2.4. Элементы пользовательского интерфейса

Сайт: Samsung Innovation Campus
Курс: Мобильная разработка на Kotlin
Книга: 2.4. Элементы пользовательского интерфейса

Напечатано:: Murad Rezvan
Дата: понедельник, 3 июня 2024, 17:46

Оглавление

2.4.1. Графический интерфейс пользователя

Мобильное приложение - обязательно объектное приложение, а значит, обладает графическим интерфейсом пользователя (Graphical user interface, GUI) . Графический интерфейс традиционно содержит представления: текстовые поля и кнопки. Кроме этих объектов на активности могут размещаться стандартные объекты: календарь, часы, панели прогресса и другие виджеты. В разделе 2.3.3 уже упоминалось о связывании элемента из макета с переменной в коде. Но переменные для представлений можно так же создавать динамически в коде, не регистрируя их в макете активности. Для описания представления в макете используют xml- описания объектов из пакета виджетов, в коде оперируют объектами из пакета `android.view.View`, располагающемся в пакете `kotlin.Any`.

В xml-файле разметки описание объекта помещается в парные теги <Тип объекта> </Тип объекта>, а в java-файле необходимо импортировать стандартный класс из пакета android.widget.Тип_объекта, а затем создать константу этого типа. В программном коде доступ к объекту осуществляется посредством его id, установленного атрибутом android:id, поэтому данный атрибут необходимо устанавливать всем создаваемым представлениям. Создание объекта на основании макета представления осуществляется методом `findViewById()` посредством передачи ему во входном параметре id создаваемого объекта.

Описание в макете:

```
<View
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/sample"/>
```

Переменная в программе:

```
var view: View = findViewById(R.id.sample)
```

Можно объекты интерфейса создавать непосредственно в коде, не записывая их в макет. В этом случае представление добавляется в уже существующий родительский контейнер.

```
val root = findViewById<LinearLayout>(R.id.root)                //"нашли" контейнер
val button= Button(this)                                                    //создали кнопку
root.addView(button)                                                       //добавили кнопку в контейнер
```

В таком случае параметры view нужно настроить **до** его добавления в родительскую разметку.

2.4.2. Текстовые представления

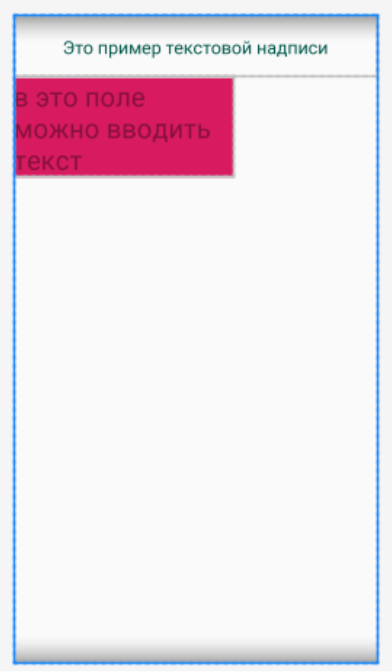
Различают два вида текстовых полей: текстовая константа `TextView` и редактируемый текст `EditText`. Оба представления работают только с типом данных String.

TextView предназначается для отображения на экране неизменяемого фрагмента текста.

EditText является полем ввода текстовых данных, у него тоже есть свойство `android:text`, которое заполняется с целью информирования пользователя о содержании вводимых данных. При запуске приложения этот текст уже установлен в текстовое поле и для ввода новых значений предыдущее нужно удалять. Чтобы исключить процедуру очистки поля от комментария, но при этом оставить информационное сообщение, используется свойство подсказки `android:hint` - текст, отображаемый в поле, пока оно не активно.

```
<TextView
    android:layout_width="220dp"
    android:layout_height="70dp"
    android:id="@+id/label"
    android:text="Это пример текстовой надписи" />

<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/read_text"
    android:hint="в это поле можно вводить текст"/>
```



Регистрация переменных в коде для этих текстовых полей:

```
val textWriter = findViewById<TextView>(R.id.label)
val textReader: EditText = findViewById(R.id.read_text)
```

Метод findViewById() не производит автоматического приведения типа, по этому ему нужно указывать класс, которому принадлежит представление. Это можно сделать при объявлении самой переменной либо передать тип в метод в качестве параметра.

После создания в программе экземпляра текстового класса, здесь же можно редактировать его атрибуты.

Изменить значение свойства *текст* можно с использованием сеттера

```
 textWriter.setText("Этот текст заменил прежнее значение")
```

или через обращение к свойству объекта

```
textWriter.text = "Ещё одно новое значение поля \"текст\""
```

Чтение данных из поля ввода EditText происходит функцией-геттером `getText()`. Данный метод возвращает значение типа Editable - надстройка над String, обладающая возможностью динамического изменения в процессе выполнения программы. Поскольку Editable не является строкой, то его необходимо приводить к String:

```
var inputText  = textReader.getText().toString()
```

При чтении числового значения сначала данные приводятся к типу String, а после строка сводится к нужному типу данных.

```
var inputText = textReader.getText().toString(); var sum = 0
for(v in 1 .. inputText.toInt()) sum +=v
textWriter.text= sum.toString()
```

2.4.3. Изображения

Графические объекты, в том числе формулы, помещаются в представления `ImageView` с атрибутом `android:src`, указывающим на ресурс, из которого выгружается изображение.

```
    <ImageView
        android:layout_width="249dp"
        android:layout_height="91dp"
        android:id="@+id/logo"
        android:src="@mipmap/logo" />
```

Изменяет источник изображения в коде метод `setImageResource()`

```
        val iv: ImageView = findViewById(R.id.logo)
        iv.setImageResource(R.drawable.ic_launcher_background)
```

Визуальные эффекты задаются свойствами

Свойство	Описание
rotation	поворот вокруг центра представления
rotationX	наклон изображения по вертикали
rotationY	наклон изображения по горизонтали
scaleX	растяжение по ширине
scaleY	растяжение по высоте
alpha	прозрачность (от 1 до 0)

Пример применения эффектов к изображению

```
    <ImageView
        android:layout_width="249dp"
        android:layout_height="91dp"
        android:id="@+id/logo"
        android:src="@mipmap/logo" />

     <ImageView
        android:layout_gravity="center_vertical"
        android:id="@+id/logo_new"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:alpha="0.5"
        android:rotation="25"
        android:scaleX="0.5"
        android:scaleY="3"
        android:src="@mipmap/logo" />
```

Рисунок 2.4.2 содержит графическое представление двух изображений, макет которых описан выше. Первое изображение без эффектов, второе с настроенными поворотом, растяжением/сжатием и прозрачностью.

Эти же эффекты можно настроить в коде после создания изображения: изменить источник картинки, настроить эффекты отображения.

```
val iv: ImageView = findViewById(R.id.logo)
      iv.setImageResource(R.drawable.ic_launcher_background)
      iv.rotation = 25f
      iv.scaleX = 1.5f
      iv.scaleY = 3f
      iv.alpha = 0.5f
```

2.4.4. Кнопка

Кнопка является управляющим элементом, определяющим действия, совершаемые в приложении. То есть, использование кнопки позволяет совершать изменения не при запуске приложения, а в процессе его работы. В макете кнопка описывается тегами <Button> </Button> и является экземпляром класса android.widget.Button.

```
<Button
        android:layout_width="234dp"
        android:layout_height="128dp"
        android:text="Кнопка"/>
```

Если на кнопке вместо текста предполагается разместить изображение, то используется объект кнопки - рисунка `ImageButton`

```
 <ImageButton
        android:layout_width="249dp"
        android:layout_height="91dp"
        android:id="@+id/picture"
        android:src="@mipmap/logo" />
```

Атрибут *android:text* у такого объекта заменяется полем `android:src`, содержащим имя ресурса с изображением для размещения на кнопке.

Кнопка с надписью на изображении реализуется обычной кнопкой с заданным фоном:

```
    <Button
        android:layout_width="234dp"
        android:layout_height="128dp"
        android:background="@mipmap/logo"
        android:text="Кнопка"/>
```

Упражнение 2.4.1. Добавление объектов в активность

1. Создайте активность с относительной разметкой ConstraintLayout (устанавливается по умолчанию). При добавлении объектов устанавливайте им привязки к границам активности.
2. Разместите на активности текстовое поле в верхнем левом углу, отцентруйте его надпись по правому краю.
3. Добавьте в ряд с текстовым полем поле ввода до правого края активности.
4. Ниже текстовых полей по центру разместите картинку с логотипом Академии.
5. По нижнему краю активности расположите три кнопки: с надписью "Кнопка №1", с иконкой приложения и с иконкой и надписью "Кнопка №3".
6. Выполните сохранение состояния активности, установив полю ввода свойство `android:id`. Проверьте сохранение, запустив приложение и выполнив поворот экрана.

Удалите у EditText атрибут *id* и убедитесь, что сохранение состояния не происходит.

2.4.5. Обработчики пользовательского взаимодействия

Экранные элементы мобильного приложения одновременно являются и управляющими. Это значит, что виджеты могут обрабатывать касания и нажатия - события взаимодействия пользователя с графическим интерфейсом приложения. Как правило, в качестве управляющих элементов используют кнопки, но и другие представления могут обрабатывать такие события. Для организации подобного взаимодействия в пакете android предусмотрены специальные классы *«слушателей» (Listeners).* Данные классы имплементируют соответственные интерфейсы с единственным методом обратного вызова, в теле которого размещается код, описывающий действия, совершаемые при взаимодействии пользователя с экранными элементами. В отличие от Java, Kotlin позволяет без описания слушателя и явного переопределения метода задать обработчик. Это значит, что среда сама создаст этот объект и «подставит» код в тело переопределяемого метода. Слушатель является свойством представления, по этому и настраивается с помощью сеттера

```
val b: Button=findViewById(R.id.button)
b.setOnClickListener( {
    b.text = “Ой! На меня нажали!!!”
} )
```

Виды обработчиков событий

По методу взаимодействия различают слушателей нажатий и касаний. Среди них так же выделяют класс «долгих» обработчиков, работающих при долгом нажатии на view.

Событие	Интерфейс обработчика	Используемый метод
Нажатие	OnClickListener	onClick(view: View!)
Долгое нажатие	OnLongClickListener	onLongClick(view: View!)
Касание	OnTouchListener	onTouch(view: View!, mEvent: MotionEvent!)

2.4.5.1 Обработчики нажатий

Нажатия "прикрепляются" к конкретным представлениям. То есть можно нажать на текстовое поле, на картинку и другие виджеты, но принято пользовательское управление приложением "доверять" кнопкам. Данный вид view интуитивно подводит к тому, чтобы на него нажимали.

Обработчик можно создать программно в момент задания свойства кликабельности представления:

```
b.setOnClickListener(View.OnClickListener {
            b.text = t.getText().toString()
        } )
```

Или без явного указания объекта - обработчика:

```
b.setOnClickListener( {
                b.text = t.getText().toString()
        } )
```

А можно просто определить свой метод с входным аргументом типа View и без возвращаемого параметра

```
fun my_click(v: View) {
        val b: Button=findViewById(R.id.button)
        val t: EditText=findViewById(R.id.read_text)
        val s = t.getText().toString()
    b.setText(s)
}
```

и в xml-описании разметки установить нужному элементу свойство `onClick`

```
<TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="my_click"/>
```

и тогда при нажатии на текстовое поле будет выполняться метод `my_click`.

Метод `onLongClick(view: View!)` отличается от `onClik(view: View!)` наличием возвращаемого логического значения. То есть при создании обработчика "долгого" нажатия необходимо возвращать логическое значение:

```
b.setOnLongClickListener({
         b.setBackgroundColor(resources.getColor(R.color.colorAccent))
            true         //возвратили логический тип
        })
```

или в собственном методе:

```
fun my_click(v: View) {
        val b: Button=findViewById(R.id.button)
        val t: EditText=findViewById(R.id.read_text)
        val s = t.getText().toString()
    b.setText(s)
true         //возвратили логический тип
}
```

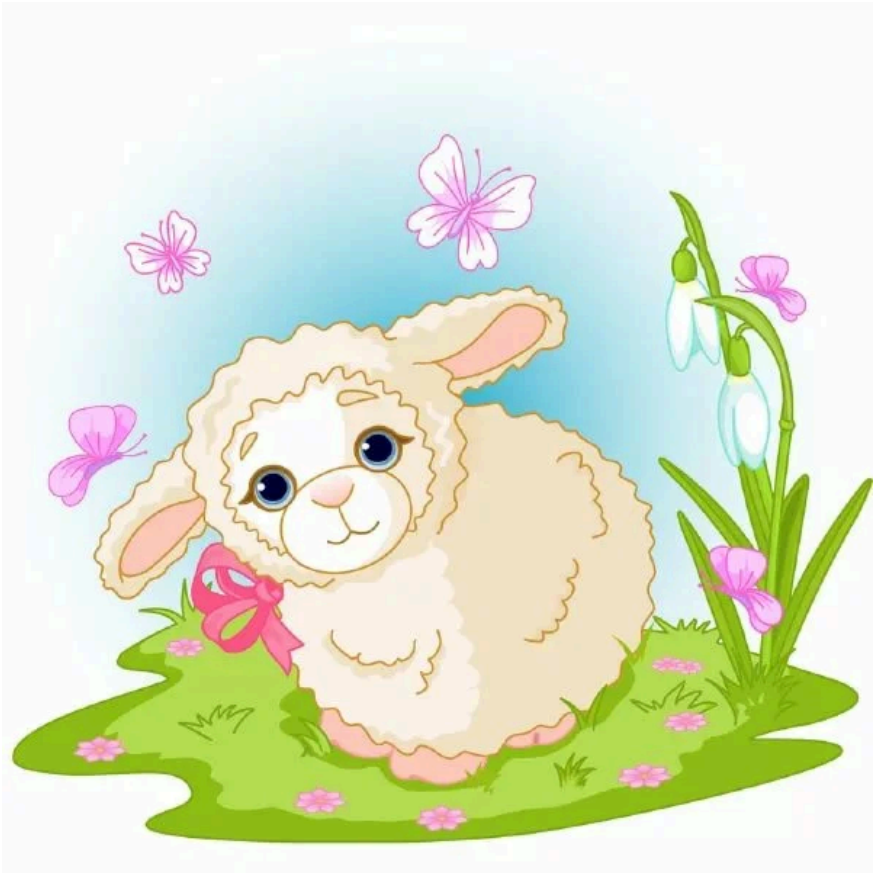
При этом в xml-описании свойство остаётся по-прежнему `android:onClick="my_click"`

Упражнение 2.4.2. Нажатия на элементы интерфейса

1. Задайте активности линейную разметку вертикальной ориентации.

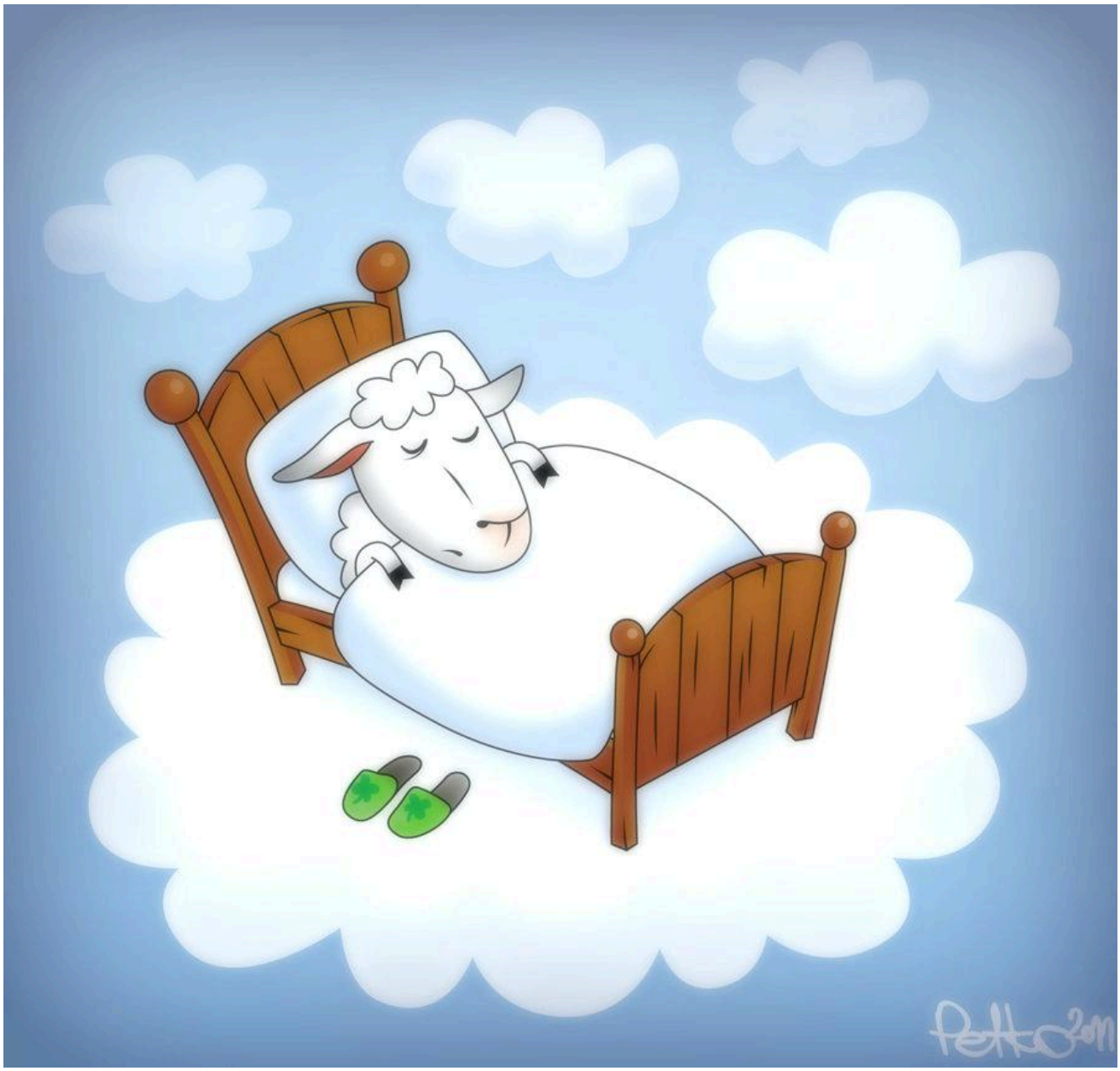
2. Разместите на экране

- текстовое поле `TextView` с надписью *Чтобы вечером быстро уснуть, нужно посчитать красивых овечек)))*
- кнопку `Button` с надписью *Сейчас овечек 0 штук*
- картинку `ImageView` с изображением овечки



3. Запрограммируйте кнопку так, чтобы при нажатии на неё увеличивался счётчик овечек.

4. Поставьте на картинку обработчик долгого нажатия, чтобы животное уснуло: замените картинку на спящую овечку



Готовый проект можно посмотреть здесь

2.4.5.2. Обработчик касаний

Нажатия на представления позволяют управлять событиями в приложении. Однако, у системы Android имеется механизм для управления самими представлениями: их перемещение по экрану.

Такое управление выполняет слушатель касаний - объект класса, имплементирующего интерфейс `onTouchListener` с методом обратного вызова с логическим значением:

`abstract fun onTouch(view: View!, mEvent: MotionEvent!): Boolean`.

Первым аргументом функции является "прослушиваемое" представление, вторым - способ касания по этому представлению. Различают три способа касания, для каждого из них зарезервирована константа класса событий *MotionEvent*

- прикасание к экрану (`MotionEvent.ACTION_DOWN`)
- перемещение пальца или стилуса по экрану (`MotionEvent.ACTION_MOVE`)
- отпускание экрана (`MotionEvent.ACTION_UP`)

При программировании обработчика касания могут быть определены все три события, но бывает, что определяют только одно или два типа касания.

Так же, как и для нажатия, для касаний требуется создание своего слушателя, однако, свойства касания у view нет, то есть обработчик можно установить только программно в коде - в макете настроить данный метод нельзя. И нужно помнить, что метод `onTouch()` возвращает логическое значение.

```
val iv: ImageView = findViewById(R.id.logo)
iv.setOnTouchListener { view, motionEvent ->
 if(motionEvent.getAction()==MotionEvent.ACTION_DOWN) // описывается, что делать при касании объекта
 if(motionEvent.getAction()==MotionEvent.ACTION_UP) // описывается, что делать при отпускании объекта
 if(motionEvent.getAction()==MotionEvent.ACTION_MOVE) //описывается, что делать при перемещении объекта
         true // обязательно указывается возвращаемое значение
     }
```

Кроме констант, определяющих тип касания, у объектов класса MotionEvent имеются свойства, считывающие координаты положения точки касания: `getX()` и `getY()` - вещественные числа. С полным списком полей и методов класса можно ознакомиться в документации

Упражнение 2.4.3. Использование касаний.

1. Добавьте в папку res/drawable/ проекта *My_app* семь любых картинок с именами: **{one.png, two.png, three.png, four.pg, five.png, six.png, seven.png}**

2. Создайте целочисленный массив с именами ресурсов картинок

```
 var m: Array<Int> = arrayOf(R.drawable.one, R.drawable.two, R.drawable.three, R.drawable.four, R.drawable.five, R.drawable.six, R.drawable.seven)
```

и целочисленную переменную для итерации изменений изображения

```
var i = 0
```

3. На ImageView установите и определите обработчик касаний:

```
 iv.setOnTouchListener { view, motionEvent ->
          when(motionEvent.getAction()) {
          MotionEvent.ACTION_DOWN -> iv.setImageResource(m[(i++) % m.size])
          MotionEvent.ACTION_UP -> iv.rotation = 45f * i
         }
         true
     }
```

4. Запустите приложение и выполните касания по картинке. **Обратите внимание, что смена картинок происходит при касании к экрану, а поворот при отпускании экрана.**

Сравните работу своего приложения с этим приложением

Домашнее задание. Программирование нажатий.

Создайте новое приложение с одной активностью и разместите на ней

- поля ввода логина и пароля: они должны задаваться в коде в виде констант и содержать подсказку (hint);
- кнопку «Вход»;
- текстовое поле, для отображения верно ли введен пароль: если верно вывести зеленым цветом «Верно», если не верно красным «Вы ошиблись в логине или пароле», при этом поля ввода очищаются.

Запрограммируйте кнопку «Вход» на выполнение описанных действий.

Начать тур для пользователя на этой странице