Практическая работа. Binding. Связывание представлений в коде фрагментов

Фрагменты тоже могут использовать связывание представлений, но код выглядит немного иначе

Итак, вы узнали, как реализовать связывание представлений в коде активности. Давайте посмотрим, как использовать его во фрагментах. Как было сказано ранее, код фрагментов несколько отличается от кода активностей.

Реализуем связывание представлений в приложении Bits and Pizzas

Чтобы понять, как работает связывание представлений для фрагментов, мы внесем изменения в приложение `Bits and Pizzas`.

Откройте проект этого приложения. Как вы, вероятно, помните, приложение `Bits and Pizzas` использует представления Material, такие как сворачиваемые панели инструментов и FAB, для построения интерактивного пользовательского интерфейса. Пользовательский интерфейс определяется фрагментом приложения, который называется `FragmentOrder`.

Код `FragmentOrder` включает множество вызовов `indViewById()`. Давайте посмотрим, как использовать связывание представлений для их замены.

Включение связывания представлений для Bits and Pizzas

Как и прежде, чтобы использовать связывание представлений в проекте `Bits and Pizzas`, необходимо сначала разрешить его использование в файле `build.gradle` приложения. Откройте файл `BitsandPizzas/app/build.gradle` и включите следующие строки в раздел android:

```
android {
 ...
 buildFeatures {
 viewBinding true
 }
}
```

Затем выберите команду `Sync Now`, чтобы синхронизировать изменения с остальными частями проекта.

Классы свзывания генерируются для каждого макета

Как и в приложении `Stopwatch`, при включении связывания представлений в приложение `Bits and Pizzas` для каждого файла макета автоматически создается класс связывания. Приложение содержит два макета: `activity_main.xml` и `fragment_order.xml`, поэтому генерируются два класса связывания: `ActivityMainBinding` и `FragmentOrderBinding`.

Как и прежде, каждый класс связывания включает свойство для каждого представления с идентификатором в его макете. Например, макет `fragment_order.xml` включает группу

переключателей с идентификатором `pizza_group`, поэтому класс связывания `FragmentOrderBinding` включает свойство с именем `pizzaGroup` и типом `RadioGroup`. В приложении `Bits and Pizzas` с представлениями взаимодействует только код `OrderFragment.kt`, так что нам остается только обновить этот файл для использования связывания представлений.

Код связывания представлений для фрагментов выглядит немного иначе

Как мы уже говорили, код связывания представлений, используемый во фрагментах, несколько отличается от кода, используемого в активностях. Но прежде чем показывать, как выглядит код связывания представлений во фрагментах, мы поглубже разберемся в теме и выясним, почему же код отличается.

Активности могут обращаться к представлениям из `onCreate()`

Как вам уже известно, активность впервые получает доступ к макету при выполнении ее метода `onCreate()`. Это первый метод жизненного цикла активности, и он используется для заполнения макета (или связывания его с новым представлением) и выполнения всей начальной настройки. Например, если кнопка должна реагировать на щелчки, активность назначает ей слушатель `OnClickListener` в своем методе `onCreate()`. Активность сохраняет доступ к своему макету, пока не будет вызван ее метод `onDestroy()`. Это последний метод в жизненном цикле активности, и после того как он будет выполнен, активность уничтожается. Так как активность сохраняет доступ к представлениям своего макета от `onCreate()` до `onDestroy()`, она может взаимодействовать с ними в любых своих методах:

Все это происходит с активностями, а с фрагментами дело обстоит немного иначе — давайте посмотрим, в чем именно.

Фрагменты могут обращаться к представлениям от `onCreateView()` до `onDestroyView()`

Как вы знаете, фрагмент впервые получает доступ к своему макету при выполнении его метода `onCreateView()`. Он вызывается, когда активности потребуется доступ к макету фрагмента, поэтому он используется для заполнения макета и выполнения всей исходной настройки (например, назначения слушателей `OnClickListener`). Тем не менее `onCreateView()` — не первый метод в жизненном цикле фрагмента. Существуют и другие методы, выполняемые до `onCreateView()` (например, `onCreate()`), и они не могут взаимодействовать с представлениями фрагмента. После выполнения метода `onCreateView()` фрагмент сохраняет доступ к своим представлениям до того, как завершится выполнение его метода `onDestroyView()`. Метод вызывается, когда макет фрагмента становится ненужным для активности, например потому что активности нужно перейти к другому фрагменту или он уничтожается. Однако `onDestroyView()` — не последний метод в жизненном цикле фрагмента. Существуют и другие методы, выполняемые после `onDestroyView()` (например, `onDestroy()`), и они тоже не могут взаимодействовать с представлениями фрагмента:

Схема жизненного цикла фрагмента приведена на следующей странице. Затем вы увидите, как выглядит код связывания представлений для фрагмента.

Жизненный цикл фрагмента очень похож на жизненный цикл активности, но он содержит ряд дополнительных фаз. На следующей схеме показано, когда вызываются методы жизненного цикла главного фрагмента, для чего они используются и какое место они занимают в состоянии активности, в которой отображается фрагмент:

Состояние активности	Метод жизненного цикла фрагмента
Создается	onCreate(Bundle). Вызывается при создании фрагмента. Может использоваться для любой исходной настройки, в которой не задействованы представления.
	onCreateView(LayoutInflater, ViewGroup, Bundle). Здесь фрагмент впервые получает доступ к своим представлениям.
Запускается	onStart(). Вызывается, когда фрагмент готовится стать видимым.
Продолжает выполнение	onResume(). Вызывается, когда фрагмент начинает активно выполняться.
Приостанавливается	onPause(). Вызывается, когда фрагмент перестает активно выполняться.
Останавливается	onStop(). Вызывается, когда фрагмент перестает быть видимым.
Уничтожается	onDestroyView(). Вызывается, когда иерархия представлений фрагмента готовится к уничтожению, чтобы фрагмент мог освободить ресурсы.
	onDestroy(). Вызывается перед уничтожением фрагмента. В этой точке могут освобождаться любые дополнительные ресурсы.

Как выглядит код связывания представлений для фрагмента

Теперь вы готовы узнать, как выглядит код связывания представлений для фрагментов. Как и в случае с активностями, связывание представлений во фрагментах реализуется с использованием свойства связывания, но способ решения этой задачи немного другой. Код выглядит так:

```
package com.hfad.bitsandpizzas
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import com.hfad.bitsandpizzas.databinding.FragmentOrderBinding
class OrderFragment : Fragment() {
 private var _binding: FragmentOrderBinding? = null
 private val binding get() = _binding!!
 override fun onCreateView(
 inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
 ): View? {

 _binding = FragmentOrderBinding.inflate(inflater, container, false)
 val view = binding.root
 return view
 }
 override fun onDestroyView() {
```

```
 super.onDestroyView()
 _binding = null
 }
}
```

Как видите, в этом коде определяются два дополнительных свойства: `binding` и `_binding`. Давайте повнимательнее разберемся в том, что делают эти два свойства.

Как вы уже видели, код связывания представлений для фрагментов определяет свойство `_binding`:

```
private var _binding: FragmentOrderBinding? = null
```

Оно имеет тип `FragmentOrderBinding?` и инициализируется значением `null`. Свойству `_binding` присваивается экземпляр `FragmentOrderBinding` в методе `onCreateView()` фрагмента, для чего используется следующий код:

```
override fun onCreateView(...): View? {
 _binding = FragmentOrderBinding.inflate(inflater, container, false)
 ...
}
```

Оно задается в этом методе, потому что именно здесь фрагмент впервые получает доступ к своим представлениям. Свойству `_binding` снова присваивается значение null в методе `onDestroyView()` фрагмента:

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

Это происходит из-за того, что фрагмент не может обращаться к своим представлениям после вызова метода `onDestroyView()`.

Свойство `binding` использует метод чтения для получения версии `_binding`, отличной от null. Если `_binding` содержит null, выдается исключение null-указателя:

```
private val binding get() = _binding!!
```

Это означает, что свойство `binding` можно использовать для взаимодействия с представлениями фрагмента без множества хлопотных проверок null-безопасности. Например, чтобы кнопка FAB фрагмента реагировала на щелчки, можно использовать простую команду:

```
binding.fab.setOnClickListener {
 //Код, который делает что-то полезное
}
```

Активности и фрагменты получают доступ к своим представлениям в разных точках своих жизненных циклов.

Активность может взаимодействовать со своими представлениями, начиная с момента их создания и вызова их метода `onCreate()`. Представления остаются доступными до момента уничтожения активности, что происходит в конце жизненного цикла. Однако фрагмент может взаимодействовать со своими представлениями только с момента вызова его метода `onCreateView()`. Это означает, что присваивание объекта связывания свойству `_binding` может выполняться только в этом методе. Фрагмент сохраняет доступ к своим представлениям до того момента, когда будет вызван его метод `onDestroyView()`. В этот момент макет фрагмента теряется, поэтому `_binding` необходимо присвоить null. Тем самым предотвращается риск того, что фрагмент попытается использовать объект связывания представлений, когда взаимодействие с представлениями уже невозможно. Теперь вы знаете все, что необходимо знать для использования связывания представлений с фрагментами. Давайте обновим код `OrderFragment` в приложении `Bits and Pizzas`.

Полный код OrderFragment.kt

Ниже приведен код для использования связывания представлений в `OrderFragment`; обновите свой код `OrderFragment.kt`:

```
package com.hfad.bitsandpizzas
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.appcompat.app.AppCompatActivity

import android.widget.Toast
import com.google.android.material.snackbar.Snackbar
import com.hfad.bitsandpizzas.databinding.FragmentOrderBinding

class OrderFragment : Fragment() {
 private var _binding: FragmentOrderBinding? = null
 private val binding get() = _binding!!
 override fun onCreateView(
 inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
 ): View? {

 _binding = FragmentOrderBinding.inflate(inflater, container, false)
 val view = binding.root
 val toolbar = view.findViewById<MaterialToolbar>(R.id.toolbar)
 (activity as AppCompatActivity).setSupportActionBar(binding.toolbar)
```

```
binding.fab.setOnClickListener {

val pizzaType = binding.pizzaGroup.checkedRadioButtonId
if (pizzaType == -1) {
val text = "You need to choose a pizza type"
Toast.makeText(activity, text, Toast.LENGTH_LONG).show()
} else {
var text = (when (pizzaType) {
R.id.radio_chilee -> "Chilee pizza"
else -> "Funghi pizza"
})

text += if (binding.parmesan.isChecked) ", extra parmesan" else ""

text += if (binding.chiliOil.isChecked) ", extra chili oil" else ""
Snackbar.make(binding.fab, text, Snackbar.LENGTH_LONG).show()
}
}
return view
}
override fun onDestroyView() {
super.onDestroyView()
_binding = null
}
}
```

И это весь код, необходимый для использования связывания представлений во фрагменте `OrderFragment`. Проведем тест-драйв приложения и убедимся в том, что оно по-прежнему нормально работает.

Запущенное приложение работает точно так же, как прежде. Если щелкнуть на кнопке FAB без выбора вида пиццы, появляется сообщение с предложением выбрать пиццу. Если выбрать вид пиццы и снова щелкнуть на FAB, в нижней части экрана появляется сообщение `Snackbar` с подробной информацией о заказе.

Резюме

- Связывание представлений — механизм взаимодействия с представлениями, безопасный по отношению к типам. Он эффективнее `findViewById()` и с меньшей вероятностью приведет к ошибкам на стадии выполнения.
- Связывание представлений может использоваться в коде активностей и фрагментов.
- Связывание представлений включается в файле `build.gradle` приложения.
- Связывание представлений генерирует класс связывания для каждого файла макета приложения.
- Каждый класс связывания включает свойство для каждого представления в макете, обладающего идентификатором.
- Используя связывание представлений с активностью, вы задаете значение его свойства `binding` в методе `onCreate()` активности.

- Используя связывание представлений с фрагментом, вы задаете значение его свойства `binding` в методе `onCreateView()` фрагмента и возвращаете ему значение null в `onDestroyView()`.
- Фрагменты содержат методы жизненного цикла, связанные с состоянием активности.