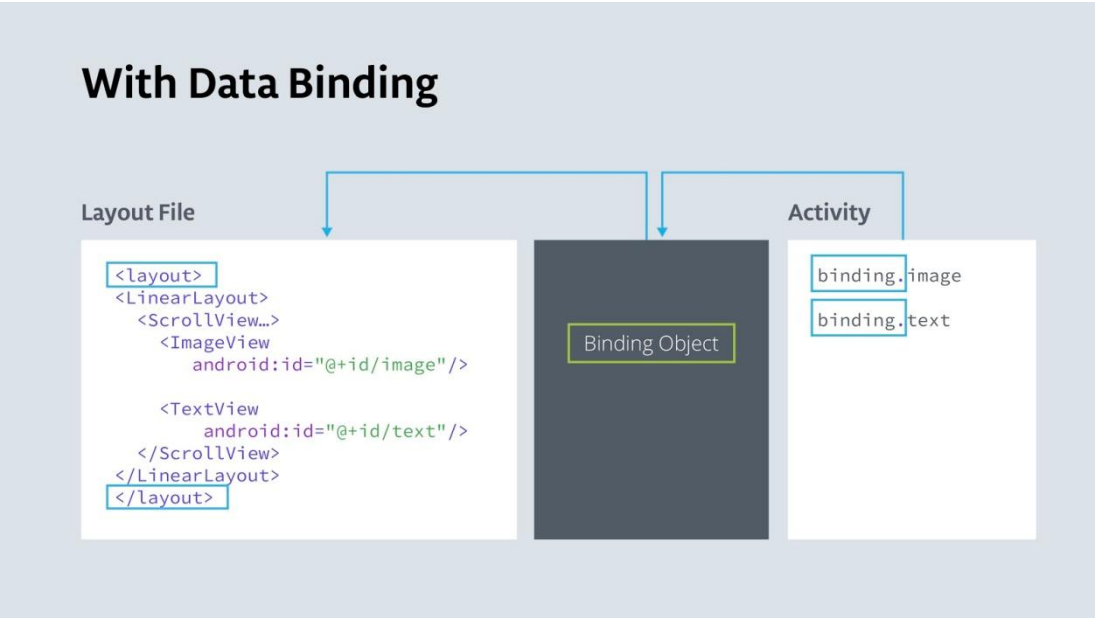
Zmodyfikuj aplikację, aby korzystała z powiązania danych (Data Binding) zamiast findViewById () i aby uzyskać dostęp do danych bezpośrednio z plików XML układu.

3. Zadanie: Użyj powiązania danych, aby wyeliminować findViewById ()

Kod napisany w poprzednich kodelabach używa funkcji findViewById () do uzyskiwania referencji do widoków.
Za każdym razem, gdy używasz findViewById () do wyszukiwania widoku po jego utworzeniu lub odtworzeniu, system Android przegląda hierarchię widoku w czasie wykonywania, aby go znaleźć.

Gdy Twoja aplikacja ma tylko kilka widoków, nie stanowi to problemu. Jednak aplikacje produkcyjne mogą mieć dziesiątki widoków w układzie, a nawet przy najlepszym projekcie będą zagnieżdżone widoki.

Pomyśl o układzie liniowym, który zawiera widok przewijania, który zawiera widok tekstu. W przypadku dużej lub głębokiej hierarchii widoku znalezienie widoku może zająć wystarczająco dużo czasu, aby wyraźnie spowolnić aplikację dla użytkownika. Buforowanie widoków w zmiennych może pomóc, ale nadal musisz zainicjować zmienną dla każdego widoku w każdej przestrzeni nazw. Z dużą ilością widoków i wieloma aktywnościami, to też się składa.
Jednym z rozwiązań jest utworzenie obiektu zawierającego odwołanie do każdego widoku. Ten obiekt, zwany obiektem Binding, może być używany przez całą aplikację. Ta technika nazywa się wiązaniem danych. Po utworzeniu obiektu powiązania dla aplikacji można uzyskać dostęp do widoków i innych danych za pośrednictwem obiektu powiązania, bez konieczności przechodzenia przez hierarchię widoków lub wyszukiwania danych.



Powiązanie danych ma następujące zalety:

- Kod jest krótszy, łatwiejszy do odczytania i łatwiejszy w utrzymaniu niż kod korzystający z `findByView()`.
- Dane i widoki są wyraźnie oddzielone.
- System Android przegląda hierarchię widoków tylko raz, aby uzyskać każdy widok, i dzieje się to podczas uruchamiania aplikacji, a nie w czasie wykonywania, gdy użytkownik wchodzi w interakcję z aplikacją.
- Otrzymujesz bezpieczeństwo typu dla dostępu do widoków. (Bezpieczeństwo typu oznacza, że kompilator sprawdza poprawność typów podczas kompilacji i generuje błąd, jeśli spróbujesz przypisać zły typ do zmiennej).

W tym zadaniu konfigurujesz powiązanie danych i używasz powiązania danych (data binding), aby zastąpić wywołania findViewById () wywołaniami obiektu powiązania (binding object).

Krok 1: Włącz powiązanie danych (Enable data Winding)

Aby użyć powiązania danych, musisz włączyć powiązanie danych w pliku Gradle, ponieważ domyślnie nie jest ono włączone. Wynika to z faktu, że wiązanie danych wydłuża czas kompilacji i może wpływać na czas uruchamiania aplikacji.

1. Otwórz plik `build.gradle (Module: app)`
2. W sekcji Androida przed nawiasem zamykającym dodaj sekcję dataBinding i ustaw wartość enabled na true.

```
dataBinding {
    enabled = true
}
```

4. Po wyświetleniu monitu zsynchronizuj projekt. Jeśli nie pojawi się monit, wybierz polecenie **File > Sync Project with Gradle Files**.
5. Możesz uruchomić aplikację, ale nie zobaczysz żadnych zmian.

Krok 2: Zmień plik layout , aby był dostępny z powiązaniem danych

Aby pracować z wiązaniem danych, musisz owinąć swój układ XML tagiem <layout>. Dzieje się tak, że klasa główna nie jest już grupą widoków, ale jest układem zawierającym grupy widoków i widoki. Obiekt powiązania może następnie wiedzieć o układzie i widokach w nim zawartych.

1. Otwórz plik `activity_main.xml`
2. Przejdź do zakładki **Text**.
3. Dodaj `<layout></layout>` jako najbardziej zewnętrzny znacznik wokół `<LinearLayout>`.

```
<layout>
   <LinearLayout ... >
   ...
   </LinearLayout>
</layout>
```

4. Wybierz **Code > Reformat code** aby uporządkować wcięcia kodu.

   Deklaracje przestrzeni nazw dla układu muszą znajdować się w najbardziej zewnętrznym znaczniku.

5. Wytnij deklaracje przestrzeni nazw z `<LinearLayout>` i wklej je do znacznika `<layout>` tag. Your opening `<layout>` powinien wyglądać tak, jak pokazano poniżej,`<LinearLayout>` powinien zawierać tylko właściwości widoku.

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto">
```

6. Zbuduj i uruchom aplikację, aby sprawdzić, czy zrobiłeś to poprawnie.

Krok 3: Utwórz binding object w main activity

Dodaj odwołanie do binding object do main activity, so aby można było użyć go do uzyskania dostępu do widoków:

1. W pliku `MainActivity.kt`.
2. Przed `onCreate()`,na najwyższym poziomie, utwórz zmienną dla obiektu powiązania. Ta zmienna jest zwyczajowo nazywana `binding`.

   Typ zmiennej `binding`, klasa `ActivityMainBinding`, jest tworzony przez kompilator specjalnie dla tego main activity. Nazwa pochodzi od nazwy pliku layout: `activity_main + Binding`.

```
private lateinit var binding: ActivityMainBinding
```

3. Na żądanie Android Studio zaimportuj `ActivityMainBinding`. Jeśli nie zostanie wyświetlony monit, kliknij `ActivityMainBinding` a potem `Alt+Enter` aby zaimportować tę brakującą klasę. (

   Instrukcja importu powinna wyglądać podobnie do poniższej..

```
import com.example.android.aboutme.databinding.ActivityMainBinding
```

Następnie zastąpisz bieżącą funkcję `setContentView()` instrukcją, która wykonuje następujące czynności:

- Tworzy obiekt wiązania(binding object).
- Używa funkcji `setContentView()` z klasy `DataBindingUtil` do powiązania layoutu `activity_main` z `MainActivity`. Ta funkcja `setContentView()` zajmuje się również niektórymi ustawieniami powiązania danych dla widoków.

4. W funkcji `onCreate()`,zamień wywołanie `setContentView()` na następujący wiersz kodu.

```
binding = DataBindingUtil.setContentView(this, R.layout.activity_main)
```

5. Importuj `DataBindingUtil`.

```
import androidx.databinding.DataBindingUtil
```

Krok 4: Użyj obiektu powiązania, aby zastąpić wszystkie wywołania findViewById ()

Teraz możesz zastąpić wszystkie wywołania funkcji findViewById () odwołaniami do widoków znajdujących się w obiekcie powiązania. Podczas tworzenia obiektu powiązania kompilator generuje nazwy widoków w obiekcie powiązania na podstawie identyfikatorów widoków w układzie.

Na przykład, `done_button` jest `doneButton` binding object dla , `nickname_edit` staje sie `nicknameEdit`, i `nickname_text` teraz jest `nicknameText`.

1. W `onCreate()`,zamień kod, który używa`findViewById()` aby znaleźć przycisk `done_button`

   dotychczasowy kod: `findViewById<Button>(R.id.done_button)`
   zastąp: `binding.doneButton`

   Gotowy kod do ustawienia click listener w `onCreate()` powinien wyglądać tak:.

```
binding.doneButton.setOnClickListener {
   addNickname(it)
}
```

2. Zrób to samo dla wszystkich wywołań`findViewById()` w funkcji `addNickname()`
3. Zamień wszystkie wystąpienia `findViewById<View>(R.id.id_view)` na `binding.idView`. Zrób to w następujący sposób:

- Usuń definicje zmiennych `editText` i `nicknameTextView` wraz z ich wywołaniami `findViewById()`.to spowoduje błędy
- Napraw błędy, uzyskując widoki `nicknameText`, `nicknameEdit`, i `doneButton` z obiektu `binding` zamiast (usuniętych) zmiennych.
- Zamień `view.visibility` na `binding.doneButton.visibility`. Użycie `binding.doneButton` zamiast przekazanego widoku powoduje, że kod jest bardziej spójny.

  Wynikiem jest następujący kod:

```
binding.nicknameText.text = binding.nicknameEdit.text
binding.nicknameEdit.visibility = View.GONE
binding.doneButton.visibility = View.GONE
binding.nicknameText.visibility = View.VISIBLE
```

- There is no change in functionality. Optionally, you can now eliminate the `view` parameter and update all uses of `view` to use `binding.doneButton` inside this function.

3. NicknameText wymaga `String`, a nicknameEdit.text jest `Editable`. Podczas korzystania z powiązania danych należy jawnie przekonwertować .`Editable` na `String`.

```
binding.nicknameText.text = binding.nicknameEdit.text.toString()
```

4. Możesz usunąć wyszarzony import.
5. Kotlinize (Możesz "Kotlinizować" funkcję za pomocą `apply{}`.

```
binding.apply {
   nicknameText.text = nicknameEdit.text.toString()
   nicknameEdit.visibility = View.GONE
   doneButton.visibility = View.GONE
   nicknameText.visibility = View.VISIBLE
}
```

6. Zbuduj i uruchom aplikację ... i powinna wyglądać i działać dokładnie tak samo jak poprzednio.

Wskazówka: Jeśli po wprowadzeniu zmian zobaczysz błędy kompilatora, wybierz polecenie **Build > Clean Project** a następnie **Build > Rebuild Project**. Wykonanie tego zazwyczaj aktualizuje wygenerowane pliki. W przeciwnym razie wybierz **File > Invalidate Caches/Restart** aby przeprowadzić dokładniejsze czyszczenie.

Tip: Wskazówka: wcześniej dowiedziałeś się o obiekcie `Resources` który zawiera odwołania do wszystkich zasobów w aplikacji. Możesz myśleć o obiekcie `binding` w podobny sposób, odwołując się do widoków; obiekt `binding` jest jednak znacznie bardziej wyrafinowany..

4. Zadanie: Użyj data binding, aby wyświetlić dane

Możesz skorzystać z powiązania danych, aby udostępnić klasę danych bezpośrednio do widoku. Ta technika upraszcza kod i jest niezwykle cenna do obsługi bardziej złożonych przypadków.

W tym przykładzie zamiast ustawiania nazwy i pseudonimu za pomocą zasobów łańcuchowych, tworzysz klasę danych dla nazwy i pseudonimu. Udostępniasz klasę danych w widoku za pomocą powiązania danych.

Krok 1: Utwórz klasę danych (MyName data class)

1. W Android Studio w katalogu java otwórz plik MyName.kt. Jeśli nie masz tego pliku, utwórz nowy plik Kotlin i nazwij go MyName.kt.
2. Zdefiniuj klasę danych dla imienia i pseudonimu. Użyj pustych łańcuchów jako wartości domyślnych.

```
data class MyName(var name: String = "", var nickname: String = "")
```

Krok 2: Dodaj dane do layout

W pliku `activity_main.xml` nazwa jest obecnie ustawiona w `TextView` z zasobu string. Należy zastąpić odwołanie do nazwy odwołaniem do danych w klasie danych.

1. Otwórz `activity_main.xml` w zakładce **Text** .
2. W górnej części układu, między znacznikami `<layout>` i `<LinearLayout>` wstaw znacznik `<data></data>` tag. Tutaj połączysz widok z danymi.

```
<data>

</data>
```

Wewnątrz znaczników danych można zadeklarować zmienne nazwane, które zawierają odwołanie do klasy.

3. Wewnątrz znacznika `<data>` dodaj znacznik `<variable>`
4. Dodaj parametr `name`, aby nadać zmiennej nazwę `"myName"`. Dodaj parametr `type` i ustaw typ na w pełni kwalifikowaną nazwę klasy danych `MyName` data class (nazwa pakietu + nazwa zmiennej).

```
<variable
       name="myName"
       type="com.example.android.aboutme.MyName" />
```

Teraz zamiast używać zasobu string dla nazwy, możesz odwoływać się do zmiennej `myName`.

5. Zamień `android:text="@string/name"` na kod:.

`@={}` is to dyrektywa służąca do pobierania danych, do których istnieją odniesienia w nawiasach klamrowych.
`myName` odwołuje się do zdefiniowanej wcześniej zmiennej myName, która wskazuje na klasę danych `myName` i pobiera właściwość name z klasy.

```
android:text="@={myName.name}"
```

Krok 3: Utwórz dane

Masz teraz odniesienie do danych w pliku layout. Teraz tworzysz aktualne dane.

1. Otwórz `MainActivity.kt`.
2. Powyżej `onCreate()`,utwórz zmienną prywatną, zwaną również konwencją myName. Przypisz zmiennej instancję klasy danych MyName, przekazując nazwę.

```
private val myName: MyName = MyName("Aleks Haecky")
```

3. W funkcji `onCreate()`,ustaw wartość zmiennej myName w pliku layout na wartość właśnie zadeklarowanej zmiennej myName. Nie możesz uzyskać dostępu do zmiennej w pliku XML bezpośrednio. Musisz uzyskać do niego dostęp poprzez obiekt powiązania.

```
binding.myName = myName
```

4. Może to oznaczać błąd, ponieważ po dokonaniu zmian należy odświeżyć obiekt powiązania. Zrestartuj aplikację, a błąd powinien zniknąć..

Krok 4: Użyj data class dla nickname w TextView

Ostatnim krokiem jest użycie klasy danych dla pseudonimu w `TextView`.

1. Otwórz `activity_main.xml`.
2. W widoku `nickname_text` dodaj właściwość `text` property. Odwołaj się do `nickname`, jak pokazano poniżej.

```
android:text="@={myName.nickname}"
```

3. W `ActivityMain`, zamień
   `nicknameText.text = nicknameEdit.text.toString()`
   z kodem do ustawienia pseudonimu w zmienne `myName`

```
myName?.nickname = nicknameEdit.text.toString()
```

Po ustawieniu pseudonimu chcesz, aby kod odświeżał interfejs użytkownika nowymi danymi. W tym celu należy unieważnić wszystkie wyrażenia wiążące, aby zostały one odtworzone z poprawnymi danymi.

4. Dodaj `invalidateAll()` po ustawieniu pseudonimu, aby interfejs użytkownika został odświeżony wartością w zaktualizowanym binding object.

```
binding.apply {
   myName?.nickname = nicknameEdit.text.toString()
   invalidateAll()
   ...
}
```

5. Zbuduj i uruchom aplikację, która powinna działać dokładnie tak samo jak poprzednio.