Практическая работа 11.1. Сохранение состояния с помощью onSaveInstanceState

С ростом сложности приложений фрагментам приходится жонглировать все большим количеством объектов. И если не проявить осторожность, это может привести к разбуханию кода, который пытается делать все сразу. Бизнес-логика, навигация, управление пользовательским интерфейсом, обработка изменений в конфигурации... что ни возьми, оно здесь. В этой главе вы узнаете, как действовать в подобных ситуациях с использованием моделей представлений. Вы узнаете, как они упрощают код активности и фрагментов и как они справляются с изменениями конфигурации, поддерживая целостное состояние приложения. Наконец, мы покажем, как построить фабрику моделей представлений и когда может возникнуть такая необходимость.

Снова об изменениях конфигурации

Изменение ориентации экрана является изменением конфигурации, которое заставляет Android уничтожить и заново создать текущую активность. В результате представления и свойства могут потерять свое состояние и вернуться к исходному значению:

Вы научились решать эту проблему с использованием метода `onSaveInstanceState` активности. Этот метод вызывается перед уничтожением активности, а для сохранения всех значений, которые могут быть потеряны, в нем обычно используется объект `Bundle`. При воссоздании активности сохраненные значения могут использоваться для восстановления состояния представлений и свойств активности

Сохранение состояния в объектах `Bundle` неплохо работает в относительно простых приложениях, но в более сложных ситуациях такое решение не идеально. Дело в том, что объекты `Bundles` предназначены для небольших объемов данных и работают с ограниченным набором типов.

Существуют и другие проблемы

Существует и другая проблема: код активностей и фрагментов быстро разрастается. Коду приходится управлять навигацией, обновлять интерфейс, сохранять состояние, а иногда он также включает более общую бизнес-логику для управления поведением приложения. Решение всех этих задач в одном месте удлиняет код, а также усложняет его чтение и сопровождение. В этой главе вы научитесь решать все перечисленные проблемы с использованием **модели представления.**

Знакомство с моделями представлений

Модель представления — отдельный класс, который существует параллельно с кодом активности или фрагмента. Он отвечает за все данные, которые должны отображаться на экране, а также может содержать любую бизнес-логику. Каждый раз, когда фрагменту требуется обновить свой макет, он запрашивает у модели представления новейшие значения, которые ему нужно отобразить, а если ему потребуется доступ к бизнес-логике, он вызывает методы, содержащиеся в модели представления.

Для чего нужны модели представлений?

Существует пара причин для использования модели представления.

Использование модели представления упрощает код активности или фрагмента. Во фрагмент уже не нужно включать код, относящийся к бизнес-логике приложения, так как все хранится в отдельном классе. Вместо этого можно сосредоточиться на таких аспектах, как обновление экрана или навигация.

Другая причина заключается в том, что модель представления может пережить изменения конфигурации. Она не уничтожается, когда пользователь поворачивает экран устройства, так что состояние отдельных переменных не будет потеряно. Это позволит вам восстановить состояние приложения без сохранения значений в `Bundle`.

Чтобы вы лучше поняли, как пользоваться моделями представлений, мы построим игру с угадыванием слов. Но прежде чем переходить к написанию кода, разберемся, как будет работать эта игра.

Как будет работать игра

В игре `Guessing Game` пользователь пытается угадать загаданное слово. При запуске приложение выбирает случайное слово из массива и выводит пробел для каждой буквы слова:

Пользователь вводит букву, которая, по его предположению, может входить в загаданное слово. Если предположение было правильным, игра выводит букву в той позиции, в какой она встречается в загаданном слове. Если предположение было ошибочным, игра выводит сообщение, а игрок теряет одну жизнь.

Пользователь продолжает делать предположения, пока не угадает все буквы или не потеряет все жизни. Когда это произойдет, появляется новый экран, на котором выводится загаданное слово, а также сообщается результат игры. На этом экране пользователю также предлагается начать новую игру.

Структура приложения

Приложение будет содержать одну активность (с именем `MainActivity`), которая используется для отображения двух фрагментов игры: `GameFragment` и `ResultFragment`.

`GameFragment` представляет главный экран приложения. На нем выводятся пустые ячейки, обозначающие буквы загаданного слова, пользователю предоставляется возможность ввести букву, а также отображается количество ошибочных предположений и оставшихся жизней.

При завершении игры GameFragment использует компонент `Navigation` для перехода к `ResultFragment`, при этом передается строка с результатом. `ResultFragment` отображает строку и кнопку, при помощи которой пользователь может начать новую игру.

Структура приложения будет более подробно описана в процессе построения. Начнем с перечисления тех шагов, которые нам предстоит сделать.

Что мы собираемся сделать

Ниже перечислены основные этапы построения приложения.

1. Написание базовой игры. Мы создадим фрагменты `GameFragment` и `ResultFragment`, напишем игровую логику и воспользуемся компонентом `Navigation` для переходов между двумя фрагментами.

2. Добавление модели представления для `GameFragment`. Мы создадим модель представления `GameViewModel` для хранения игровой логики и данных `GameFragment`. Такой подход упрощает код `GameFragment` и гарантирует, что игра переживет изменения конфигурации.

3. Добавление модели представления для `ResultFragment`. Мы добавим вторую модель представления `ResultViewModel`, которая должна использоваться `ResultFragment`. В этой модели представления будет храниться результат игры, только что завершенной пользователем.

Создание проекта Guessing Game

Для приложения `Guessing Game` нужно создать новый проект. Выберите вариант Empty Activity, введите имя приложения «Guessing Game» и имя пакета «com.hfad.guessinggame» и подтвердите каталог для сохранения по умолчанию. Убедитесь в том, что выбран язык Kotlin с минимальным уровнем SDK API 29, чтобы приложение работало на большинстве устройств Android. Приложение `Guessing Game` будет использовать механизм связывания представлений для обращения к своим представлениям и компонент `Navigation` для перехода между своими фрагментами. Обновим файлы `build.gradle` проекта и приложения и включим в них эти библиотеки.

Обновление файла build.gradle проекта...

В файл `build.gradle` проекта будет добавлена новая переменная, определяющая используемую версию компонента `Navigation`, а также путь к классам `Safe Args`. Откройте файл *GuessingGame/build.gradle* и добавьте следующие строки в указанные разделы:

```
buildscript {
 ext.nav_version = "2.3.5"
 ...
 dependencies {
 classpath "androidx.navigation:navigation-safe-args-gradle-plug-in:$nav_version"
 ...
 }
}
```

В файле `build.gradle` приложения необходимо включить связывание представлений, зависимость для библиотеки компонента `Navigation`, применить плагин `Safe Args` для передачи аргументов фрагментам. Откройте файл *GuessingGame/app/build.gradle* и добавьте следующие строки (выделенные жирным шрифтом) в соответствующие разделы:

```
plug-ins {
 ...
 id 'androidx.navigation.safeargs.kotlin'
}
android {
 ...
 buildFeatures {
```

```
 viewBinding true
 }
}
dependencies {
 ...
 implementation "androidx.navigation:navigation-fragment-ktx:$nav_version"
 ...
}
```

После внесения изменений щелкните на ссылке *Sync Now*, чтобы синхронизировать изменения с остальными частями проекта. Следующим шагом станет создание фрагментов приложения.

Приложению Guessing Game необходимы два фрагмента: `GameFragment` и `ResultFragment`. `GameFragment` представляет главный экран приложения, а `ResultFragment` будет использоваться для отображения результата:

Сделаем следующий шаг и добавим два фрагмента в проект.

Создание GameFragment...

Сначала добавим `GameFragment` в проект. Выделите пакет `com.hfad.guessinggame` в папке *app/src/main/java*, после чего выберите команду `File→New→Fragment→Fragment (Blank)`. Введите имя фрагмента «GameFragment» и имя макета «fragment_ game» и убедитесь в том, что выбран язык Kotlin.

Затем добавьте фрагмент `ResultFragment` — выделите пакет `com.hfad.guessinggame` в папке *app/src/main/java* и выберите команду `File→New→Fragment→Fragment(Blank)`. Введите имя фрагмента «ResultFragment» и имя макета «fragment_result» и убедитесь в том, что выбран язык Kotlin.

Код этих двух фрагментов будет обновлен через несколько страниц. Но сначала мы добавим в проект граф навигации, который сообщит приложению, как должны происходить переходы между двумя фрагментами.

Как должна работать навигация

Как вы уже знаете, граф навигации сообщает Android информацию о возможных целях проекта и о переходах между ними. В приложении `Guessing Game` навигация должна работать так:

1. При запуске приложения отображается `GameFragment`.
2. При выигрыше или проигрыше `GameFragment` переходит к фрагменту `ResultFragment`, передавая ему результат.
3. Когда пользователь щелкает на кнопке `New Game`, приложение переходит к `GameFragment`.

Чтобы реализовать эту схему, мы добавим `GameFragment` и `ResultFragment` в новый граф навигации (который вскоре будет создан) и укажем, что каждый фрагмент может переходить к другому. Также следует указать, что `GameFragment` будет передавать строковый аргумент фрагменту `ResultFragment`. Аргумент должен содержать сообщение, указывающее, выиграл или проиграл пользователь только что сыгранную игру. Фрагмент `ResultFragment` будет выводить сообщение при переходе к нему.

Создание графа навигации

Чтобы создать граф навигации, выделите папку *GuessingGame/app/src/ main/res* на панели проекта, а затем выберите команду `File→New→AndroidResource File`. Введите имя файла «nav_graph», выберите тип ресурса «Navigation» и щелкните на кнопке OK. В результате будет создан граф навигации с именем `nav_graph.xml`. Граф навигации необходимо обновить и включить в него фрагменты `GameFragment` и `ResultFragment` вместе с действиями навигации, приводящими к ним. Мы сделаем это на следующем шаге.

Обновление графа навигации

Чтобы обновить граф навигации, откройте файл `nav_graph.xml` (если он не был открыт ранее), переключитесь в режим Code и обновите код (изменения выделены жирным шрифтом):

```
<?xml version="1.0" encoding="utf-8"?>
<navigation 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/nav_graph"
 app:startDestination="@id/gameFragment">
 <fragment
 android:id="@+id/gameFragment"
 android:name="com.hfad.guessinggame.GameFragment"
 android:label="fragment_game"
 tools:layout="@layout/fragment_game" >
 <action
 android:id="@+id/action_gameFragment_to_resultFragment"
 app:destination="@id/resultFragment"
 app:popUpTo="@id/gameFragment"
 app:popUpToInclusive="true" />
 </fragment>
 <fragment
 android:id="@+id/resultFragment"
 android:name="com.hfad.guessinggame.ResultFragment"
 android:label="fragment_result"
 tools:layout="@layout/fragment_result" >
 <argument
 android:name="result"
 app:argType="string" />
 <action
 android:id="@+id/action_resultFragment_to_gameFragment"
 app:destination="@id/gameFragment" />
 </fragment>
</navigation>
```

После обновления графа навигации свяжем его с активностью `MainActivity`, чтобы она отображала каждый фрагмент при переходе к нему.

Вывод текущего фрагмента в макете MainActivity

Для отображения фрагментов необходимо включить в макет `MainActivity` хост навигации, связанный с только что созданным графом навигации. Для этого мы воспользуемся представлением `FragmentContainerView`, как и в предыдущих главах. Код для решения этой задачи уже знаком вам по предыдущим главам. Обновите файл `activity_main.xml`, чтобы он соответствовал приведенному ниже:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.fragment.app.FragmentContainerView
 xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:app="http://schemas.android.com/apk/res-auto"
 android:id="@+id/nav_host_fragment"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:name="androidx.navigation.fragment.NavHostFragment"
 app:navGraph="@navigation/nav_graph"
 app:defaultNavHost="true" />
```

После обновления макета откройте файл `MainActivity.kt` и убедитесь в том, что он содержит следующий код:

```
package com.hfad.guessinggame
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
class MainActivity : AppCompatActivity() {
 override fun onCreate(savedInstanceState: Bundle?) {
 super.onCreate(savedInstanceState)
 setContentView(R.layout.activity_main)
 }
}
```

И это все, что необходимо сделать с `MainActivity`. На следующем шаге мы обновим код двух фрагментов, начиная с `GameFragment`.

Обновление макета GameFragment

`GameFragment` — главный экран приложения, на котором пользователь взаимодействует с игрой. В его макет необходимо добавить несколько представлений: три текстовых представления для угадываемого слова, количество оставшихся жизней и сделанных ошибочных предположений, текстовое поле для ввода буквы и кнопки для обработки предположения. Чтобы включить эти представления в макет, обновите код `fragment_game.xml`, чтобы он совпадал с приведенным ниже кодом:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
```

```
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    tools:context=".GameFragment">
    <TextView
    android:id="@+id/word"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:textSize="36sp"
    android:letterSpacing="0.1" />
    <TextView
    android:id="@+id/lives"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="16sp" />
    <TextView
    android:id="@+id/incorrect_guesses"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="16sp" />
    <EditText
    android:id="@+id/guess"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="16sp"
    android:hint="Guess a letter"
    android:inputType="text"
    android:maxLength="1" />
    <Button
    android:id="@+id/guess_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:text="Guess!"/>
  </LinearLayout>
```

И это все, что необходимо включить в макет `GameFragment`. Теперь нужно определить поведение игры, для чего мы внесем изменения в ее код Kotlin.

Что должен делать фрагмент GameFragment

В первой версии приложения файл `GameFragment.kt` должен включать весь код, необходимый для игры, а также для обновления экрана и обеспечения навигации. Он должен:

- Выбрать случайное слово. Мы определим список, из которого будет выбираться загаданное слово.
- Дать пользователю возможность ввести букву.

- Отреагировать на предположение. Если пользователь угадал правильно, фрагмент `GameFragment` должен добавить букву в список правильных предположений и показать, где в загаданном слове встречается эта буква. Если буква в слове не встречается, она добавляется в строку ошибочных предположений, а количество оставшихся жизней уменьшается.
- Перейти к `ResultFragment` при завершении игры.

Реализация состоит из кода Kotlin и кода Android, который уже встречался вам ранее. Полный код приводится на нескольких ближайших страницах.

Код GameFragment.kt

Ниже приводится код `GameFragment`; обновите свою версию файла `GameFragment.kt`, чтобы она соответствовала приведенной ниже:

```
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 com.hfad.guessinggame.databinding.FragmentGameBinding
import androidx.navigation.findNavController
class GameFragment : Fragment() {
 private var _binding: FragmentGameBinding? = null
 private val binding get() = _binding!!
 val words = listOf("Android", "Activity", "Fragment")
 val secretWord = words.random().uppercase()
 var secretWordDisplay = ""
 var correctGuesses = ""
 var incorrectGuesses = ""
 var livesLeft = 8
 override fun onCreateView(
 inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
 ): View? {
 _binding = FragmentGameBinding.inflate(inflater, container, false)
 val view = binding.root
 secretWordDisplay = deriveSecretWordDisplay()
 updateScreen()
 binding.guessButton.setOnClickListener() {
 makeGuess(binding.guess.text.toString().uppercase())
 binding.guess.text = null
 updateScreen()
 if (isWon() || isLost()) {
 val action = GameFragmentDirections
 .actionGameFragmentToResultFragment(wonLostMessage())
 view.findNavController().navigate(action)
 }
 }
 return view
 }
```

```
    override fun onDestroyView() {
    super.onDestroyView()
    _binding = null
    }
    fun updateScreen() {
    binding.word.text = secretWordDisplay
    binding.lives.text = "You have $livesLeft lives left."
    binding.incorrectGuesses.text = "Incorrect guesses: $incorrectGuesses"
    }
    fun deriveSecretWordDisplay() : String {
    var display = ""
    secretWord.forEach {
    display += checkLetter(it.toString())
    }
    return display
    }
    fun checkLetter(str: String) = when (correctGuesses.contains(str)) {
    true -> str
    false -> "_"
    }
    fun makeGuess(guess: String) {
    if (guess.length == 1) {
    if (secretWord.contains(guess)) {
    correctGuesses += guess
    secretWordDisplay = deriveSecretWordDisplay()
    } else {
    incorrectGuesses += "$guess "
    livesLeft--
    }
    }
    }
    fun isWon() = secretWord.equals(secretWordDisplay, true)
    fun isLost() = livesLeft <= 0
    fun wonLostMessage() : String {
    var message = ""
    if (isWon()) message = "You won!"
    else if (isLost()) message = "You lost!"
    message += " The word was $secretWord."
    return message
    }
  }
```

И это весь код, необходимый для `GameFragment`. Перейдем к коду `ResultFragment`.

Обновление макета ResultFragment

`ResultFragment` использует текстовое представление для вывода информации о том, выиграл он или проиграл, и кнопку для запуска следующей игры. Оба представления необходимо добавить в макет фрагмента. Этот код вам уже знаком. Откройте файл `fragment_result.xml` и обновите его содержимое, чтобы оно совпадало с приведенным ниже:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.com/tools"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:orientation="vertical"
 tools:context=".ResultFragment">
 <TextView
 android:id="@+id/won_lost"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:gravity="center"
 android:textSize="28sp" />
 <Button
 android:id="@+id/new_game_button"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_gravity="center"
 android:text="Start new game"/>
</LinearLayout>
```

Также необходимо обновить `ResultFragment.kt`. После того как текстовое представление и кнопка будут добавлены в макет `ResultFragment`, необходимо определить их поведение в коде фрагмента, написанном на Kotlin. Содержимое текстового представления должно обновляться результатом, а по щелчку на кнопке приложение должно возвращаться к `GameFragment`. Код `ResultFragment.kt` приведен на следующей странице.

Код ResultFragment.kt

Ниже приведен код `ResultFragment`; обновите содержимое файла `ResultFragment.kt`, чтобы оно соответствовало приведенному ниже:

```
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 com.hfad.guessinggame.databinding.FragmentResultBinding
import androidx.navigation.findNavController
class ResultFragment : Fragment() {
 private var _binding: FragmentResultBinding? = null
 private val binding get() = _binding!!
 override fun onCreateView(
 inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
 ): View? {
 _binding = FragmentResultBinding.inflate(inflater, container, false)
 val view = binding.root
 binding.wonLost.text = ResultFragmentArgs.fromBundle(requireArguments()).result
```

```
    binding.newGameButton.setOnClickListener {
    view.findNavController()
    .navigate(R.id.action_resultFragment_to_gameFragment)
    }
    return view
    }
    override fun onDestroyView() {
    super.onDestroyView()
    _binding = null
    }
}
```

И это весь код, необходимый для этой версии приложения `Guessing Game`. Давайте разберемся, что происходит при выполнении кода, и проведем тест-драйв приложения.

Что происходит при выполнении приложения

Во время выполнения приложения происходят следующие события:

1. Приложение запускается, и фрагмент `GameFragment` отображается в `MainActivity`. `GameFragment` присваивает `livesLeft` значение 8, `correctGuesses` и `incorrectGuesses` — пустую строку "", `secretWord` — случайно выбранное слово, а `secretWordDisplay`– значение `deriveSecretWordDisplay()`. Затем вызывается метод `updateScreen()`, который выводит значения `livesLeft`, `incorrectGuesses` и `secretWordDisplay`.

2. Когда пользователь вводит предположение, `GameFragment` вызывает свой метод `makeGuess()`. Метод проверяет, содержит ли `secretWord` букву, введенную пользователем. Если буква найдена, то `makeGuess()` добавляет букву в `correctGuesses` и обновляет `secretWordDisplay`. Если буква отсутствует, метод добавлят букву в `incorrectGuesses` и уменьшает `livesLeft` на 1. Затем метод `updateScreen()` снова вызывается для отображения новых значений.

3. После каждого предположения `GameFragment` проверяет результаты `isWon()` и `isLost()` на истинность. Эти методы проверяют, угадал ли пользователь все буквы слова или у него кончились жизни. Если один из двух методов возвращает true, то `GameFragment` передает результат фрагменту `ResultFragment` для вывода.

При запуске приложения отображается фрагмент `GameFragment`. Он показывает, сколько букв содержит загаданное слово и сколько жизней осталось у пользователя. Чтобы сделать предположение, пользователь вводит букву в текстовом поле и щелкает на кнопке. Если предположение оказалось правильным, буква включается в загаданное слово, но в случае ошибки теряется одна жизнь. Если пользователь угадал все буквы или у него кончились жизни, в `ResultFragment` выводится сообщение о выигрыше или проигрыше.