Практическая работа. Compose и представления

Лучшие результаты достигаются при совместной работе. К настоящему моменту вы научились строить пользовательские интерфейсы из представлений и компонентов `Compose`. А если вы хотите использовать их одновременно? В этой главе вы узнаете, как пользоваться преимуществами обеих технологий, добавляя компоненты `Compose` в пользовательские интерфейсы на базе представлений. Вы освоите средства, позволяющие компонентам работать с моделями представлений. Вы даже узнаете, как заставить их реагировать на обновления `Live Data`. К концу главы у вас появится все необходимое для использования компонентов Compose с представлениями и даже для перехода на пользовательские интерфейсы, построенные исключительно на базе Compose.

Компоненты Compose могут включаться в интерфейсы на базе View

В предыдущей вы научились реализовывать пользовательские интерфейсы Compose на примере нового приложения `Temperature Converter`. Вместо того чтобы добавлять представления в файл макета, мы строили интерфейс вызовами компонентных функций из кода Kotlin активности. Однако в некоторых ситуациях требуется совместить представления с компонентами `Compose` в одном пользовательском интерфейсе, например, если вы хотите использовать элементы, доступные только в форме представлений или компонентов `Compose`, или хотите перевести части своего приложения на `Compose`. А теперь хорошая новость: компоненты `Compose` можно добавить в пользовательский интерфейс, определенный в файле макета. Чтобы показать, как это делается, мы вернемся к приложению `Guessing Game`, созданному ранее в книге, и переведем его на `Compose`.

Возвращаемся к приложению `Guessing Game` Наверняка вы помните, что в приложении `Guessing Game` пользователь вводит свои предположения о том, какие буквы входят в загаданное слово. Пользователь побеждает, если все буквы будут отгаданы успешно, и проигрывает, если у него заканчиваются жизни.

Структура приложения Guessing Game

Пользовательский интерфейс приложения `Guessing Game` состоит из двух фрагментов: `GameFragment` и `ResultFragment`. `GameFragment` — главный экран приложения, на котором, собственно, проходит игра. Здесь выводится информация (количество оставшихся жизней и ошибочные предположения пользователя), а пользователь вводит свои предположения. Также здесь находится кнопка, по щелчку на которой пользователь немедленно завершает игру. Фрагмент `ResultFragment` отображается при завершении игры. Он сообщает, выиграл ли пользователь, а также выводит загаданное слово. Приложение также включает две модели представлений (`GameViewModel` и `ResultViewModel`), которые содержат логику и данные игры и сохраняют состояние при повороте устройства. GameFragment использует `GameViewModel`, а `ResultFragment` использует `ResultViewModel`:

Что мы собираемся сделать

Замена представлений `Guessing Game` компонентами Compose будет выполнена в два этапа:

1. Замена представлений `ResultFragment` компонентами `Compose`. Мы добавим библиотеки `Compose` в файлы `build.gradle` приложения, а затем добавим компоненты в макет `ResultFragment` для замены текущих представлений. Когда вы будете уверены в том, что

компоненты делают то, что требуется, представления будут удалены из пользовательского интерфейса.

2. Замена представлений GameFragment компонентами `Compose`. Затем аналогичная процедура будет повторена для `GameFragment`. Сначала мы добавим компоненты `Compose` в макет, чтобы продублировать существующие представления. А когда вы будете уверены в том, что компоненты делают то, что требуется, представления будут удалены из пользовательского интерфейса.

Обновление файла build.gradle проекта и приложения

Начнем с добавления в файл `build.gradle` проекта новой переменной, определяющей используемую версию `Compose`. Откройте файл `GuessingGame/build.gradle` и добавьте следующую строку в раздел `buildscript`:

```
buildscript {
 ext.compose_version = "1.0.1"
}
```

В файл `build.gradle` приложения необходимо добавить некоторые настройки и библиотеки `Compose`, а также установить минимальный уровень SDK 29. Откройте файл `GuessingGame/app/build.gradle` и добавьте следующие строки в соответствующие разделы:

```
android {
 defaultConfig {
 minSdk 21
 ...
 }
 buildFeatures {
 ...
 compose true
 }
 composeOptions {
 kotlinCompilerExtensionVersion compose_version
 }
}
dependencies {
 ...
 implementation("androidx.compose.ui:ui:$compose_version")
 implementation("androidx.compose.ui:ui-tooling:$compose_version")
 implementation("androidx.compose.foundation:foundation:$compose_version")
 implementation("androidx.compose.material:material:$compose_version")
 implementation("androidx.compose.material:material-icons-core:$compose_version")
 implementation("androidx.compose.material:material-icons-
extended:$compose_version")
 implementation("androidx.compose.runtime:runtime-livedata:$compose_version")
 ...
}
```

После внесения всех изменений щелкните на ссылке Sync Now.

Замена представлений в ResultFragment компонентами Compose

Мы обновили файлы build.gradle и включили в них зависимости Compose, теперь можно заменить представления приложений компонентами Compose. Начнем с более простого компонента ResultFragment. Как вы помните, макет ResultFragment включает представление TextView для отображения результата игры и кнопку Button для запуска новой игры. Он выглядит так:

Эти представления можно заменить компонентами Compose, иначе говоря, вместо TextView будут использоваться компоненты Text, а вместо представлений Button — компоненты Button. Новый пользовательский интерфейс будет выглядеть так:

Новый пользовательский интерфейс будет строиться постепенно, так что на начальной стадии в ResultFragment будут использоваться как представления, так и компоненты Compose. Давайте разберемся, как добавить компоненты Compose в файл макета.

ComposeView позволяет включить компоненты Compose в макет

Чтобы включить компоненты Compose в пользовательский интерфейс на базе View, добавьте в файл макета элемент ComposeView — разновидность представлений, в которой могут отображаться представления Compose. Код выглядит так:

```
<androidx.compose.ui.platform.ComposeView
 android:id="@+id/compose_view"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"/>
```

ComposeView — своего рода представление-заместитель для любых компонентов Compose, которые вы хотите включить в пользовательский интерфейс из кода Kotlin. При запуске приложение отображает представления макета и заполняет ComposeView компонентами Compose.

Добавление ComposeView в fragment_result.xml

В пользовательском интерфейсе ResultFragment должны присутствовать компоненты Text и Button, поэтому в файл макета необходимо добавить компонент ComposeView. Ниже приведена новая версия fragment_result.xml; обновите файл:

```
<?xml version="1.0" encoding="utf-8"?>
<layout...>
 <data>
 <variable
 name="resultViewModel"
 type="com.hfad.guessinggame.ResultViewModel" />
 </data>
 <LinearLayout...>
 ...
 <androidx.compose.ui.platform.ComposeView
```

```
 android:id="@+id/compose_view"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"/>
 </LinearLayout>
</layout>
```

После того как представление `ComposeView` было добавлено в макет, включим в него компоненты `Compose`.

Добавление компонентов Compose в код Kotlin

После того как в макет будет включено представление `ComposeView`, к нему можно будет добавить компоненты `Compose` в методе `onCreateView()` фрагмента. Для этого используется следующий код:

```
override fun onCreateView(
 inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
): View? {
 _binding = FragmentResultBinding.inflate(inflater, container, false).apply {
 composeView.setContent {
 //Добавление компонентов Compose
 }
 }
 ...
}
```

В этом коде вызывается `setContent()` для представления `ComposeView` из макета; он сообщает, какие компоненты необходимо включить. Результат применяется к заполненному макету фрагмента. Например, если вы захотите добавить компонент `Text` к макету `ResultFragment`, для этого будет использоваться следующий код:

```
override fun onCreateView(
 inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
): View? {
 _binding = FragmentResultBinding.inflate(inflater, container, false).apply {
 composeView.setContent {
 Text("This is a composable")
 }
 }
 ...
}
```

При выполнении этого кода компонент `Text` будет отображаться в представлении `ComposeView`:

Добавление компонентной функции для содержимого фрагмента

Итак, вы научились добавлять компоненты `Compose` в `ComposeView`. Воспользуемся компонентом, добавленным в `fragment_result.xml`, для замены существующих представлений. Начнем с добавления в `ResultFragment.kt` новой компонентной функции с именем `ResultFragmentContent`, которая будет использоваться для определения пользовательского интерфейса фрагмента. Функция будет вызываться из `setContent()`, чтобы все добавленные в нее компоненты Compose выполнялись при отображении `ResultFragment`. Новый код приведен ниже; мы добавим его в `ResultFragment.kt` через несколько страниц:

```
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Surface
import androidx.compose.runtime.Composable
class ResultFragment : Fragment() {
 ...
 override fun onCreateView(
 inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
 ): View? {
 _binding = FragmentResultBinding.inflate(inflater, container, false).apply {
 composeView.setContent {
 MaterialTheme {
 Surface {
 ResultFragmentContent()
 }
 }
 }
 }
 ...
 }
 ...
}
@Composable
fun ResultFragmentContent() {
}
```

Замена кнопки Start New Game

После того как компонентная функция `ResultFragmentContent` будет добавлена в `ResultFragment.kt`, мы можем использовать ее для включения компонентов `Compose` в пользовательский интерфейс. Начнем с добавления кнопки. `ResultFragment` в настоящее время включает кнопку `View` с надписью «Start new game», которая использует следующего слушателя `OnClickListener`, чтобы по щелчку на кнопке происходил переход к `GameFragment`:

```
binding.newGameButton.setOnClickListener {
 view.findNavController()
 .navigate(R.id.action_resultFragment_to_gameFragment)
}
```

Чтобы смоделировать кнопку средствами Compose, можно создать новую компонентную функцию с именем NewGameButton, которая будет выполняться из ResultFragmentContent. К NewGameButton будет добавлен аргумент с лямбда-выражением, при помощи которого ResultFragmentContent сообщит, что должно происходить по щелчку. Новый код приведен ниже; мы добавим его в ResultFragment.kt через пару страниц:

```
@Composable
fun NewGameButton(clicked: () -> Unit) {
 Button(onClick = clicked) {
 Text("Start New Game")
 }
}
@Composable
fun ResultFragmentContent(view: View) {
 Column(modifier = Modifier.fillMaxWidth(),
 horizontalAlignment = Alignment.CenterHorizontally) {
 NewGameButton {
 view.findNavController()
 .navigate(R.id.action_resultFragment_to_gameFragment)
 }
 }
}
```

Замена TextView в ResultFragment

Напомним, что ResultFragment использует представление TextView в своем макете для отображения результата игры. Представление определяется следующим кодом:

```
<TextView
 android:id="@+id/won_lost"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:gravity="center"
 android:textSize="28sp"
 android:text="@{resultViewModel.result}" />
```

Вместо того чтобы использовать для вывода текста TextView, мы определим новую компонентную функцию ResultText, которая выводит текст в компоненте Compose Text. В функцию ResultText будет включен строковый аргумент, который будет использоваться ResultFragmentContent для передачи текста. Ниже приведен новый код; он будет добавлен в файл ResultFragment.kt на следующей странице:

```
@Composable
fun ResultText(result: String) {
 Text(text = result,
 fontSize = 28.sp,
```

```
 textAlign = TextAlign.Center)
}
@Composable
fun ResultFragmentContent(view: View, viewModel: ResultViewModel) {
 Column(modifier = Modifier.fillMaxWidth(),
 horizontalAlignment = Alignment.CenterHorizontally) {
 ResultText(viewModel.result)
 NewGameButton {
 view.findNavController()
 .navigate(R.id.action_resultFragment_to_gameFragment)
 }
 }
}
```

И это все, что необходимо для замены всех представлений `ResultFragment` компонентами. Посмотрим, как выглядит полный код.

Обновленный код ResultFragment.kt

Ниже приведена новая версия кода `ResultFragment.kt`; обновите файл:

```
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Surface
import androidx.compose.runtime.Composable
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.material.Button
import androidx.compose.material.Text
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.sp
class ResultFragment : Fragment() {
 private var _binding: FragmentResultBinding? = null
 private val binding get() = _binding!!
 lateinit var viewModel: ResultViewModel
 lateinit var viewModelFactory: ResultViewModelFactory
 override fun onCreateView(
 inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
 ): View? {
 _binding = FragmentResultBinding.inflate(inflater, container, false).apply {
 composeView.setContent {
 MaterialTheme {
 Surface {
 view?.let { ResultFragmentContent(it, viewModel) }
 }
 }
 }
 }
 val view = binding.root
 val result = ResultFragmentArgs.fromBundle(requireArguments()).result
```

```
 viewModelFactory = ResultViewModelFactory(result)
 viewModel = ViewModelProvider(this, viewModelFactory)
 .get(ResultViewModel::class.java)
 binding.resultViewModel = viewModel
 binding.newGameButton.setOnClickListener { ... }
 return view
 }
 override fun onDestroyView() { ... }
}
@Composable
fun ResultText(result: String) {
 Text(text = result,
 fontSize = 28.sp,
 textAlign = TextAlign.Center)
}
@Composable
fun NewGameButton(clicked: () -> Unit) {
 Button(onClick = clicked) {
 Text("Start New Game")
 }
}
@Composable
fun ResultFragmentContent(view: View, viewModel: ResultViewModel) {
 Column(modifier = Modifier.fillMaxWidth(),
 horizontalAlignment = Alignment.CenterHorizontally) {
 ResultText(viewModel.result)
 NewGameButton {
 view.findNavController()
 .navigate(R.id.action_resultFragment_to_gameFragment)
 }
 }
}
```

При запуске приложения отображается фрагмент `GameFragment`. Если щелкнуть на кнопке `Finish Game`, приложение переходит к `ResultFragment`. `ResultFragment` отображает исходные представления и компоненты, которые мы только что добавили. Компонент `Text` вводит правильный текст, а по щелчку на компоненте `Button` происходит переход к `GameFragment`.

Компоненты, добавленные в `ResultFragment`, работают так, как планировалось. А значит, на следующем этапе представления можно удалить.

Необходимо удалить представления из ResultFragment и обновить ResultFragment.kt

Представления можно удалить из фрагмента или активности двумя способами:

1. Удалить все лишние представления из файла макета. Такой подход может быть полезен, если пользовательский интерфейс содержит как представления, так и компоненты `Compose`.

2. Удалить весь файл макета. Если пользовательский интерфейс содержит только компоненты, файл макета можно удалить и убрать любые ссылки на него из кода Kotlin активности или фрагмента.

В приложении `Guessing Game` мы заменили все представления `ResultFragment` компонентами `Compose`, поэтому все исходные представления уже не нужны. Это означает, что файл макета `fragment_result.xml` можно удалить, чтобы пользовательский интерфейс содержал только компоненты `Compose`.

Прежде чем удалять файл макета, необходимо сначала удалить все ссылки на его представления из `ResultFragment.kt` и отказаться от использования связывания представлений. Также необходимо изменить метод `onCreateView()` фрагмента, чтобы вместо заполнения файла макета он добавлял компоненты `Compose` в пользовательский интерфейс фрагмента. Как вы узнали, с активностью это делается простым вызовом `setContent()` из ее метода `onCreate()`:

```
class MainActivity : ComponentActivity() {
 override fun onCreate(savedInstanceState: Bundle?) {
 super.onCreate(savedInstanceState)
 setContent {
 //Код, в котором выполняются компоненты Compose
 }
 }
}
```

Однако с фрагментами потребуется несколько иной подход. Чтобы понять, с чем это связано, вернемся к методу `onCreateView()` фрагмента.

onCreateView() возвращает корневое представление

Как вы уже знаете, метод `onCreateView()` фрагмента вызывается тогда, когда активности требуется отобразить пользовательский интерфейс фрагмента. Если пользовательский интерфейс определяется в файле макета, код в методе `onCreateView()` заполняет макет иерархией представлений и возвращает корневое представление. Корневое представление добавляется в макет активности, которая отображает пользовательский интерфейс фрагмента. Но что происходит, если файл макета не существует?

Получение ComposeView для интерфейсов Compose

Если пользовательский интерфейс фрагмента строится из компонентов `Compose` и не имеет файла макета, метод `onCreateView()` все равно должен вернуть `View?`, в противном случае код не будет компилироваться. Проблема решается возвращением представления `ComposeView`, которое включает все компоненты `Compose` пользовательского интерфейса. Код выглядит примерно так:

```
override fun onCreateView(
 inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
): View? {
 //Код, не использующий компоненты Compose
 return ComposeView(requireContext()).apply {
 setContent {
 //Код Compose, определяющий UI
 }
```

```
    }
}
```

Когда активности потребуется отобразить пользовательский интерфейс фрагмента, она вызывает метод `onCreateView()` фрагмента, как и прежде. Метод возвращает представление `ComposeView`, содержащее компоненты фрагмента, которое затем отображается активностью. Это все, что необходимо знать для завершения кода `ResultFragment.kt`. Полный код приводится на нескольких ближайших страницах.

Полный код ResultFragment.kt

Ниже приведен полный код `ResultFragment.kt`; обновите файл:

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

import androidx.navigation.findNavController
import androidx.lifecycle.ViewModelProvider
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Surface
import androidx.compose.runtime.Composable
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.material.Button
import androidx.compose.material.Text
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.sp
import androidx.compose.ui.platform.ComposeView

class ResultFragment : Fragment() {

 lateinit var viewModel: ResultViewModel
 lateinit var viewModelFactory: ResultViewModelFactory
 override fun onCreateView(
 inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
 ): View? {
 val result = ResultFragmentArgs.fromBundle(requireArguments()).result
 viewModelFactory = ResultViewModelFactory(result)
 viewModel = ViewModelProvider(this@ResultFragment, viewModelFactory)
 .get(ResultViewModel::class.java)

 return ComposeView(requireContext()).apply {
 .setContent {
 MaterialTheme {
 Surface {
 view?.let { ResultFragmentContent(it, viewModel) }
```

```
 }
 }
 }
 }
 }
@Composable
fun ResultText(result: String) {
 Text(text = result,
 fontSize = 28.sp,
 textAlign = TextAlign.Center)
}
@Composable
fun NewGameButton(clicked: () -> Unit) {
 Button(onClick = clicked) {
 Text("Start New Game")
 }
}
@Composable
fun ResultFragmentContent(view: View, viewModel: ResultViewModel) {
 Column(modifier = Modifier.fillMaxWidth(),
 horizontalAlignment = Alignment.CenterHorizontally) {
 ResultText(viewModel.result)
 NewGameButton {
 view.findNavController()
 .navigate(R.id.action_resultFragment_to_gameFragment)
 }
 }
}
```

Удаление fragment_result.xml

Теперь `ResultFragment` не использует файл макета, поэтому мы можем удалить любые ссылки на `fragment_result.xml` из графа навигации и удалить файл макета. Откройте файл `nav_graph.xml` из папки `app/src/main/res/navigation` и удалите ссылку "@layout/fragment_result" из раздела `ResultFragment`:

```
<fragment
 android:id="@+id/resultFragment"
 ...
 tools:layout="@layout/fragment_result"  удалить  >
```

Щелкните правой кнопкой мыши на файле `fragment_result.xml` на панели проекта, выберите команду `Refactor`, а затем вариант `Safe Delete`. Если щелкнуть на кнопке `OK` и выбрать вариант с проведением рефакторинга, файл будет удален. Посмотрим, что произойдет при запуске приложения.

Что происходит при выполнении приложения

При выполнении приложения происходят следующие события:

1. `MainActivity` запускается и отображает `GameFragment` в своем макете.
2. Когда пользователь щелкает на кнопке `Finish Game`, а также в случае выигрыша или проигрыша приложение переходит к фрагменту `ResultFragment` и вызывает его метод `onCreateView()`. `
3. Метод `onCreateView()` создает `ComposeView`.
4. Метод `onCreateView()` присваивает `ResultFragmentContent` содержимое `ComposeView`.
5. При выполнении `ResultFragmentContent` выполняются компоненты `ResultText` и `NewGameButton`. Компоненты добавляются в `ComposeView`.
6. Метод `onCreateView()` возвращает `ComposeView` активности `MainActivity`.
7. `MainActivity` отображает `ComposeView` в своем макете. Компоненты `ComposeView` отображаются на устройстве.

Если запустить приложение и щелкнуть на кнопке `Finish Game` фрагмента `GameFragment`, происходит переход к `ResultFragment`, как и прежде. Однако на этот раз пользовательский интерфейс строится из компонентов `Compose`.

Если запустить приложение и щелкнуть на кнопке `Finish Game` фрагмента `GameFragment`, происходит переход к `ResultFragment`, как и прежде. Однако на этот раз пользовательский интерфейс строится из компонентов `Compose`.

GameFragment тоже переводится на использование компонентов Compose

Итак, мы успешно перевели `ResultFragment` на использование компонентов вместо представлений; теперь можно сделать то же самое с `GameFragment`. Как вы помните, макет `GameFragment` включает представления `TextView`, `Button` и `EditText`, при помощи которых пользователь вводит свои предположения в игре. Напомним, как выглядит интерфейс игры:

На нескольких ближайших страницах мы заменим эти представления компонентами `Compose`. После завершения работы пользовательский интерфейс будет выглядеть примерно так:

Как и на предыдущем этапе, мы начнем с добавления новых компонентов `Compose` в пользовательский интерфейс `GameFragment`; это означает, что в его макет необходимо добавить представление `ComposeView`. Код включения `ComposeView` приведен на следующей странице.

Добавление ComposeView в fragment_game.xml

Ниже приведен код, добавляющий представление `ComposeView` в макет `GameFragment`; обновите файл `fragment_game.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<layout
 xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.com/tools"
 tools:context=".GameFragment">
 <data>
```

```
 <variable
 name="gameViewModel"
 type="com.hfad.guessinggame.GameViewModel" />
 </data>
 <LinearLayout
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:orientation="vertical"
 android:padding="16dp">
 ...
 <androidx.compose.ui.platform.ComposeView
 android:id="@+id/compose_view"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"/>
 </LinearLayout>
</layout>
```

Файл макета успешно обновлен, можно переходить к добавлению в него компонентов `Compose`.

Добавление компонентной функции для содержимого GameFragment

По аналогии с `ResultFragment.kt` мы добавим в `GameFragment.kt` новую компонентную функцию, которая будет использоваться для пользовательского интерфейса фрагмента. Функция с именем `GameFragmentContent` будет вызываться из `setContent()`, чтобы все добавленные в нее компоненты Compose выполнялись при отображении `GameFragment`. Ниже показано, как выглядит новый код; мы добавим его в `GameFragment.kt` через несколько страниц:

```
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Surface
import androidx.compose.runtime.Composable
class GameFragment : Fragment() {
 ...
 override fun onCreateView(
 inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
 ): View? {
 _binding = FragmentGameBinding.inflate(inflater, container, false).apply {
 composeView.setContent {
 MaterialTheme {
 Surface {
 GameFragmentContent()
 }
 }
 }
 }
 ...
 }
 ...
}
@Composable
```

```
fun GameFragmentContent() {
}
```

Замена кнопки Finish Game

Как и в случае с `ResultFragment`, мы добавим компоненты `Compose` в компонентную функцию `GameFragmentContent`, чтобы они отображались в пользовательском интерфейсе `GameFragment`. Начнем с замены кнопки `Finish Game`. Кнопка `Finish Game` определяется следующим кодом в макете фрагмента:

```
<Button
 android:id="@+id/finish_game_button"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_gravity="center"
 android:text="Finish Game"
 android:onClick="@{() -> gameViewModel.finishGame()}" />
```

Как видите, по щелчку на кнопке вызывается метод `finishGame()` объекта `GameViewModel`. Чтобы воспроизвести эту функциональность в `Compose`, создайте новую функцию с именем `FinishGameButton`, которая будет выполняться из `GameFragmentContent`. Новый код приведен ниже; мы добавим его в `GameFragment.kt` через несколько страниц:

```
@Composable
fun FinishGameButton(clicked: () -> Unit) {
 Button(onClick = clicked) {
 Text("Finish Game")
 }
}
@Composable
fun GameFragmentContent(viewModel: GameViewModel) {
 Column(modifier = Modifier.fillMaxWidth(),
 horizontalAlignment = Alignment.CenterHorizontally) {
 FinishGameButton {
 viewModel.finishGame()
 }
 }
}
```

Замена EditText на TextField

Следующим шагом станет замена представления `EditText`, в котором пользователь вводит буквы. Мы создадим новую компонентную функцию с именем `EnterGuess`, которая использует `TextField` и получает два аргумента: строку с введенной буквой и лямбда-выражение, которое указывает, что должно происходить при изменении значения. Функция `EnterGuess` будет выполняться из

`GameFragmentContent`, чтобы компонент был добавлен в пользовательский интерфейс фрагмента. Также мы добавим в `GameFragmentContent` объект `MutableState` с именем `guess`, который будет использоваться для управления состоянием `TextField`. Новый код приводится ниже; мы добавим его в файл `GameFragment.kt` через пару страниц:

```
@Composable
fun EnterGuess(guess: String, changed: (String) -> Unit) {
 TextField(
 value = guess,
 label = { Text("Guess a letter") },
 onValueChange = changed
 )
}
@Composable
fun GameFragmentContent(viewModel: GameViewModel) {
 val guess = remember { mutableStateOf("") }
 Column(modifier = Modifier.fillMaxWidth()) {
 EnterGuess(guess.value) { guess.value = it }
 Column(modifier = Modifier.fillMaxWidth(),
 horizontalAlignment = Alignment.CenterHorizontally) {
 FinishGameButton {
 viewModel.finishGame()
 }
 }
 }
}
```

Замена кнопки Guess

После добавления в интерфейс поля для ввода предположений будет добавлен компонент `Button`, который передает букву методу `makeGuess()` модели представления. Для добавления кнопки мы воспользуемся новой компонентной функцией с именем `GuessButton`, которая выполняется из `GameFragmentContent`. Новый код приводится ниже; мы добавим его в файл `GameFragment.kt` на следующей странице:

```
@Composable
fun GuessButton(clicked: () -> Unit) {
 Button(onClick = clicked) {
 Text("Guess!")
 }
}
@Composable
fun GameFragmentContent(viewModel: GameViewModel) {
 val guess = remember { mutableStateOf("") }
 Column(modifier = Modifier.fillMaxWidth()) {
 EnterGuess(guess.value) { guess.value = it }
 Column(modifier = Modifier.fillMaxWidth(),
 horizontalAlignment = Alignment.CenterHorizontally) {
 GuessButton {
```

```
 viewModel.makeGuess(guess.value.uppercase())
 guess.value = ""
 }
 FinishGameButton {
 viewModel.finishGame()
 }
 }
 }
}
```

Мы заменили три представления `GameFragment` компонентами `Compose`. Прежде чем заниматься остальными компонентами, обновим `GameFragment.kt` и проведем тест-драйв приложения.

Обновленный код GameFragment.kt

Ниже приведена новая версия `GameFragment.kt`; обновите файл:

```
import androidx.compose.runtime.Composable
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.material.*
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
class GameFragment : Fragment() {
 ...
 override fun onCreateView(
 inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
 ): View? {
 _binding = FragmentGameBinding.inflate(inflater, container, false).apply {
 composeView.setContent {
 MaterialTheme {
 Surface {
 GameFragmentContent(viewModel)
 }
 }
 }
 }
 ...
 return view
 }
 ...
}
@Composable
fun FinishGameButton(clicked: () -> Unit) {
 Button(onClick = clicked) {
 Text("Finish Game")
 }
}
@Composable
```

```
fun EnterGuess(guess: String, changed: (String) -> Unit) {
 TextField(
 value = guess,
 label = { Text("Guess a letter") },
 onValueChange = changed
 )
}
@Composable
fun GuessButton(clicked: () -> Unit) {
 Button(onClick = clicked) {
 Text("Guess!")
 }
}
@Composable
fun GameFragmentContent(viewModel: GameViewModel) {
 val guess = remember { mutableStateOf("") }
 Column(modifier = Modifier.fillMaxWidth()) {
 EnterGuess(guess.value) { guess.value = it }
 Column(modifier = Modifier.fillMaxWidth(),
 horizontalAlignment = Alignment.CenterHorizontally) {
 GuessButton {
 viewModel.makeGuess(guess.value.uppercase())
 guess.value = ""
 }
 FinishGameButton {
 viewModel.finishGame()
 }
 }
 }
}
```

При запуске приложения отображается фрагмент `GameFragment`. Он включает все исходные представления, а также три дополнительных компонента, при помощи которых пользователь может вводить предположения и завершать игру. Когда вы используете компоненты `EnterGuess` и `GuessButton` для ввода предположений о том, какие буквы входят в загаданное слово, приложение регистрирует каждую догадку. Если предположение было правильным, буква добавляется в шаблон загаданного слова на экране, а если неправильным — количество оставшихся жизней обновляется и добавляется в список неправильных предположений.

Вывод ошибочных предположений в компоненте Text

На следующем шаге мы заменим представление `TextView`, которое использует механизм `Live Data` для отображения ошибочных предположений пользователя. Каждый раз, когда пользователь вводит отсутствующую букву, она добавляется в свойство `incorrectGuesses` модели представления, а `TextView` реагирует обновлением выводимого текста. Напомним, как выглядит код `TextView`:

```
<TextView
 android:id="@+id/incorrect_guesses"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
```

```
    android:textSize="16sp"
    android:text="@{@string/incorrect_guesses(gameViewModel.incorrectGuesses)}" />
```

`TextView` можно заменить компонентом `Compose Text`, в котором выводится тот же текст. Но что нужно сделать, чтобы текст обновлялся при изменении значения свойства `incorrectGuesses`?

Использование observeAsState() для реагирования на Live Data

Как вы узнали, компоненты перестраиваются при появлении новых значений в объектах `State` или `MutableState`, от которых они зависят. Однако с объектами `Live Data`, такими как свойство `incorrectGuesses` модели представления, этого не происходит. Если вы попытаетесь использовать значение объекта `Live Data` с компонентом `Compose`, он не будет перерисовываться при обновлении значения; он просто продолжит использовать исходное значение объекта, и данные перестанут быть актуальными. Если вы хотите, чтобы компонент Compose реагировал на обновления `Live Data`, воспользуйтесь функцией `observeAsState()`. Функция возвращает State-версию объекта `Live Data`, чтобы любые зависящие от него компоненты `Compose` перестраивались при изменении его значения. Пример использования функции `observeAsState()`:

```
val incorrectGuesses = viewModel.incorrectGuesses.observeAsState()
```

Эта команда определяет переменную (с типом `State`), которая наблюдает за `Live Data`- свойством `incorrectGuesses` модели представления. Это означает, что при изменении его значения компоненты смогут отреагировать на него. Применим этот способ на практике, заменив представление `TextView` в `GameFragment` компонентом `Compose`.

Создание компонентной функции IncorrectGuessesText

Чтобы заменить представление `TextView` для ошибочных предположений, мы определим новую компонентную функцию с именем `IncorrectGuessesText`. Эта функция получает аргумент `GameViewModel`, наблюдает за свойством `incorrectGuesses` и использует компонент Text для вывода своего значения в пользовательском интерфейсе. При каждом обновлении свойства компонент Text перестраивается и выводит обновленный текст. Функция `IncorrectGuessesText` выглядит так:

```
@Composable
fun IncorrectGuessesText(viewModel: GameViewModel) {
 val incorrectGuesses = viewModel.incorrectGuesses.observeAsState()
 incorrectGuesses.value?.let {
 Text(stringResource(R.string.incorrect_guesses, it))
 }
}
```

Как видите, компонент `Text` в этом коде использует функцию `stringResource()` для назначения своего текста. Эта функция позволяет использовать строковые ресурсы с компонентами и передавать им аргументы.

Выполнение IncorrectGuessesText из GameFragmentContent

Как и в случае с остальными компонентными функциями, созданными нами ранее, для включения компонента `IncorrectGuessesText` в пользовательский интерфейс `GameFragment` мы выполним его из компонентной функции `GameFragmentContent`. Новая версия кода выглядит так:

```
@Composable
fun GameFragmentContent(viewModel: GameViewModel) {
 ...
 Column(modifier = Modifier.fillMaxWidth()) {
 IncorrectGuessesText(viewModel)
 ...
 }
}
```

Обновленный код GameFragment.kt

Теперь вы знаете, как заменить все представления `GameFragment` компонентами `Compose`, и мы можем добавить компоненты в `GameFragment.kt`. Обновите файл:

```
import androidx.compose.runtime.livedata.observeAsState
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.unit.em
import androidx.compose.ui.unit.sp
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Arrangement
class GameFragment : Fragment() {
 ...
}
@Composable
fun FinishGameButton(clicked: () -> Unit) { ... }
@Composable
fun EnterGuess(guess: String, changed: (String) -> Unit) { ... }
@Composable
fun GuessButton(clicked: () -> Unit) { ... }
@Composable
fun IncorrectGuessesText(viewModel: GameViewModel) {
 val incorrectGuesses = viewModel.incorrectGuesses.observeAsState()
 incorrectGuesses.value?.let {
 Text(stringResource(R.string.incorrect_guesses, it))
 }
}
@Composable
fun LivesLeftText(viewModel: GameViewModel) {
 val livesLeft = viewModel.livesLeft.observeAsState()
 livesLeft.value?.let {
 Text(stringResource(R.string.lives_left, it))
 }
}
@Composable
```

```
fun SecretWordDisplay(viewModel: GameViewModel) {
 val display = viewModel.secretWordDisplay.observeAsState()
 display.value?.let {
 Text(text = it,
 letterSpacing = 0.1.em,
 fontSize = 36.sp)
 }
}
@Composable
fun GameFragmentContent(viewModel: GameViewModel) {
 val guess = remember { mutableStateOf("") }
 Column(modifier = Modifier.fillMaxWidth()) {
 Row(modifier = Modifier.fillMaxWidth(),
 horizontalArrangement = Arrangement.Center) {
 SecretWordDisplay(viewModel)
 }
 LivesLeftText(viewModel)
 IncorrectGuessesText(viewModel)
 EnterGuess(guess.value) { guess.value = it }
 Column(modifier = Modifier.fillMaxWidth(),
 horizontalAlignment = Alignment.CenterHorizontally) {
 GuessButton {
 viewModel.makeGuess(guess.value.uppercase())
 guess.value = ""
 }
 FinishGameButton {
 viewModel.finishGame()
 }
 }
 }
}
```

При запуске приложения фрагмент `GameFragment` включает как все исходные представления, так и их аналоги `Compose`. Когда пользователь вводит буквы, которые, по его мнению, могут входить в загаданное слово, текст в компонентах `SecretWordDisplay`, `LivesLeftText` и `IncorrectGuessesText` автоматически обновляется.

Теперь все компоненты `Compose` в `GameFragment` работают именно так, как требовалось. Остается лишь удалить представления из фрагмента.

Удаление представлений из GameFragment.kt

Как и в случае с `ResultFragment`, для исключения представлений из `GameFragment` мы удалим файл макета. Но сначала необходимо удалить все ссылки на представления из `GameFragment.kt`. Также будет отключено связывание данных, так как в новой версии оно не используется Ниже приведен полный код новой версии; обновите файл `GameFragment.kt`:

```
package com.hfad.guessinggame
import android.os.Bundle
import androidx.fragment.app.Fragment
```

```
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup

import androidx.navigation.findNavController
import androidx.lifecycle.ViewModelProvider
import androidx.lifecycle.Observer
import androidx.compose.runtime.Composable
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.material.*
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.runtime.livedata.observeAsState
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.unit.em
import androidx.compose.ui.unit.sp
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.ui.platform.ComposeView

class GameFragment : Fragment() {

 lateinit var viewModel: GameViewModel
 override fun onCreateView(
 inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
 ): View? {
 viewModel = ViewModelProvider(this).get(GameViewModel::class.java)
 viewModel.gameOver.observe(viewLifecycleOwner, Observer { newValue ->
 if (newValue) {
 val action = GameFragmentDirections
 .actionGameFragmentToResultFragment(viewModel.wonLostMessage())
 view?.findNavController()?.navigate(action)
 }
 })

 return ComposeView(requireContext()).apply {
 .setContent {
 MaterialTheme {
 Surface {
 GameFragmentContent(viewModel)
 }
 }
 }
 }
@Composable
fun FinishGameButton(clicked: () -> Unit) {
 Button(onClick = clicked) {
 Text("Finish Game")
 }
}
@Composable
```

```
fun EnterGuess(guess: String, changed: (String) -> Unit) {
 TextField(
 value = guess,
 label = { Text("Guess a letter") },
 onValueChange = changed
 )
}
@Composable
fun GuessButton(clicked: () -> Unit) {
 Button(onClick = clicked) {
 Text("Guess!")
 }
}
@Composable
fun IncorrectGuessesText(viewModel: GameViewModel) {
 val incorrectGuesses = viewModel.incorrectGuesses.observeAsState()
 incorrectGuesses.value?.let {
 Text(stringResource(R.string.incorrect_guesses, it))
 }
}
@Composable
fun LivesLeftText(viewModel: GameViewModel) {
 val livesLeft = viewModel.livesLeft.observeAsState()
 livesLeft.value?.let {
 Text(stringResource(R.string.lives_left, it))
 }
}
@Composable
fun SecretWordDisplay(viewModel: GameViewModel) {
 val display = viewModel.secretWordDisplay.observeAsState()
 display.value?.let {
 Text(text = it,
 letterSpacing = 0.1.em,
 fontSize = 36.sp)
 }
}
@Composable
fun GameFragmentContent(viewModel: GameViewModel) {
 val guess = remember { mutableStateOf("") }
 Column(modifier = Modifier.fillMaxWidth()) {
 Row(modifier = Modifier.fillMaxWidth(),
 horizontalArrangement = Arrangement.Center) {
 SecretWordDisplay(viewModel)
 }
 LivesLeftText(viewModel)
 IncorrectGuessesText(viewModel)
 EnterGuess(guess.value) { guess.value = it }
 Column(modifier = Modifier.fillMaxWidth(),
 horizontalAlignment = Alignment.CenterHorizontally) {
 GuessButton {
 viewModel.makeGuess(guess.value.uppercase())
 guess.value = ""
 }
 FinishGameButton {
```

```
 viewModel.finishGame()
 }
 }
 }
}
```

И это все изменения, которые необходимо внести в код `GameFragment.kt`, чтобы он не заполнял макет и не содержал ссылки на представления. Так как фрагменту уже не нужен файл макета, после удаления всех ссылок на него из графа навигации этот файл можно удалить. Это будет сделано на следующей странице.

Удаление fragment_game.xml

Граф навигации включает ссылку на файл макета `fragment_game.xml`. Перед удалением файла макета ее необходимо удалить. Откройте `nav_graph.xml` из папки `app/src/main/res/navigation` и удалите строку с упоминанием "@layout/fragment_game" из раздела `GameFragment`:

```
<fragment
 android:id="@+id/gameFragment"
 android:name="com.hfad.guessinggame.GameFragment"
 android:label="fragment_game"

 <action ... />
</fragment>
```

Затем щелкните правой кнопкой мыши на файле `fragment_game.xml` на панели проекта, выберите команду `Refactor` и вариант `Safe Delete`. Если щелкнуть на кнопке `OK` и выбрать вариант проведения рефакторинга, файл будет удален.

Отключение связывания данных

Остается внести последнее изменение в приложение `Guessing Game`: отключить связывание данных. Как вы помните, ранее мы включили связывание данных, чтобы представления из `fragment_game.xml` и `fragment_result.xml` могли взаимодействовать с моделью представления фрагментов. После того как файлы макетов будут удалены, связывание данных становится ненужным. Чтобы отключить связывание данных, откройте файл `GuessingGame/app/build.gradle` и удалите строку связывания данных из раздела `buildFeatures`:

```
buildFeatures {
 dataBinding true
 ...
 }
```

Когда это будет сделано, щелкните на ссылке `Sync Now`, чтобы синхронизировать изменения с остальными частями проекта.

При выполнении приложения отображается фрагмент `GameFragment`. На этот раз пользовательский интерфейс строится исключительно из компонентов `Compose`. Они работают именно так, как нам нужно.

Поздравляем! Теперь вы знаете, как добавить компоненты в существующий пользовательский интерфейс на базе `View` и даже заменить пользовательский интерфейс другим, в котором используются только компоненты `Compose`. Мы полагаем, что `Compose` ждет большое будущее; вы можете получить дополнительную информацию об этой технологии в приложении.

Резюме

- Приложения могут включать комбинации представлений `View` и компонентов `Compose`.
- `Compose` может использоваться в существующих приложениях при условии, что в них используется версия SDK 29 и выше и выбран язык Kotlin.
- Прежде чем использовать `Compose` в существующих пользовательских интерфейсах на базе `View`, необходимо добавить настройки и библиотеки `Compose` в файл `build.gradle`.
- `ComposeView` — представление, в которое можно добавлять компоненты `Compose`. Его можно рассматривать как своего рода заместитель для компонентов `Compose`, которые добавляются в файл макета.
- Компоненты `Compose` могут взаимодействовать со свойствами и методами модели представления.
- Используйте метод `observeAsState()` для того, чтобы компоненты `Compose` могли реагировать на обновления `Live Data`.
- После того как представления активности или фрагмента будут заменены компонентами `Compose`, файл макета можно будет удалить.