Wprowadzenie do fragmentów w Android Studio

Co to jest fragment?

Fragment to samowystarczalny moduł interfejsu użytkownika aplikacji oraz odpowiadające mu działanie, który może być osadzony w aktywności (Activity).

Podczas fazy projektowania aplikacji fragmenty można łączyć (aby utworzyć aktywność), oraz je dodawać lub usuwać z aktywności podczas działania aplikacji, tworząc dynamicznie zmieniający się interfejs użytkownika.

Fragmenty mogą być użyte tylko jako część aktywności i nie mogą być samodzielnym elementem aplikacji.

Jednakże, fragment można uznać za pod-aktywność (sub-activity), posiadającą własny cykl życia, podobny do cyklu życia pełnej aktywności.

Można je dodać do aktywności poprzez umieszczenie odpowiednich elementów <fragment> w pliku aktywności (.xml) lub poprzez wstawienie kodu w implementacji klasy aktywności (.kt)

Tworzenie Fragmentu

Fragment składa się z dwóch komponentów:

- plik układu (layoutu) XML i
- odpowiadająca mu klasa Kotlin.

Plik układu XML ma ten sam format, co układ dowolnej innej aktywnośc i może składać się z dowolnych kombinacji układów i widoków.

Poniższy plik XML jest układem dla fragmentu składającego się z układu ConstraintLayout z czerwonym tłem i mający pojedyńczy element TextView w kolorze białym:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/constraintLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/holo_red_dark"
    tools:context=".FragmentOne">
     <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="My First Fragment"
        android:textAppearance="@style/TextAppearance.AppCompat.Large"
        android:textColor="@color/white"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Odpowiadająca układowi klasa musi dziedziczyć z klasy Android *Fragment* i powinna, co najmniej, nadpisać metodę *onCreateView()*, która jest odpowiedzialna za załadowanie układu (layoutu) fragmentu. Na przykład:

```
package com.example.myfragmentdemo

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment

class FragmentOne : Fragment() {

    private var _binding: FragmentTextBinding? = null
    private val binding get() = _binding!!

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        _binding = FragmentTextBinding.inflate(inflater, container, false)
        return binding.root
    }
}
```

Dodatkowo oprócz metody *onCreateView()*, klasa może również nadpisywać standardowe metody cyklu życia fragmentu (standard lifecycle methods).

Po utworzeniu pliku układu i klasy fragment jest gotowy do użycia w aplikacji.

Dodanie fragmentu do Aktywności używając plik układu XML

Fragmenty można włączyć do Aktywności, pisząc kod w Kotlinie lub osadzając fragment w pliku układu XML.

Niezależnie od zastosowanej metody, kluczową kwestią, o której należy pamiętać, jest to, że gdy biblioteka wsparcia (suport library) jest używana w celu zapewnienia zgodności ze starszymi wersjami Androida, wszelkie działania wykorzystujące fragmenty muszą być zaimplementowane jako podklasa (dziedziczyć z) *FragmentActivity* zamiast *AppCompatActivity*:

```
package com.example.myFragmentDemo

import androidx.fragment.app.FragmentActivity
import android.os.Bundle

class MainActivity : FragmentActivity() {
.
.
```

Fragmenty są osadzane w pliku układu aktywności przy użyciu klasy **FragmentContainerView**.

Poniższy przykład osadza fragment w pliku xml aktywności:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/fragment2"
        android:name="com.amw.myfragmentdemo.FragmentOne"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="32dp"
        android:layout_marginEnd="32dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:layout="@layout/fragment_one" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Kluczowe właściwości dla elementu <fragment> to *android:name*, który musi się odwoływać do klasy powiązanej z fragmentem, oraz *tools:layout*, który musi się odwoływać do pliku XML zawierającego układ fragmentu.

Po dodaniu do aktywności układu aktywności, fragmenty można widzieć i manipulować nimi za pomocą narzędzi edytora układu.

Dodawanie i zarządzanie fragmentami w kodzie

Łatwość dodawania fragmentu do aktywności za pośrednictwem pliku układu XML odbywa się kosztem aktywności, która nie może usunąć fragmentu w czasie wykonywania (run time).

Aby uzyskać pełną dynamiczną kontrolę nad fragmentami w czasie wykonywania, działania te należy dodać za pomocą kodu.

Ma to tę zaletę, że fragmenty można dodawać, usuwać, a nawet dynamicznie zastępować podczas działania aplikacji.

W przypadku używania kodu do zarządzania fragmentami fragment będzie nadal składał się z pliku układu XML i odpowiedniej klasy.

Różnica pojawia się podczas pracy z fragmentem wewnątrz głównej aktywności (hosting activity).

Podczas dodawania fragmentu do działania za pomocą kodu obowiązuje standardowa sekwencja kroków:

1. Tworzenie instancji klasy fragmentu.
2. Przekazanie dodatkowych argumenty intencji do instancji klasy.
3. Utworzenie odwołanie (referencji) do instancji menedżera fragmentów.
4. Wykonanie metody *beginTransaction()* managera fragmentów. Otrzymamy instancję transakcji fragmentu.
5. Wykonanie metody *add()* transakcji przekazując jako argumenty identyfikator zasobu widoku, który ma zawierać fragment oraz instancję klasy fragment.
6. Wykonanie metody *commit()* transakcji.

Poniższy przykładowy kod dodaje fragment zdefiniowany przez klasę FragmentOne, który pojawi się w kontenerze z ID LinearLayuot1:

```
val firstFragment = FragmentOne()
firstFragment.arguments = intent.extras
val transaction = fragmentManager.beginTransaction()
transaction.add(R.id.LinearLayout1, firstFragment)
transaction.commit()
```

Powyższy kod zapisuje każdy krok w osobnych liniach (dla jasności kodu).

Aczkolwiek, cztery ostatnie linie, mogą być skrócone do jednej linii kodu:

```
supportFragmentManager.beginTransaction().add(
      R.id.LinearLayout1, firstFragment).commit()
```

Po dodaniu do kontenera, fragment może być usunięty metodą *remove()*:

```
transaction.remove(firstFragment)
```

Podobnie jeden fragment można zastąpić innym poprzez wywołanie metody *replace()*, jako argumenty podając identyfikator ID widoku zawierającego fragment i instancję nowego fragmentu.
Zastąpiony fragment można również umieścić na tak zwanym stosie, aby można go było szybko przywrócić, jeśli użytkownik wróci do niego.
Osiąga się to poprzez wywołanie metody *addToBackStack()* przed wywołaniem metody *commit()*:

```
val secondFragment = FragmentTwo()
transaction.replace(R.id.LinearLayout1, secondFragment)
transaction.addToBackStack(null)
transaction.commit()
```

Obsługa zdarzeń fragmentu

Fragment jest jakby podaktywnością (sub-activity) ze swoim układem, klasą i cyklem życia.

Komponenty widoków (takie jak klawisze, pola tekstowe) wewnątrz fragmentu mogą generować zdarzenia tak jak normalne elementy.

To rodzi pytanie, która klasa odbiera zdarzenie z elementu fragmentu, samego fragmentu czy aktywności, w której fragment jest osadzony.

Odpowiedź zależy od sposobu zadeklarowania obsługi zdarzenia.

Pierwsza metoda polega na konfiguracji detectora zdarzeń (event listener) oraz na wykonaniu metody callback w kodzie aktywności:

```
binding.button.setOnClickListener { // Code to be performed on button click }
```

Druga metoda obsługi zdarzeń kliknięcia polega na ustawieniu atrybutu *android:onClick* w pliku układu XML:

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onClick"
    android:text="Click me" />
```

Ogólna zasada dotycząca zdarzeń generowanych przez element fragmentu jest taka, że jeśli detektor zdarzeń (event listener) został zadeklarowany w klasie fragmentu przy użyciu detektora zdarzeń i metody callback, zdarzenie zostanie obsłużone jako pierwsze przez fragment.

Aczkolwiek, jeśli używamy *android:onClick,* zdarzenie będzie przekazane bezpośrednio do aktywności, w której fragment jest osadzony.

Implementacja komunikacji fragmentu

Jeśli mamy osadzone jeden lub więcej fragmentów w aktywności, to jest pewne, że będą się one musiały komunikować z aktywnością lub pomiędzy sobą.

Dobra praktyka nakazuje, aby fragmenty nie komunikowały się ze sobą bezpośrednio.

Cała komunikacja powinna się odbywać poprzez aktywność, w której są osadzone.

Do komunikacji z fragmentem musimy zidentyfikować jego ID.

Po identyfikacji fragmentu, aktywność może wywoływać jego publiczne metody (public methods).

Komunikacja w drugą stronę (z fragmentu do aktywności) jest bardziej skomplikowana.

W pierwszej kolejności fragment musi definiować interfejs nasłuchu (listener interface), który następnie jest implementowany w klasie aktywności.

Poniższy przykładowy kod deklaruje interfejs PierwszyListener fragmentu PierwszyFragment oraz zmienne w których będzie później przechowywana referencja do aktywności:

```
class PierwszyFragment : Fragment() {

    var activityCallback: PierwszyFragment.PierwszyListener? = null

    interface PierwszyListener {
        fun onButtonClick(arg1: Int, arg2: String)
    }
…..
}
```

Powyższy kod nakazuje, że każda klasa implementująca interfejs PierwszyListener musi także implementować metodę wywołania zwrotnego (callback) o nazwie onButtonClick, która z kolei akceptuje liczbę całkowitą i ciąg znaków jako argumenty (przykładowo).

Następnie należy nadpisać (override) metodę *onAttach()* klasy fragment.

Metoda ta jest wywoływana automatycznie przez system Android, gdy fragment został zainicjowany i powiązany z aktywnością.

Do metody przekazywane jest referencja do aktywności, w której osadzony jest fragment.

Metoda musi przechowywać lokalną referencję do tej aktywności i sprawdzić, czy implementuje interfejs PierwszyListener:

```
override fun onAttach(context: Context?) {
    super.onAttach(context)
    try {
        activityCallback = context as PierwszyListener
    } catch (e: ClassCastException) {
        throw ClassCastException(context?.toString()
           + " musimy zaimplementować PierwszyListener")
    }
}
```

Po wykonaniu tego przykładu referencja do aktywności zostanie zapisana w lokalnej zmiennej *activityCallback* i zostanie zgłoszony wyjątek, jeśli to działanie nie implementuje interfejsu PierwszyListener.

Następnym krokiem jest wywołanie metody callback (aktywności) z wewnątrz fragmentu.

Kiedy i jak to nastąpi, zależy całkowicie od okoliczności, w jakich fragment ma nawiązać kontakt z aktywnością.

Poniższy przykładowy kod wywołuje metodę callback po kliknięciu klawisza:

```
override fun onButtonClick(arg1: Int, arg2: String) {
   activityCallback.onButtonClick(arg1, arg2)
}
```

Pozostaje tylko zmodyfikować klasę aktywności aby zaimplementować interfejs PierwszyListener:

```
class MainActivity : FragmentActivity(),
      PierwszyFragment.PierwszyListener {

    override fun onButtonClick(arg1: Int, arg2: String) {
          // Implement code for callback method
    }
.
.
}
```

Jak widać z powyższego kodu, aktywność deklaruje, że będzie implementować interfejs PierwszyListener klasy PierwszyFragment, a następnie przystępuje do implementacji metody *onButtonClick()* zgodnie z wymaganiami interfejsu.

Tworzenie i używanie fragmentów - przykład

Opis tworzonej aplikacji

Tworzona aplikacja będzie składała się z jednej aktywności i dwóch fragmentów.

Interfejs użytkownika pierwszego fragmentu:

- Układ ConstraintLayout,
- PlainText (EditText),
- Button,
- 2 x TextView

Interfejs użytkownika drugiego fragmentu:

- Układ ConstraintLayout,
- PlainText (EditText),
- Button,
- 2 x TextView

Obydwa fragmenty zostaną osadzone w głównej aktywności aplikacji i będą współpracowały.

Po kliknięciu klawisza w pierwszym fragmencie tekst wprowadzony do widoku EditText pojawi się w widoku TextView drugiego fragmentu.

Po kliknięciu klawisza w drugim fragmencie tekst wprowadzony do widoku EditText pojawi się w widoku TextView pierwszego fragmentu.

Ponieważ ta aplikacja jest przeznaczona do pracy na wcześniejszych wersjach Androida, będziemy musieli skorzystać z odpowiedniej biblioteki obsługi Androida.

Utworzenie przykładowego projektu

Tworzymy nowy projekt według szablonu Empty Views Activity i nazywamy go ***ProsteFragmenty***.

Modyfikujemy projekt aby użyć view binding.

1. Edytujemy plik skryptowy Gradle: *Gradle Scripts -> build.gradle.kts (Module :app)*.

2. W sekcji *android* dodajemy zezwolenie na *viewBinding*:

```
android {

    buildFeatures {
        viewBinding = true
    }
```

3. Klikamy klawisz *Sync Now* w górnej części edytora, aby ponownie zsynchronizować projekt z nowymi ustawieniami kompilacji.
4. Modyfikujemy plik *MainActivity.kt*:

```
package com.amw.prostefragmenty

import android.os.Bundle
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import com.amw.prostefragmenty.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)
            insets
        }
    }
}
```

Wracamy do pliku *Gradle Scripts -> build.gradle.kts (Module :app)* i dodajemy dyrektywy do sekcji *dependencies* (mogą być dostępne nowsze wersje biblioteki):

```
implementation ("androidx.navigation:navigation-fragment-ktx:2.6.0")
implementation ("androidx.navigation:navigation-fragment-ktx:2.7.7")
```

Ponownie synchronizujemy projekt (*Sync Now*).

Tworzymy pierwszy Fragment oraz jego układ (layout)

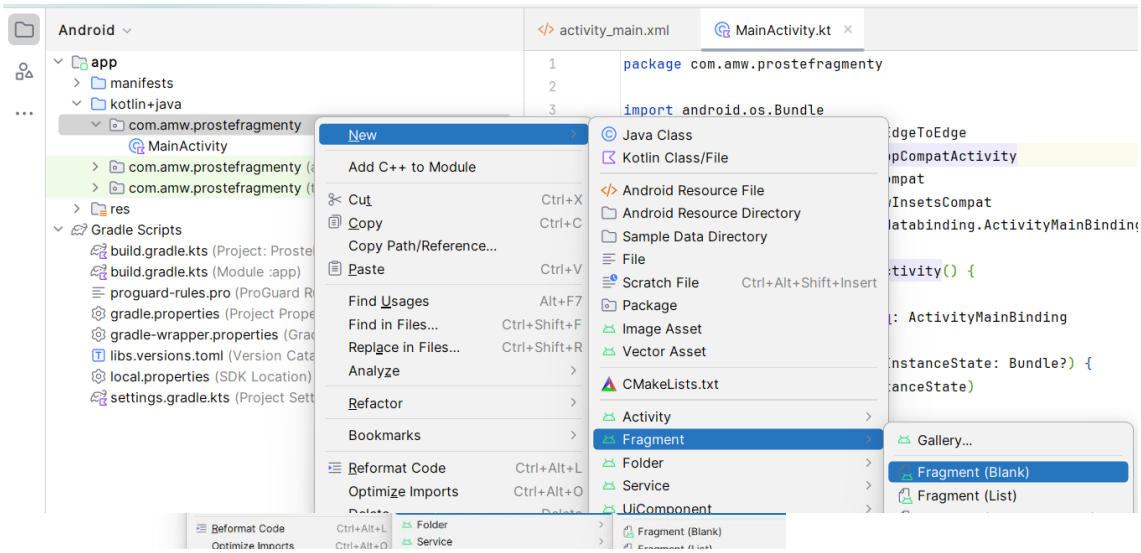
Interfejs będzie się składał z pliku układu XML oraz klasy fragmentu.

Można to zrobić manualnie, ale szybciej będzie jeśli wykona to za nas Android Studio.

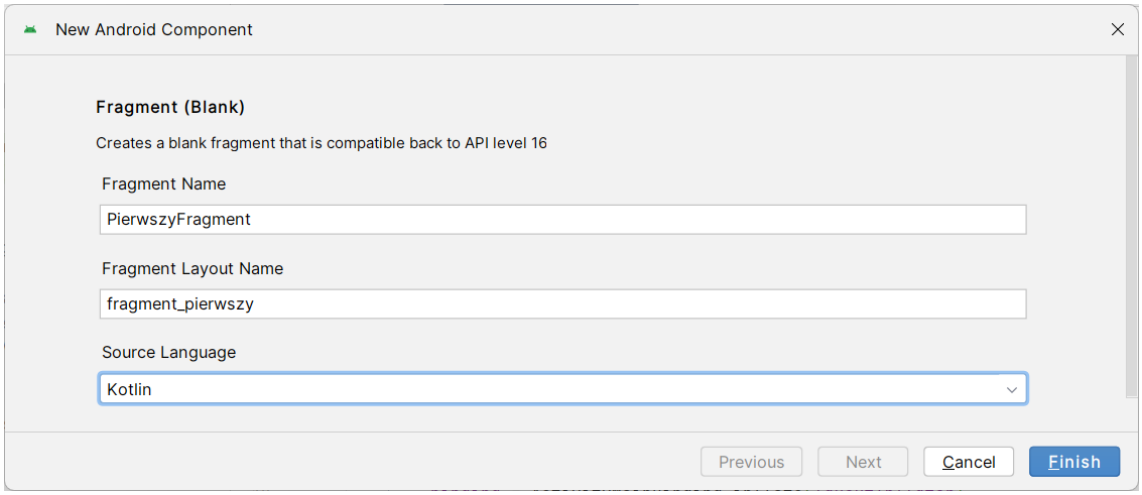
W oknie narzędzia projektu zlokalizuj plik:

app -> kotlin+java -> com.amw.prostefragmenty i kliknij prawym przyciskiem myszy.

Pojawi się menu z którego wybieramy opcję *New -> Fragment -> Fragment (Blank)*

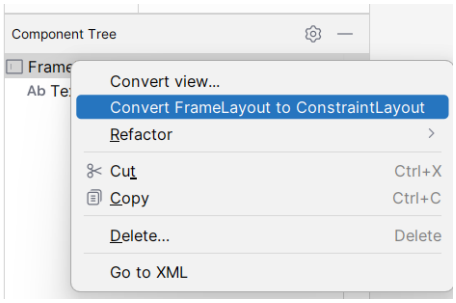


Na następnym ekranie fragment nazywamy *PierwszyFragment* a plik układu będzie się nazywał *fragment_pierwszy.xml*:



Otwieramy do edycji plik *fragment_pierwszy.xml* w trybie Design.

Następnie w panelu Component Tree klikamy prawym przyciskiem na FrameLayout i wybieramy *Convert FrameLayout to ConstraintLayout*, akceptując ustawienia domyślne.

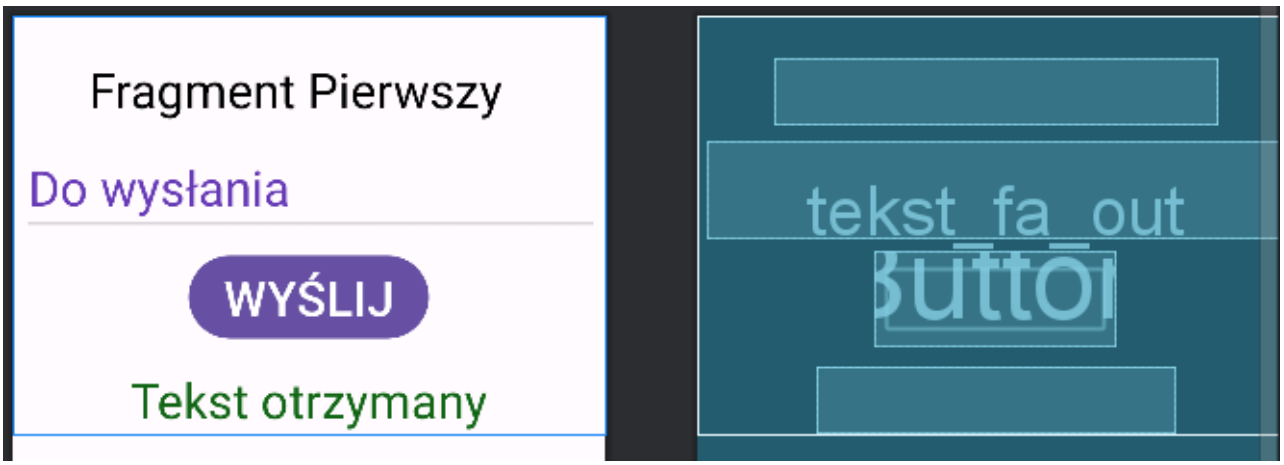


Zmieniamy jego id na *fragmentA*.

Sprawdzamy czy tryb autoconnect jest zezwolony i usuwamy domyślny TextView, po czym dodajemy widgety TextView, PlainText, Button, TextView:

parametr	**TextView**	**PlainText**	**Button**	**TextView**	**układ**
id	tekstFaName	tekstFaOut	klawiszFa	tekstFaIn	fragmentA
text	FRAGMENT A	Do wysłania	WYŚLIJ	Tekst otrzymany	
width	match_parent	wrap_content	wrap_content	wrap_content	match_parent
height	wrap_content	wrap_content	wrap_content	wrap_content	wrap_content
textColor	#000000	#673AB7		#126716	
textSize	34sp	30sp	30sp	30sp	
textStyle	bold				
background	#FFFFFF				#FFEB3B
paddingTop					20sp
paddingBottom					20sp

Użyj przycisku *Infer constraints* aby dodać brakujące powiązania, układ powinien wyglądać jak na rysunku:



Stosujemy View Binding

Podobnie jak w przypadku szablonu EmptyViewActivity, Android Studio nie umożliwia obsługi View Binding po dodaniu nowych fragmentów do projektu.

Dlatego, przed przejściem do następnego kroku, będziemy to muieli wykonać sami.

Otwieramy plik *PierwszyFragment.kt* i dokonujemy koniecznych zmian:

- kasujemy niepotrzebne instrukcje,
- importujemy binding,
- modyfikujemy metodę *onCreateView()* i
- dodajemy metodę *onDestroyView()*). Dodana metoda powoduje usunięcia bindowania fragmentu po jego zniszczeniu.

```
package com.amw.prostefragmenty

import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import com.amw.prostefragmenty.databinding.FragmentPierwszyBinding

class PierwszyFragment : Fragment() {

    private var _binding: FragmentPierwszyBinding? = null
    private val binding get() = _binding!!

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        _binding = FragmentPierwszyBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```

Po wprowadzeniu tych zmian fragment jest gotowy do użycia view binding.

Dodanie drugiego Fragmentu

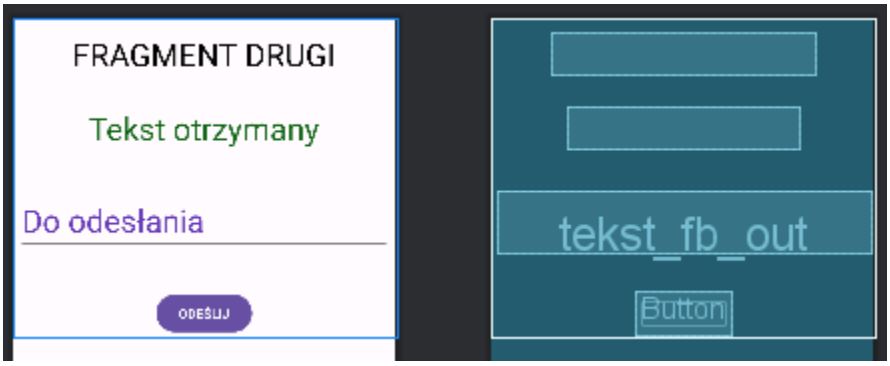
Powtarzając kroki z tworzenia pierwszego fragmentu dodajemy drugi pusty fragment i nazywamy go DrugiFragment z plikiem układu nazwanym *fragment_drugi*.

Konwertujemy FrameLayout na ConstraintLayout (zmieniając id na *fragment2*).

Sprawdzamy czy tryb autoconnect jest zezwolony i usuwamy domyślny TextView, po czym dodajemy widgety TextView, PlainText, Button, TextView:

parametr	**TextView**	**TextView**	**PlainText**	**Button**	**układ**
id	tekstFbName	tekstFbIn	tekstFbOut	klawiszFb	fragmentB
text	FRAGMENT B	Tekst otrzymany	Do odesłania	ODEŚLIJ	
width	wrap_content	wrap_content	wrap_content	wrap_content	match_parent
height	wrap_content	wrap_content	wrap_content	wrap_content	wrap_content
textColor	#000000	#126716	#673AB7		
textSize	34sp	30sp	30sp	30sp	
textStyle	bold				
background	#FFFFFF				#CAECA4
paddingTop					20sp
paddingBottom					20sp

Ostatecznie układ powinien wyglądać jak na rysunku (uzywając klawisza *Infer constraints* do dodania brakujących powiązań):



Otwieramy plik *DrugiFragment.kt* i dokonujemy koniecznych zmian *(jak w PierwszyFragment.kt)*.

```
 package com.amw.prostefragmenty

import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import com.amw.prostefragmenty.databinding.FragmentDrugiBinding

class DrugiFragment : Fragment() {

    private var _binding: FragmentDrugiBinding? = null
    private val binding get() = _binding!!

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        _binding = FragmentDrugiBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```

Dodanie fragmentów do aktywności

Główna aktywność aplikacji jest powiązana z plikiem układu *activity_main.xml*, w którym dodamy fragmenty do aktywności stosując element <fragment>.

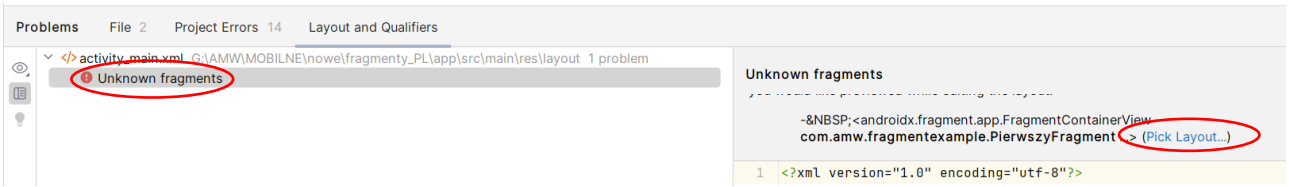
Otwieramy do edycji plik *activity_main.xml*. Usuwamy z układu domyślny widget TextView.

Konwertujemy układ domyślny ConstraintLayout na LinearLayout (vertical) i wybieramy kategorię *Common* z palety komponentów.

Przeciągamy z listy komponent *FragmentContainerView* i umieszczamy go w układzie (layout).

Pojawi się okno dialogowe z dostępnymi w projekcie fragmentami:

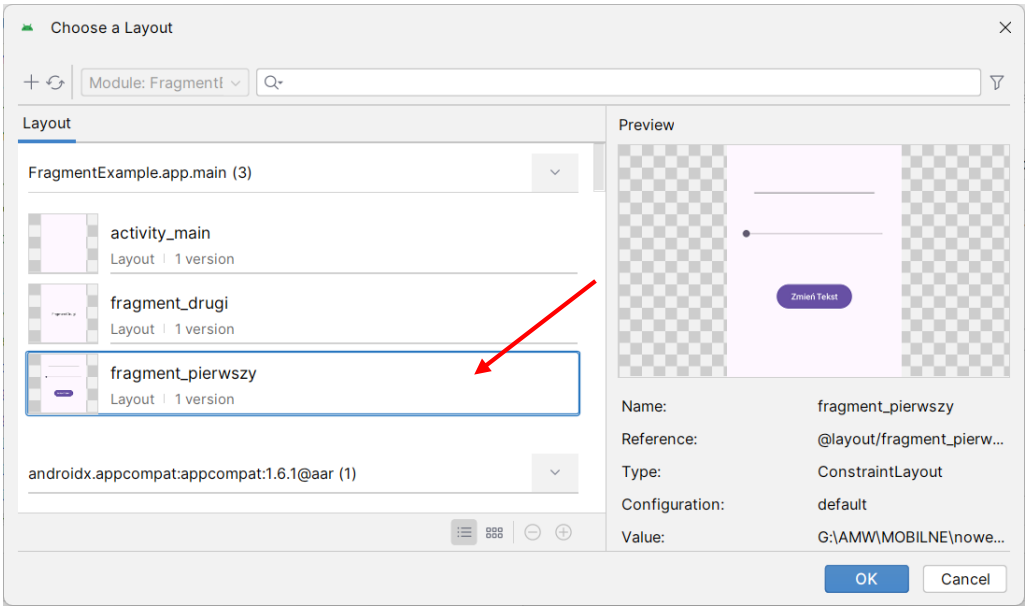
Wybierz **PierwszyFragment** z listy. Po dodaniu, kliknij czerwone ostrzegawcze pole w prawym-górnym rogu układu aby wyświetlić okno Problems



Wiadomość *unknown fragments* message wskazuje, że narzędzie Layout Editor musi wiedzieć który fragment ma wyświetlić w podglądzie.

Wybierz Unknown fragment i kliknij link *Pick Layout…* w prawym panelu.

Pojawi się okno w którym wybierz *fragment_pierwszy* i kliknij OK:



Dla wstawionego fragmentu zmień właściwość *layout_width* na *match_constraint* aby zajęła całą szerokość ekranu.

Przeciągnij następny element *FragmentContainerView* i umieść go centralnie poziomo poniżej pierwszego fragmentu.

Wybierz z listy *DrugiFragment* i kliknik OK.

Wyświetl okno narzędzi Problems i powtórzpoprzednie kroki wybierając układ *fragment_drugi*.

Fragmenty są teraz widoczne w układzie:

Nawiązywanie kontaktu fragmentu pierwszego z aktywnością główną

Kiedy użytkownik kliknie klawisz we fragmencie pierwszym, klasa fragmentu będzie musiała wyodrębnić tekst z widoku EditText (tekstFaOut) i wysłać go do fragmentu drugiego.

Fragmenty nie powinny komunikować się ze sobą bezpośrednio, dlatego muszą wykorzystywać aktywność w której są osadzone, jako pośrednika.

Fragment pierwszy musi reagować na kliknięcie klawisza.

Uczynimy to w klasie PierwszyFragment. Zmodyfikujemy plik *PierwszyFragment.kt*:

```
package com.amw.prostefragmenty

import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import com.amw.prostefragmenty.databinding.FragmentPierwszyBinding

class PierwszyFragment : Fragment() {

    private var _binding: FragmentPierwszyBinding? = null
    private val binding get() = _binding!!

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        _binding = FragmentPierwszyBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.klawiszFa.setOnClickListener { v: View -> onButtonFaClicked(v) }
    }

    private fun onButtonFaClicked(view: View) {
    }

}
```

Zanim będziemy kontynuować musimy wyjaśnić dokonane zmiany.

Dodano metodę *onViewCreated()* w celu skonfigurowania obiektu *onClickListener* dla klawisza wywołującego metodę *onButtonFaClicked()* po wykryciu zdarzenia kliknięcia. Ta metoda jest następnie wywoływana, chociaż w tym momencie nic nie robi.

Następną fazą tego procesu jest skonfigurowanie detektora (listener), który pozwoli fragmentowi wywołać działanie po kliknięciu przycisku:

```
package com.amw.prostefragmenty

import android.content.Context
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import com.amw.prostefragmenty.databinding.FragmentPierwszyBinding

class PierwszyFragment : Fragment() {

    private var _binding: FragmentPierwszyBinding? = null
    private val binding get() = _binding!!

    var pierwszyFragmentCallback: PierwszyListener? = null

    interface PierwszyListener {
        fun doFragmentu2(tekst: String)
    }

    override fun onAttach(context: Context) {
        super.onAttach(context)
        try {
            pierwszyFragmentCallback = context as PierwszyListener
        } catch (e: ClassCastException) {
            throw ClassCastException(context.toString()
                    + " trzeba zaimplementować PierwszyListener")
        }
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        _binding = FragmentPierwszyBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.klawiszFa.setOnClickListener { v: View -> onButtonFaClicked(v) }
    }

    private fun onButtonFaClicked(view: View) {
        pierwszyFragmentCallback?.doFragmentu2(binding.tekstFaOut.text.toString())
    }

}
```

Powyższa implementacja spowoduje, że metoda o nazwie *doFragmentu1()* należąca do klasy aktywności zostanie wywołana, gdy użytkownik kliknie klawisz.

Pozostaje zatem zadeklarować, że klasa aktywności implementuje nowo utworzony interfejs PierwszyListener i zaimplementować metodę *onButtonFaClicked()*.

Ponieważ biblioteka Android Support Library jest używana do obsługi fragmentów we wcześniejszych wersjach systemu Android, aktywność musi być zmieniona na podklasę z *FragmentActivity* zamiast *AppCompatActivity*.

Połączenie tych wymagań skutkuje następującym zmodyfikowanym plikiem MainActivity.kt:

```
package com.amw.prostefragmenty

import android.os.Bundle
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import androidx.fragment.app.FragmentActivity
import com.amw.prostefragmenty.databinding.ActivityMainBinding

class MainActivity : FragmentActivity(), PierwszyFragment.PierwszyListener {

    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom)
            insets
        }
    }

    override fun doFragmentu2(tekst: String) {
    }
}
```

Po zmianie kodu pierwszy fragment wykryje, kiedy użytkownik kliknie przycisk i wywoła metodę odczytującą zawartość pola EditText.

Zadaniem aktywności jest komunikacja z drugim fragmentem i przekazanie odczytanej wartości do drugiego fragmentu.

Przekazanie polecenia do drugiego fragmentu

Aktywność może komunikować się z fragmentem poprzez uzyskanie referencji do instancji klasy fragmentu, a następnie wywołanie publicznych metod obiektu.

W związku z tym w klasie DrugiFragment zaimplementujemy teraz metodę publiczną o nazwie *zmienTekst()*, która jako argument przyjmuje ciąg znaków określający nowy tekst, który ma zostać wyświetlony.

Wartość ta zostanie użyta do zmodyfikowania obiektu tekstFbIn. Zmodyfikujemy plik DrugiFragment.kt, aby dodać nową metodę:

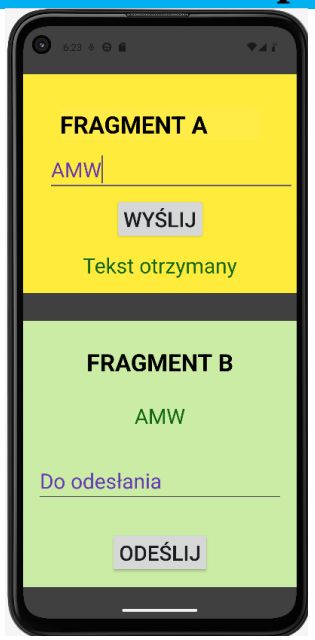
```
package com.amw.prostefragmenty

import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import com.amw.prostefragmenty.databinding.FragmentDrugiBinding

class DrugiFragment : Fragment() {

    private var _binding: FragmentDrugiBinding? = null
    private val binding get() = _binding!!

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        _binding = FragmentDrugiBinding.inflate(layoutInflater, container, false)
        return binding.root
    }

    fun zmienTekst(tekst: String) {
        binding.trekstFbIn.text = tekst
    }

}
```

Korzystając z tego ID, aktywność może uzyskać odwołanie do instancji fragmentu i wywołać metodę *zmienTekst()*.

Zmodyfikujemy plik *MainActivity.kt* file zmieniając metodę *doFragmentu2()*:

```
package com.amw.prostefragmenty

import android.os.Bundle
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import androidx.fragment.app.FragmentActivity
import com.amw.prostefragmenty.databinding.ActivityMainBinding

class MainActivity : FragmentActivity(), PierwszyFragment.PierwszyListener {

    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets -
>
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom)
            insets
        }
    }

    override fun doFragmentu2(tekst: String) {
        var drugiFragment = supportFragmentManager.findFragmentById(R.id.drugiFragment)
as DrugiFragment

        drugiFragment.zmienTekst(tekst)
    }
}
```

Uruchomienie aplikacji



Nawiązywanie kontaktu fragmentu drugiego z aktywnością główną (tak samo jak dla fragmentu pierwszego)

Dokonujemy analogicznych zmian do przesłania danych z fragmentu drugiego do fragmentu pierwszego.

PierwszyFragment.kt

```
package com.amw.prostefragmenty

import android.content.Context
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import com.amw.prostefragmenty.databinding.FragmentPierwszyBinding

class PierwszyFragment : Fragment() {

    private var _binding: FragmentPierwszyBinding? = null
    private val binding get() = _binding!!

    var pierwszyFragmentCallback: PierwszyListener? = null

    interface PierwszyListener {
        fun doFragmentu2(tekst: String)
    }

    override fun onAttach(context: Context) {
        super.onAttach(context)
        try {
            pierwszyFragmentCallback = context as PierwszyListener
        } catch (e: ClassCastException) {
            throw ClassCastException(context.toString()
                    + " trzeba zaimplementować PierwszyListener")
        }
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        _binding = FragmentPierwszyBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.klawiszFa.setOnClickListener { v: View -> onButtonFaClicked(v) }
    }

    private fun onButtonFaClicked(view: View) {
        pierwszyFragmentCallback?.doFragmentu2(binding.tekstFaOut.text.toString())
    }

    fun zmienTekst(tekst: String) {
        binding.tekstFaIn.text = tekst
    }

}
```

DrigiFragment.kt

```
package com.amw.prostefragmenty

import android.content.Context
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import com.amw.prostefragmenty.databinding.FragmentDrugiBinding

class DrugiFragment : Fragment() {

    private var _binding: FragmentDrugiBinding? = null
    private val binding get() = _binding!!

    var drugiFragmentCallback: DrugiListener? = null

    interface DrugiListener {
        fun doFragmentu1(tekst: String)
    }

    override fun onAttach(context: Context) {
        super.onAttach(context)
        try {
            drugiFragmentCallback = context as DrugiListener
        } catch (e: ClassCastException) {
            throw ClassCastException(context.toString()
                    + " trzeba zaimplementować DrugiListener")
        }
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        _binding = FragmentDrugiBinding.inflate(layoutInflater, container, false)
        return binding.root
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.klawiszFb.setOnClickListener { v: View -> onButtonFbClicked(v) }
    }

    private fun onButtonFbClicked(view: View) {
        drugiFragmentCallback?.doFragmentu1(binding.tekstFbOut.text.toString())
    }

    fun zmienTekst(tekst: String) {
        binding.tekstFbIn.text = tekst
    }
}
```

MainActivity.kt

```
package com.amw.prostefragmenty

import android.health.connect.datatypes.units.Length
import android.os.Bundle
import android.widget.Toast
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import androidx.fragment.app.FragmentActivity
import com.amw.prostefragmenty.databinding.ActivityMainBinding

class MainActivity : FragmentActivity(), PierwszyFragment.PierwszyListener,
DrugiFragment.DrugiListener {

    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom)
            insets
        }
    }

    override fun doFragmentu2(tekst: String) {
        var drugiFragment = supportFragmentManager.findFragmentById(R.id.drugiFragment) as
DrugiFragment

        drugiFragment.zmienTekst(tekst)
    }

    override fun doFragmentu1(tekst: String) {
        var pierwszyFragment =
supportFragmentManager.findFragmentById(R.id.pierwszyFragment) as PierwszyFragment

        pierwszyFragment.zmienTekst(tekst)
    }
}
```

Uruchomienie aplikacji