słuchacz ten dostarcza metodę onLongClick, która wywoływana jest chwilę po przytrzymaniu widoku.

Zarejestrowanie przycisku do „zbiorczego” słuchacza zdarzeń obsługującego kilka przycisków.

Aby słuchacz mógł obsługiwać kilka kontrolek musi ma on być klasycznie utworzonym obiektem takim do którego posiadamy referencję (mówiąc nieco nieprecyzyjnie – musimy nadać mu nazwę)

- Aby zdarzenie zostało przekazane do obsługi słuchaczowi musi być on przyłączony do widoku który jest jego źródłem.
- Operacja ta wykonywana jest za pomocą metody setOnClickListener()

Zarejestrowanie przycisku do „zbiorczego” słuchacza zdarzeń obsługującego kilka przycisków.



- Aby zdarzenie zostało przekazane do obsługi słuchaczowi musi być on przyłączony do widoku który jest jego źródłem.
- Operacja ta wykonywana jest za pomocą metody setOnClickListener()

Zarejestrowanie przycisku do „zbiorczego” słuchacza zdarzeń obsługującego kilka przycisków.

```
public class MainActivity extends AppCompatActivity {
    Button button01, button02, button03;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        button01 = findViewById(R.id.button01);
        button02 = findViewById(R.id.button02);
        button03 = findViewById(R.id.button03);

        View.OnClickListener sluchacz = new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // tu wpisz kod
            }
        };
        button01.setOnClickListener(sluchacz);
        button02.setOnClickListener(sluchacz);
        button03.setOnClickListener(sluchacz);
    }
}
```

Tworzymy obiekt klasy View.OnClickListener. Listenrowi nadano nazwę „sluchacz”

Podobnie jak w poprzednim przykładzie posiada on metodę onClick(), którą oprogramowujemy.

Listener „słuchacz” rejestrujemy wszystkie kontrolki, z którymi ma współpracować. Inaczej słuchacz będzie istniał – ale nigdy nie zostanie wywołony

UWAGA: należy uzupełnić średnik, środowisko samo tego nie zrobi.

Zarejestrowanie przycisku do „zbiorczego” słuchacza zdarzeń obsługującego kilka przycisków.

```
View.OnClickListener sluchacz = new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        int Id = v.getId();

        if(Id==R.id.button01)
        {
            //akcja po kliknięciu w button01
        }
        else if( Id==R.id.button02)
        {
            //akcja po kliknięciu w button02
        }
        else if( Id==R.id.button03)
        {
            //akcja po kliknięciu w button03
        }
    }
};
```

Referencja do obiektu, na rzecz którego wywołany będzie słuchacz przekazana jest w jego parametrze jako zmienna „v” typu View.

Z „v” wyciągamy id obiektu na rzecz którego wywołano słuchacza za pomocą metody .getId().

Pozwala to rozpoznać, który przycisk został kliknięty. Porównujemy Id z identyfikatorem nadanym w XML-u za pośrednictwem pliku R

Zarejestrowanie przycisku do „zbiorczego” słuchacza zdarzeń obsługującego kilka przycisków.

CIEKAWOSTKA: W większości starszych podręczników i tutoriali zamiast drabinki else-if wykorzystywana jest znacznie czytelniejsza instrukcja switch(Id).
Niestety od połowy 2023r. konstrukcja ta już nie działa.

```
View.OnClickListener sluchacz = new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        int id = view.getId();
        switch(id)
        {
            case R.id.button01: //alcja dla przycisku button1
                break;
            case R.id.button02: //alcja dla przycisku button2
                break;
        }
    }
};
```

Wynika to z bardziej restrykcyjnego podejścia środowiska Android Studio do składni języka JAVA.

Zmienna sterująca w instrukcji switch() powinna być porównywana ze stałą, a „R.id.button01” stałą nie jest

Słuchacz reagujący na wybrane elementu (fokus)

OnFocusChangeListener

```
editText.setOnFocusChangeListener(new View.OnFocusChangeListener() {
    @Override
    public void onFocusChange(View v, boolean hasFocus) {
        if (hasFocus) {
            v.setBackgroundColor(Color.GREEN);
        } else {
            v.setBackgroundColor(Color.WHITE);
        }
    }
});
```

Słuchacz ten posiada 2 parametry:

- referencje do obiektu dla którego został wywołany ,
- informację czy obiekt ten posiada fokus.

Obsługa komponentu EditText

Słuchacz reagujący na zmianę tekstu (edycję)

addTextChangedListener

TextWatcher()

```
editText.addTextChangedListener(new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int after) {
    }
    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {
    }
    @Override
    public void afterTextChanged(Editable s) {
    }
});
```

Uwaga: słuchacz zdarzeń TextChangedListener korzysta z interfejsu TextWatcher() co może być nieco mylące.

Obsługa komponentu EditText

beforeTextChanged(CharSequence s, int start, int count, int after)
Kiedy: Wywoływana przed zmianą tekstu w widoku tekstowym.
Parametry:
- s: Aktualny tekst przed zmianą.
- start: Pozycja początkowa zmienianego tekstu.
- count: Liczba znaków, które zostaną usunięte.
- after: Liczba znaków, które zostaną dodane.

onTextChanged(CharSequence s, int start, int before, int count)
Kiedy: Wywoływana podczas zmiany tekstu w widoku tekstowym.
Parametry:
- s: Tekst po zmianie.
- start: Pozycja początkowa zmienianego tekstu.
- before: Liczba znaków, które zostały usunięte.
- count: Liczba znaków, które zostały dodane.

afterTextChanged(Editable s)
Kiedy: Wywoływana po zmianie tekstu w widoku tekstowym.
Parametry:
- s: Tekst po zmianie. Można go modyfikować.

Modyfikowanie tekstu w metodzie onTextChanged może prowadzić do nieskończonej pętli.

Zaleca się modyfikowanie tekstu w metodzie afterTextChanged.

Obsługa komponentu EditText

Przykład: Zamiana „w locie” na wielkie litery

```
editText.addTextChangedListener(new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int after) {
    }
    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {
    }
    @Override
    public void afterTextChanged(Editable s) {
        if (s != null) {
            String upperCaseText = s.toString().toUpperCase();
            if (!upperCaseText.equals(s.toString())) {
                s.replace(0, s.length(), upperCaseText);
            }
        }
    }
});
```

Wyrażenia Lambda

Wyrażenia lambda w języku Java to konstrukcja wprowadzona w wersji 8 języka, która umożliwia tworzenie funkcji anonimowych, tj. funkcji, które nie mają zdefiniowanej nazwy. Są one przydatne do zwięzłego definiowania zachowań w kodzie.

(parametry) -> { ciało wyrażenia }

W kontekście tego wykładu – wykorzystamy je do budowy słuchaczy zdarzeń.

```
Button button01 = findViewById(R.id.button01);

button01.setOnClickListener((e) ->{
   // akcja po kliknięciu przycisku
});
```

Zamiast tworzyć obiekt anonimowy (słuchacza zdarzeń) można w jego miejsce użyć wyrażenia lambda.
Pamiętajmy jednak, że może ona zastąpić jedynie słuchacza zdarzeń, który posiada tylko jedną, domyślną metodę obsługi zdarzenia.

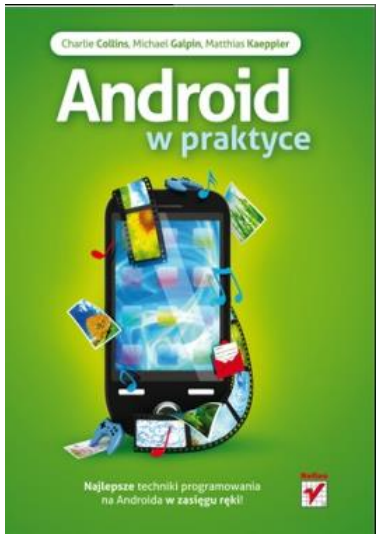
Wyrażenia Lambda

Porównanie – obiekt anonimowy vs wyrażenie lambda

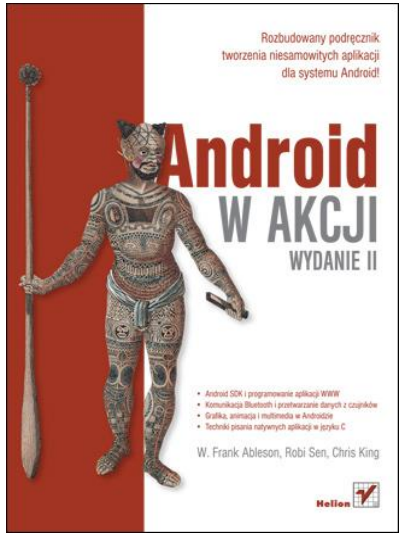
```
editText.setOnFocusChangeListener(new View.OnFocusChangeListener() {
    @Override
    public void onFocusChange(View v, boolean hasFocus) {
        if (hasFocus) {
            v.setBackgroundColor(Color.GREEN);
        } else {
            v.setBackgroundColor(Color.WHITE);
        }
    }
});
```

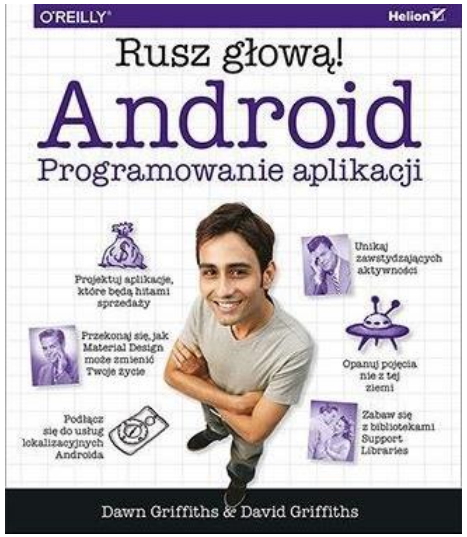
```
editText.setOnFocusChangeListener((v, hasFocus) -> {
    if (hasFocus) {
        v.setBackgroundColor(Color.GREEN);
    } else {
        v.setBackgroundColor(Color.WHITE);
    }
});
```

Literatura



https://developer.android.com





https://javastart.pl/baza-wiedzy/android/

https://forum.android.com.pl