Kotlin & Android w AndroidStudio (3.5.1)

Fragmenty

Fragment

Wprowadzone w Android 3.0 (Honeycomb), obiekty **Fragment** są świetnym narzędziem, które przydaje się w wielu problematycznych sytuacjach, w których dostępne były tylko klasy **Activity**. Fragmenty pozwalają na organizowanie komponentów interfejsu projektu dla różnych urządzeń, dając możliwość wyświetlania wielu segmentów interfejsu na dużym ekranie, np. na tablecie lub wyświetlać jeden i połączyć wszystkie ze sobą na mniejszym ekranie.

Pomagają również podzielić kod na łatwe do zarządzania kawałki, bez potrzeby polegania na dużych i skomplikowanych klasach Activity. Jedną z ostatnich i prawdopodobnie najbardziej wartościową funkcją jest to, że fragmenty pozwalają na łatwe poruszanie się w aplikacji i umożliwiają proste komunikowanie się między różnymi sekcjami aplikacji.

AKTYWNOŚĆ	FRAGMENT

• Posiada własny cykl życia
• **Nie wymaga hostowania**
• Może wykonywać logikę biznesową
• **Jedna aktywność może działać w jednym momencie**
	• Posiada własny cykl życia
	• **Wymagane hostowanie przez aktywność**
	• Może wykonywać logikę biznesową
	• **Jedna aktywność może hostować wiele fragmentów jednocześnie**
	• **Może być dynamicznie zmieniany podczas działania aktywności** (dodawanie, usuwanie, podmiana, itp.)
	• Można implementować na 2 sposoby:
	  ○ Statyczny
	  ○ Dynamiczny

Cykl życia Fragmentu

Podobnie jak klasa Activity, klasa Fragment posiada własny cykl życia, który zarządza Fragment od czasu przyłączenia do Activity, aż do jego zniszczenia. Zrozumienie tego cyklu życia pozwoli tworzyć aplikacje, które są stabilne i wolne od błędów.

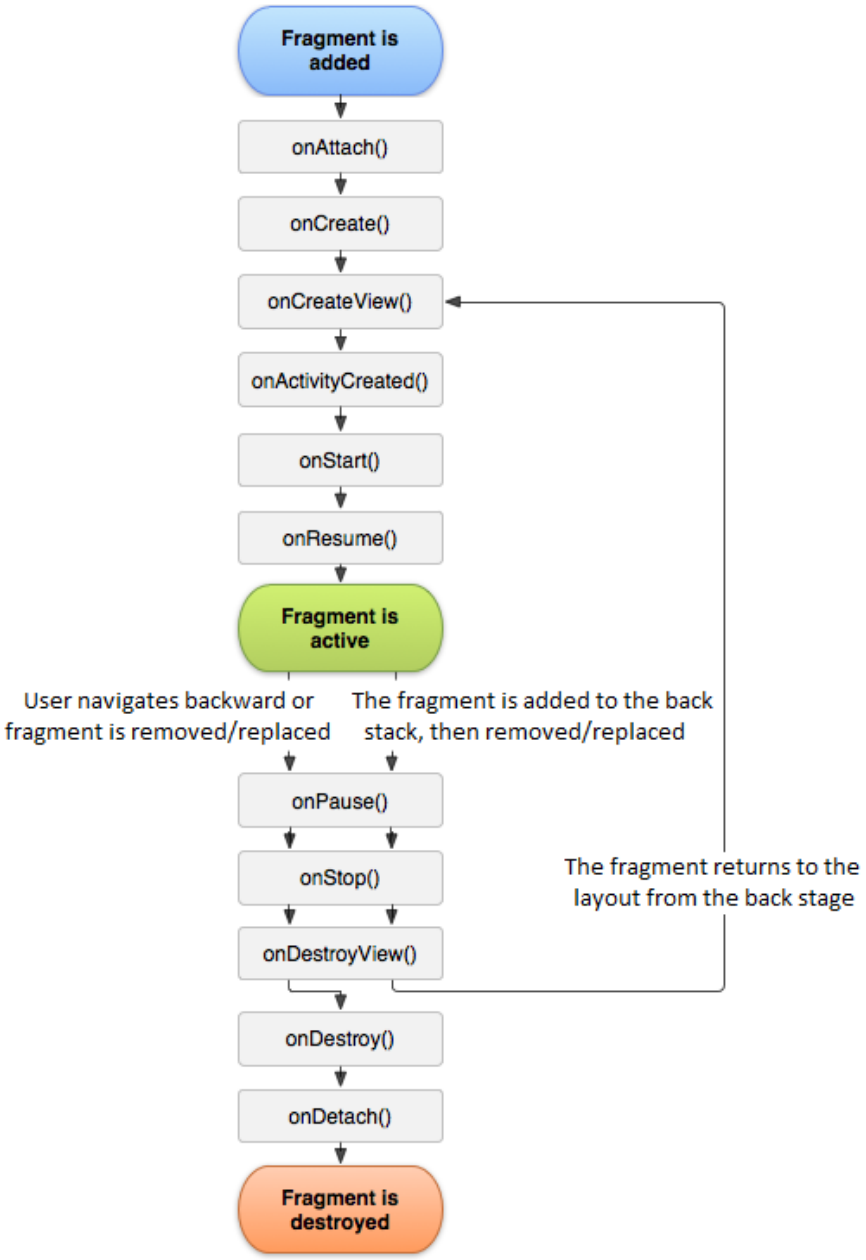


Figure 1. The lifecycle of a fragment (while its activity is running).

- **onAttach()** i **onDetach()** sygnalizuje, kiedy fragment został dodany do Activity i nadaje się do użytku w aplikacji;
- **onCreate()** jest wywoływana jako inicjująca metoda dla klas Fragment, a **onDestroy()** jest odpowiednikiem metody odwrotnej;
- **onCreateView()** jest metodą, w której tworzony jest układ i obiekty View dla fragmentu. **onDestroyView()** jest wywoływana, kiedy hierarchia widoku powiązana z fragmentem jest usuwana.
- **onStart()** i **onStop()** działają podobnie do odpowiedników cyklu życia ich Activity. Te metody są włączane, kiedy Fragment odpowiednio staje się lub przestaje być widoczny.
- **onPause()** i **onStart()** są również podobne do odpowiedników Activity. Kiedy Fragment jest widoczny, ale jego koncentracja się zmieniła, wywołuje się jedną z tych dwóch metod.

Tworzenie aplikacji wykorzystującej fragmenty

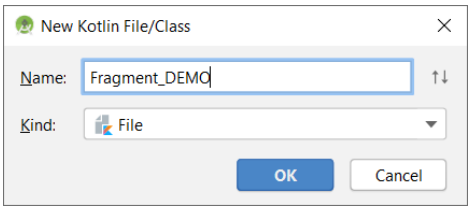
Statyczna implementacja fragmentów

W tej sekcji utworzymy prostą aplikację wykorzystującą statyczną implementację fragmentów. Nasza aplikacja będzie zbudowana z jednej aktywności i dwóch fragmentów, a jej głównym celem będzie zademonstrowanie czym są fragmenty, jak działają i jak ich używać.

ZADANIE

Utwórz nowy projekt Android Studio o nazwie **FragmentApp**, zawierający **EmptyActivity**.

W projekcie utwórz nowy plik Kotlin o nazwie **Fragment_DEMO**. W tym celu kliknij PPM na pakiet zawierający MainActivity i dalej **New -> Kotlin File / Class**.



Zaimplementuj kod klasy fragmentu:

```
package android.myapplication

import androidx.fragment.app.Fragment

class Fragment_DEMO: Fragment(){

}
```

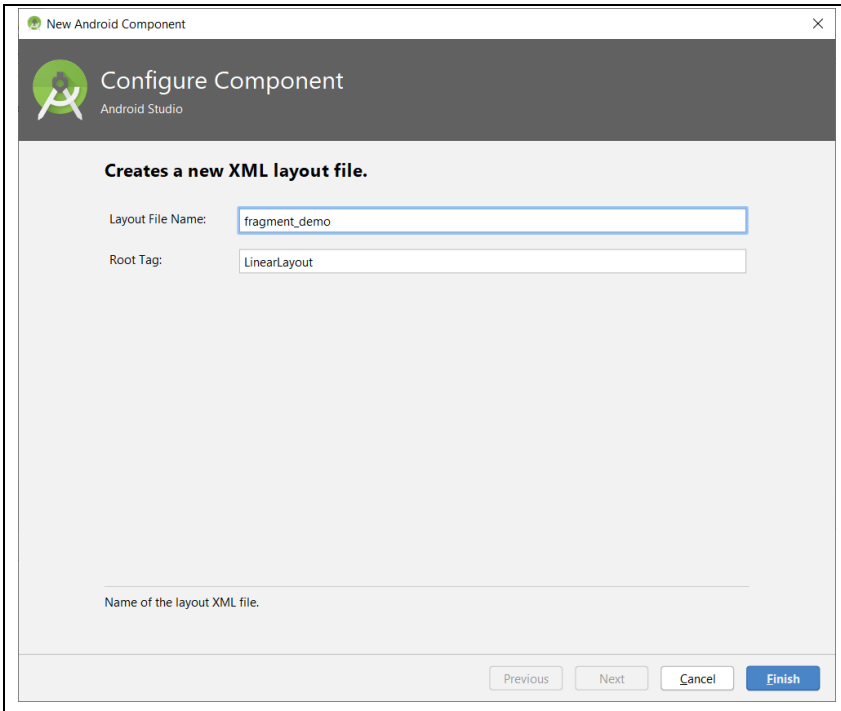
Analogicznie utwórz plik Kotlin o nazwie **Fragment_DEMO2** i napisz w nim kod:

```
package android.fragmentapp

import androidx.fragment.app.Fragment

class Fragment_DEMO: Fragment(){

}
```

Utworzymy teraz dwa layouty, które będą wykorzystywane przez nasze fragmenty.

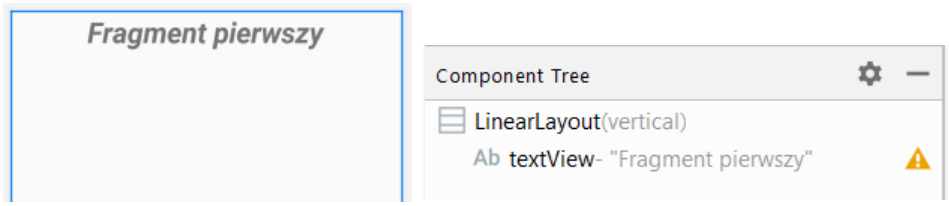
ZADANIE

Dodaj do projektu plik layoutu i nazwij go **fragment_demo**. Jako Root wskaż **Linear Layout** (vertical).

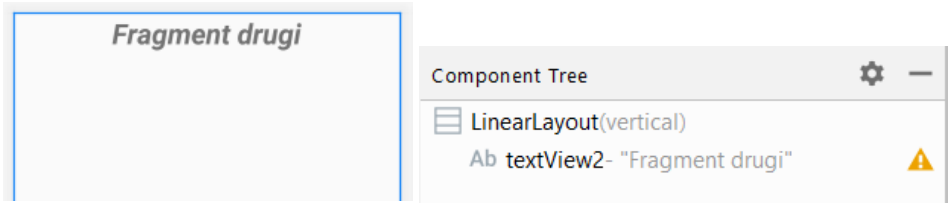


Ustaw **layout_height** na **wrap_content**.

Dodaj do utworzonego layoutu element **TextView** i wprowadź dla niego następujące ustawienia: gravity = center; textSize = 30sp; textStyle = bold, italic; text = „Fragment pierwszy”.



Analogicznie utwórz plik layoutu dla drugiego fragmentu i nazwij go **fragment_demo2**. Również do tego layoutu wstaw **TextView** o tych samych ustawieniach atrybutów (oprócz tekstu). Ustaw tekst „Fragment drugi”.



Teraz rozwiniemy kod naszych fragmentów.

ZADANIE

Przejdź do pliku **Fragment_DEMO** i wprowadź zmiany:

```
package android.fragmentapp

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

class Fragment_DEMO: Fragment(){
    // Tworzenie ("rozdmuchiwanie") widoku fragmentu
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle? ): View? {
        return inflater.inflate(R.layout.fragment_demo, container,  attachToRoot: false)
    }
}
```

Ten sam kod umieść w **Fragment_DEMO2** z tym, że zamiast **R.layout.fragment_demo** użyj **R.layout.fragment_demo2**.

Dalej dodajemy utworzone fragmenty do głównej aktywności.

ZADANIE

Przejdź do pliku layoutu głównej aktywności. Usuń **ConstraintLayout** i wstaw **LinearLayout**(vertical).

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
</LinearLayout>
```

Dodaj do utworzonego layoutu nasze fragmenty:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <fragment
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:name="android.fragmentapp.Fragment_DEMO"
        android:id="@+id/fragment1"/>

    <fragment
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:name="android.fragmentapp.Fragment_DEMO2"
        android:id="@+id/fragment2"/>

</LinearLayout>
```

Uruchom aplikacje.

Nasza aplikacja działa prawidłowo wyświetlając jedną aktywność zawierającą dwa fragmenty.

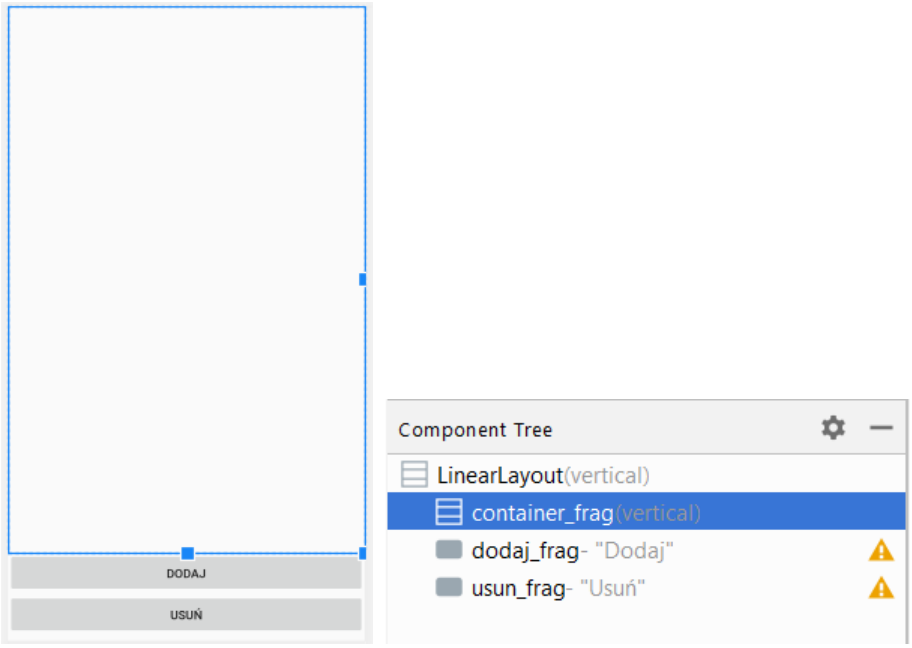
Dynamiczna implementacja fragmentów

W tej sekcji rozwiniemy naszą aplikację w taki sposób, aby można było dodawać i usuwać fragmenty w trakcie jej działania. Zadanie to wymaga dynamicznej implementacji fragmentów, ponieważ liczba wyświetlanych fragmentów w naszej aplikacji będzie się zmieniać.

Zaczynamy od utworzenia layoutów naszej aplikacji. Na początek modyfikujemy layout głównej aktywności.

ZADANIE

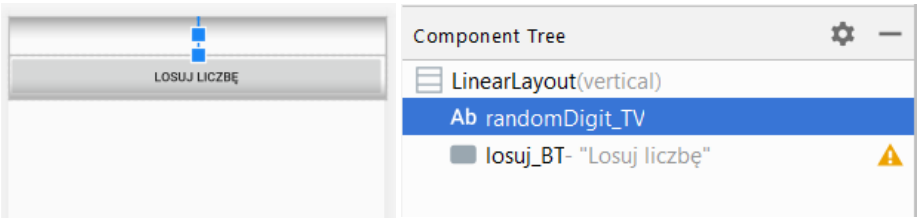
Przejdź do pliku layoutu głównej aktywności. Zmodyfikuj jego wygląd stosując **LinearLayout**(vertical), w którym umieszczone będą: **LinearLayout**(vertical) stanowiący „pojemnik" na nasze fragmenty oraz 2 przyciski **Button** do dodawania i usuwania fragmentów.



Następnie zmieniamy wygląd layoutu **fragment_DEMO**.

ZADANIE

Zmodyfikuj layout fragmentu stosując **LinearLayout**(vertical), w którym umieszczone będą: **TextView** oraz **Button**:



Teraz zajmiemy się oprogramowaniem funkcjonalności naszej aplikacji, tzn. dynamicznym zarządzaniem wyświetlanymi fragmentami.

ZADANIE

Przejdź do **MainActivity** i wprowadź zmiany w metodzie **onCreate()**.

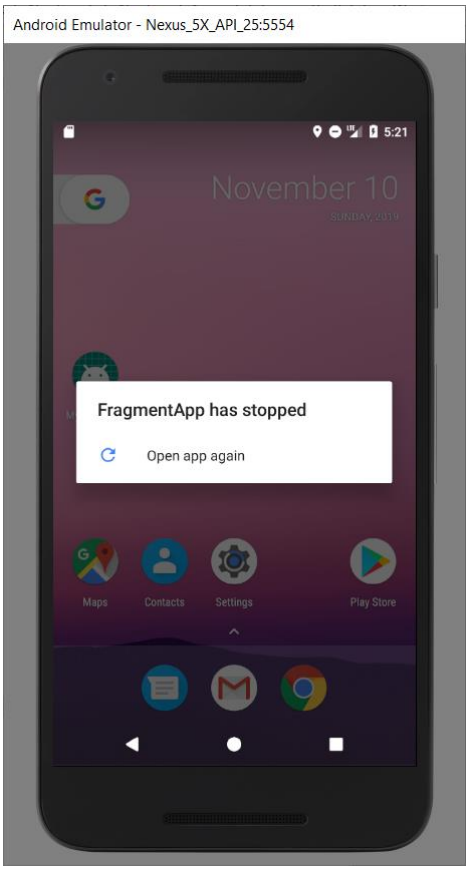
```
package android.fragmentapp

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import androidx.fragment.app.FragmentManager
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {

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

        // obiekt do zarzadzania fragmentami
        val fm:FragmentManager = supportFragmentManager
        val fragmentDemo = Fragment_DEMO()

        // dodawanie fragmentow
        dodaj_frag.setOnClickListener { it: View!
            // wywolanie transakcji dodajacej fragment i jej zatwierdzenie
            fm.beginTransaction().add(R.id.container_frag, fragmentDemo).commit()
        }
        // usuwanie fragmentow
        usun_frag.setOnClickListener { it: View!
            // wywolanie transakcji usuwajacej fragment i jej zatwierdzenie
            fm.beginTransaction().remove(fragmentDemo).commit()
        }
    }
}
```

Uruchom aplikację i przetestuj jej działanie.

Po uruchomieniu aplikacji widzimy aktywność główną z dwoma przyciskami ale bez wyświetlonego fragmentu:

Klikając na przycisk **Dodaj** na ekranie pojawia się nasz fragment (na razie bez działającego przycisku do losowania liczb):

Klikając **Usuń** fragment znika. Natomiast klikając kilkukrotnie na **Dodaj** nasza aplikacja zostaje zatrzymana z powodu błędu:



Dzieje się tak, ponieważ próbujemy kilkukrotnie uruchomić nasz fragment, co nie jest możliwe. Aby zapobiec tej sytuacji dodajmy stosowne zabezpieczenie w kodzie.

ZADANIE

Zmodyfikuj kod przycisku dodawania fragmentu w następujący sposób:

```
dodaj_frag.setOnClickListener { it: View!
    if(fragmentDemo.isAdded){ // sprawdzenie czy fragment zostal juz dodany do aktywnosci
        Toast.makeText(applicationContext, text: "Fragment jest już aktywny!", Toast.LENGTH_SHORT).show()
    }
    else {
        // wywolanie transakcji dodajacej fragment i jej zatwierdzenie
        fm.beginTransaction().add(R.id.container_frag, fragmentDemo).commit()
    }
}
```

Sprawdź jak teraz działa aplikacja.

Teraz w momencie dwukrotnego kliknięcia na **Dodaj** pojawia się komunikat **Toast**.

Pozostaje nam już tylko oprogramować sam fragment, a dokładniej dodać akcję dla przycisku losowania liczby.

ZADANIE

Zaimplementuj kod przycisku naszego fragmentu.

```
class Fragment_DEMO: Fragment(){
    // Tworzenie ("rozdmuchiwanie") widoku fragmentu
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle? ): View? {
        return inflater.inflate(R.layout.fragment_demo, container,  attachToRoot: false)
    }

    override fun onStart() {
        super.onStart()
        losuj_BT.setOnClickListener { it: View!
            val randNumber = Random.nextInt( until: 100)
            randomDigit_TV.setText(randNumber.toString())
        }
    }
}
```

Przetestuj jak działa aplikacja.

Cofnięcie wywołania fragmentu

Nasza aplikacja umożliwia dodanie fragmentu do aktywności i jego usunięcie. Zauważmy jednak, że po dodaniu fragmentu i kliknięciu przycisku funkcyjnego **Wstecz** nasza aplikacja kończy swoje działanie.

W przypadku aktywności jest inaczej. Jeżeli mamy aplikację zbudowaną np. z 2 aktywności **A1** i **A2**, gdzie **A2** jest wywoływana z **A1**, to będąc w **A2** i klikając **Wstecz** przenosimy się do **A1**. Cofamy wywołanie aktywności **A2** i nasza aplikacja działa dalej. Dzieje się tak ponieważ aktywności „odkładane są" na stosie. W momencie uruchomienia aplikacji na stosie mamy tylko **A1**. Jeżeli teraz wywołamy drugą aktywność to na szczycie stosu znajdzie się **A2**, zaś następna będzie aktywność **A1**. Wychodząc z **A2** - „ściągając" ze stosu – na jego szczycie pozostaje **A1**.

Chcemy teraz, aby nasz fragment zachowywał się podobnie. Wprowadźmy więc stosowne zmiany, które polegają na „odkładaniu" fragmentu na stosie.

ZADANIE

Wprowadź zmiany w kodzie obsługi przycisku dodawania fragmentu.

```
dodaj_frag.setOnClickListener { it: View!
    if(fragmentDemo.isAdded){ // sprawdzenie czy fragment zostal juz dodany do aktywnosci
        Toast.makeText(applicationContext,  text: "Fragment jest już aktywny!", Toast.LENGTH_SHORT).show()
    }
    else {
        // wywolanie transakcji dodajacej fragment i jej zatwierdzenie
        fm.beginTransaction().add(R.id.container_frag, fragmentDemo).addToBackStack( name: null).commit()
    }
}
```

Sprawdź jak teraz zachowuje się nasza aplikacja.

Po dodaniu fragmentu i kliknięciu **Wstecz** wyświetlany fragment znika. Klikając drugi raz na **Wstecz** wychodzimy z aplikacji.

Wykonajmy teraz ciąg akcji polegający na dodaniu fragmentu, usunięciu go, kolejnym dodaniu, kolejnym usunięciu i znów dodaniu. Następnie kliknijmy **Wstecz**. Wyświetlony fragment znika – prawidłowo. Kliknijmy powtórnie **Wstecz**. Tym razem nie widzimy żadnego rezultatu, a przecież jakby mogło się wydawać – powinniśmy wyjść z aplikacji. Kliknijmy ponownie **Wstecz** – dalej nic. Kliknijmy jeszcze raz **Wstecz** – dopiero teraz wychodzimy z aplikacji.

Dlaczego tak się dzieje? Otóż wykonaliśmy akcję polegającą na 3-krotnym dodaniu fragmentu i 2-krotnym jego usunięciu. W momencie dodawania fragmentu zostawał on „odłożony” na stos. Zatem na stosie mieliśmy naszą aktywność na samym dole, a na niej 3 fragmenty. Dlatego właśnie trzeba było 4-krotnie kliknąć **Wstecz** aby opróżnić stos i zakończyć działanie aplikacji.

Spróbujmy temu zaradzić dodając dla przycisku **Usuń** akcję usunięcia fragmentu ze stosu.

ZADANIE

Wprowadź zmiany w kodzie obsługi przycisku usuwania fragmentu.

```
usun_frag.setOnClickListener { it: View!
    // wywolanie transakcji usuwajacej fragment i jej zatwierdzenie
    fm.beginTransaction().remove(fragmentDemo).commit()
    // sciagniecie ze stosu fragmentu
    fm.popBackStack()
}
```

Sprawdź działanie aplikacji.

Aplikacja działa poprawnie. Aby jednak kod wyglądał lepiej i akcja „ściągania” fragmentu ze stosu nie była wykonywana gdy fragment nie jest aktywny (możemy przecież wielokrotnie klikać na **Usuń**) dodajmy zabezpieczenie.

ZADANIE

Wprowadź poniższe zmiany w kodzie.

```
usun_frag.setOnClickListener { it: View!
    // wywolanie transakcji usuwajacej fragment i jej zatwierdzenie
    fm.beginTransaction().remove(fragmentDemo).commit()
    // sciagniecie ze stosu fragmentu
    if(fragmentDemo.isAdded)
        fm.popBackStack()
}
```

Nasza aplikacja wykorzystująca fragmenty jest już gotowa.