2.7. Элементы пользовательского интерфейса

Сайт: Samsung Innovation Campus
Курс: Мобильная разработка на Kotlin
Книга: 2.7. Элементы пользовательского интерфейса

Напечатано:: Murad Rezvan
Дата: понедельник, 3 июня 2024, 17:47

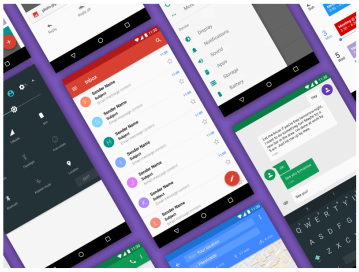
Описание

Поле ввода, чекбокс, радио кнопки, переключатель, всплывающие подсказки, всплывающие сообщения (pop-up message)

Оглавление

2.7.1 Продолжение знакомства с пользовательским интерфейсом

Пользовательский интерфейс (UI) приложения – это ключевая составляющая в мобильной разработке. UI – это то, что пользователь видит на экране и с чем взаимодействует. Как вы уже знаете, средства Android-разработки предоставляют большое количество готовых решений для пользовательского интерфейса, которые позволяют создавать графический интерфейс приложений. Средствами Android можно создавать поля ввода, чекбоксы, радио кнопки, спиннеры, переключатели, всплывающие подсказки, всплывающие сообщения (pop-up message). В этом уроке приведем описание некоторых элемнтов UI.

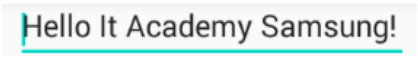


2.7.2 Поля ввода EditText

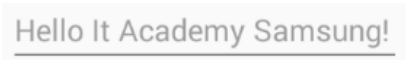
Во многих языках программирования существуют специальные формы для ввода текста. Не исключением является Kotlin Данный элемент интерфеса предназначен для ввода и изменения текста. Вы должны указать атрибут inputType. Например, для ввода простого текста установите значение `inputType` в значение поле текст *«text»* и когда Когда вы определяете виджет для редактирования текста возможно атрибут будет следующим:

```
<EditText
      android:id="@+id/plain_text_input"
      android:layout_height="wrap_content"
      android:layout_width="match_parent"
      android:inputType="text"
      android:inputType:@+id/textView1/>
```

На экране виждет может выглядеть так:



Текст для ввода может содержать подсказку (hint):



В зависимости от выбранного значения атрибута поле ввода настраивает отображаемый тип клавиатуры, допустимые символы и внешний вид текста. Например, если вы хотите принять секретный номер, например, уникальный пин-код или серийный номер, вы можете установить для атрибута `inputType` значение `numericPassword`.

Кроме этого можно установить размер текста с помощью атрибута `android:textSize`. При установке размера текста используются несколько единиц измерения:

- px — пиксели;
- dp — независимые от плотности пиксели. Это абстрактная единица измерения, основанная на физической плотности экрана;
- sp — независимые от масштабирования пиксели, т.е. зависимы от пользовательских настроек размеров шрифтов;
- in — дюймы, базируются на физических размерах экрана;
- pt — 1/72 дюйма, базируются на физических размерах экрана;
- mm — миллиметры, также базируются на физических размерах экрана.

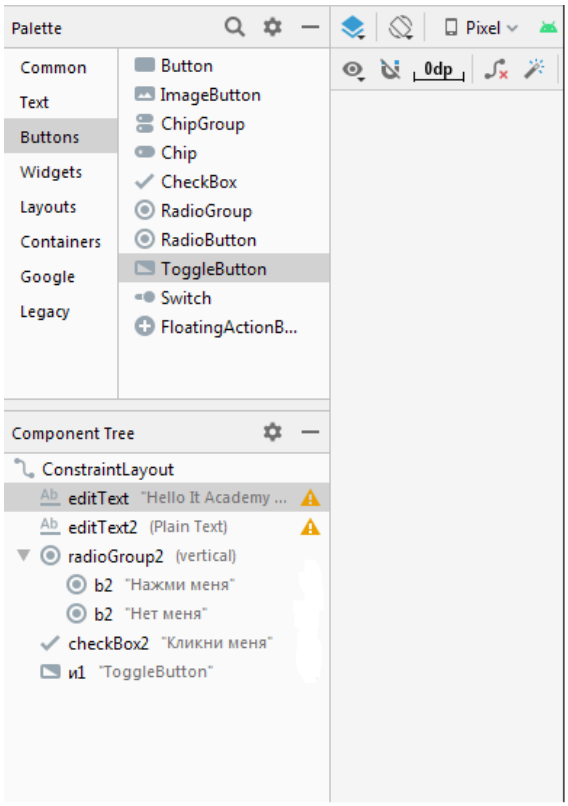
Обычно при установке размера текста используются единицы измерения sp, которые наиболее корректно отображают шрифты.

И д.т. Все ограничено Вашей фантазией Более подробно с с аттрибутами EditText можно ознакомится по ссылке

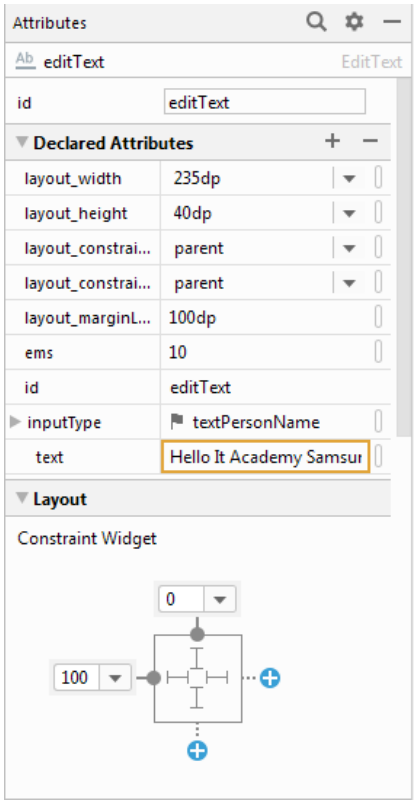
В коде языка Kotlin поиск `EditText` можно выполнять уже привычным образом:

```
var editTextHello = findViewById(R.id.editTextHello)
```

Кроме выбора характеристик всех виджетов и настройки их атрибутов в редакторе XML в Android Studio есть возможность делать это автоматически с помощью окон Pallete:



и Attributes:



2.7.3 Checkbox

Как вам уже всем известно `Checkbox` позволяют пользователю выбирать один из вариантов. один или несколько параметров из набора.

Приведем небольшой пример `Checkbox` в Kotlin. Предположим у нас есть такая разметка:

```
<CheckBox
        android:id="@+id/checkBox"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="10dp"
        android:text="@string/check_it"/>
```

Open in Playground → Target: JVM Running on v.2.0.0

В активити напишем следующее

```
class MainActivity : AppCompatActivity() {
     override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
         val checkBox = findViewById<CheckBox>(R.id.checkBox)
        checkBox?.setOnCheckedChangeListener { buttonView, isChecked ->
            val msg = "You have " + (if (isChecked) "checked" else "unchecked") + " this Check it Checkbox."
            Toast.makeText(this@MainActivity, msg, Toast.LENGTH_SHORT).show()
        }
    }
}
```

В этом примере получили доступ к `CheckBox` , используя его `id`. Затем мы добавили слушателя, который показывает `Toast`-сообщение, когда установлен / снят флажок.

2.7.5 ToggleButton

Кнопки переключения ToogleButtons позволяют пользователю менять настройки настройку между двумя состояниями приложения.

Вы можете добавить базовую кнопку-переключатель в разметку с помощью объекта `ToggleButton`. Начиная с Android 4.0 (уровень API 14) есть возможность задействовать другой тип переключателя, так называемый `Switch`, который обеспечивает управление ползунком, который вы можете добавить с помощью объекта `Switch`. `SwitchCompat` - данная версия работает на устройствах начиная с API 7. Рассмотрим его в следующей теме. Если вам нужно изменить состояние кнопки самостоятельно, вы можете использовать инструменты CompoundButton.setChecked() или CompoundButton.toggle().

Разметка для `ToogleButton` может выглядеть следующим образом:

```
<ToggleButton
        android:id="@+id/toggleButton2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textOff="NIGHT"
        android:checked="true"
        android:textOn="DAY" />
```

В коде Kotlin изменение атрибутов может быть таким:

```
val toggleButton = ToggleButton(this)
toggleButton.textOff = "ABC"
toggleButton.textOn = "DEF"
toggleButton.isChecked = true
```

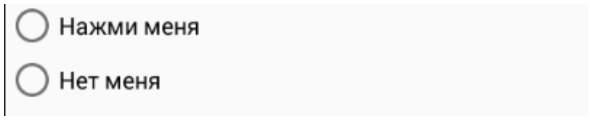
Для `ToogleButton` можно установить слушателя на изменение следующим образом:

```
 toggle_button.setOnCheckedChangeListener { buttonView, isChecked ->
            if (isChecked) {
                // The toggle is enabled/checked
                Toast.makeText(applicationContext,"Toggle on",Toast.LENGTH_SHORT).show()
            } else {
                // The toggle is disabled
                Toast.makeText(applicationContext,"Toggle off",Toast.LENGTH_SHORT).show()
            }
        }
```

Более подробно мы еще рассмотрим этот виджет, когда будем рассматривать практический пример.

2.7.4 Переключатели/включатели

Представления этого типа позволяют выбирать одну из предложенных позиций и относятся к java-классу `RadioGroup`. Группа переключателей в файле разметки выделяется тегами `<RadioGroup></RadioGroup>`, каждый из переключателей описывается тегами `<RadioButton></RadioButton>`. В группе активным может быть только один переключатель. Если теги не ставить, то кнопки будут независимы друг от друга. Так делать не стоит, поскольку для независимых переключателей предусмотрен класс `CheckBox`.



Когда пользователь выбирает один из переключателей, нажимает соответствующий объект `RadioButton` Чтобы определить по какому переключателю нажато, добавьте в разметку атрибут `android:onClick` . Значением этого атрибута должно быть имя метода, который вы хотите вызвать в ответ на событие нажатие переключателя. Например это можно сделать вот таким образом. Имеем файл разметки:

```
<?xml version="1.0" encoding="utf-8"?>
<RadioGroup xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton android:id="@+id/b1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="1"
        android:onClick="onClicked1"/>
    <RadioButton android:id="@+id/b2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="2"
        android:onClick="onClicked2"/>
</RadioGroup>
```

Затем в коде Kotlin пишем

```
fun onRadioButtonClicked(view: View) {
    if (view is RadioButton) {
        // Is the button now checked?
        val checked = view.isChecked

        // Check which radio button was clicked
        when (view.getId()) {
            R.id.b1 ->
                if (checked) {
                    // You pressed firstbtn
                }
            R.id.b2 ->
                if (checked) {
                    // You pressed second btn
                }
        }
    }
}
```

Метод, который вы объявляете в `onClick`, должен называться в точности так, как показано выше. В частности:

- быть публичным
- возврат недействительным
- с единственным параметром `view` как его единственный параметр (это будет вид, по которому щелкнули).

2.7.6 Всплывающие окна

Ранее в курсе мы уже сталкивались со всплывающими уведомлениями.Рассмотрим их подробнее. Всплывающее уведомление (Toast Notification) является сообщением, которое появляется на поверхности активности, заполняя необходимое ему количество пространства, требуемого для сообщения. При этом приложение не останавливается. Затем, в течении нескольких секунд сообщение закрывается. Для создания всплывающего уведомления необходимо инициализировать объект Toast при помощи метода `Toast.makeText()`, а затем вызвать метод `show()` для отображения сообщения на экране:

```
val text = "Hello!"
val duration = Toast.LENGTH_SHORT
val toast = Toast.makeText(applicationContext, text, duration)
toast.show()
```

Однако обычно можно записать в одну строчку, соединяя вызов методов в цепочку.

```
Toast.makeText(applicationContext, text, duration).show()
```

У `makeText()` есть три параметра:

- контекст
- текстовое сообщение;
- продолжительность.

В качестве параметра продолжительности можно использовать две константы:

- LENGTH_SHORT — показывает уведомление на короткий промежуток времени;
- LENGTH_LONG — показывает уведомление в течение длительного периода времени.

По умолчанию сообщение появляется в нижней центральной части экрана. Для размещения его иначе, вызовите метод `setGravity(int, int, int)` и задайте следующие параметры: константа `Gravity`, смещение по оси `X` и смещение по оси `Y`. Например так:

Например, если вы хотите разместить тост в верхнем левом углу экрана, определите атрибут Gravity следующим образом:

```
toast.setGravity(Gravity.TOP or Gravity.LEFT, 0, 0)
```

На самом деле у класса `Toast` есть множество возможностей. подробнее с ним можно ознакомится в официальной документации. или здесь

2.7.8 Всплывающие сообщения (pop-up message)

Во многих ситуациях вы можете захотеть, чтобы ваше приложение показывало пользователю быстрое сообщение, не ожидая ответа пользователя. Например, когда пользователь выполняет действие, такое как отправка сообщения или удаление файла, ваше приложение должно показать быстрое подтверждение (или отказ). При этом зачастую пользователю не нужно отвечать на сообщение. Всплывающее сообщение должно быть заметным, чтобы можно было его увидеть, но не настолько заметным, чтобы оно мешало работе.

Для этого Android предоставляет класс `Snackbar` . `Snackbar` предоставляет пользователю быстрое всплывающее сообщение (pop-up message). При этом работа с приложением продолжается. Через некоторое время `Snackbar` исчезает автоматически отключается.

Ранее мы рассматривали `Toast`-сообщения, но на сегодняшний день `Snackbar` является более предпочтительным классом для разработчиков. Все это потому, что у класса `Snackbar` несколько больше возможностей:

Важное примечание: чтобы использовать Snackbar в своем приложении для Android, вам нужно включить пакет «com.android.support.design» в ваши зависимости build.gradle (app). Существуют разные доступные версии пакета. Вам нужно включить тот, версия которого соответствует compileSdkVersion.

```
android {
    compileSdkVersion 26
    ...
}

dependencies {
    ...
    implementation 'com.android.support:design:26.1.0'
    ...
}
```

Не углубляясь в возможности класса `Snackbar` отобразим кнопку, при нажатии которой, `Snackbar` отображается внизу экрана. Разметка может выглядеть следующим образом:

```
<?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:orientation="vertical"
    android:gravity="center"
    android:background="#DDDDDD"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/btn"
        android:background="#FFFFFF"
        android:textAllCaps="false"
        android:padding="10sp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="HI! I am Snackbar"
         />
 </LinearLayout>
```

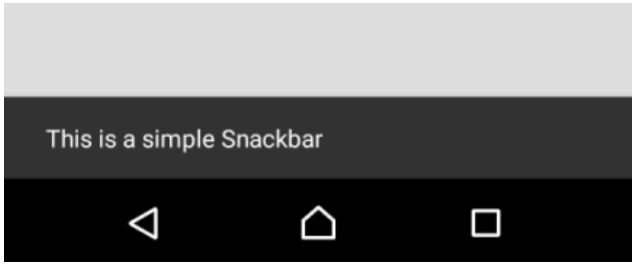
В коде Kotlin напишем:

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        btn.setOnClickListener {
            val snack = Snackbar.make(it,"This is a simple Snackbar",Snackbar.LENGTH_LONG)
            snack.show()
        }
    }
}
```

Тогда на экране появится следующее:


This is a simple Snackbar

Упражнение 2.7

Давайте разработаем приложение в котором продемонстрируем некоторые возможности изученных нами представлений. Пусть у нас имеется поле для ввода, две RadioButton, TuggleButton и чекбокс. С помощью радиокнопок будем вставлять в текстовое поле разный текст, с помощью TuggleButton менять задний фон EditText, а с помощью CheckBox показывать Toast-сообщение - активна она или нет.

Разметка может быть такой:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">


    <RadioGroup
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <RadioButton
            android:id="@+id/b1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="First" />

        <RadioButton
            android:id="@+id/b2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Second" />

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="horizontal">

            <EditText
                android:id="@+id/editText"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:ems="10"
                android:inputType="textPersonName"
                android:text="Name" />

            <CheckBox
                android:id="@+id/checkBox"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="Activate" />

            <ToggleButton
                android:id="@+id/toggleButton"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="Font" />

        </LinearLayout>
    </RadioGroup>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal"/>

</RelativeLayout>
```

В итоге должна получится такая картинка:

Сам код на Kotlin может выглядеть следующим образом.

```
package ru.samsung.itacademy.mdev

import android.graphics.Color
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.*

class MainActivity : AppCompatActivity() {


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val editText = findViewById<EditText>(R.id.editText)
        val checkBox = findViewById<CheckBox>(R.id.checkBox)
        val radioBtn1 = findViewById<RadioButton>(R.id.b1)
        val radioBtn2 = findViewById<RadioButton>(R.id.b2)
        val toogleBtn = findViewById<ToggleButton>(R.id.toggleButton)

        radioBtn1.setOnClickListener(View.OnClickListener {
            val s = "Hello world!"
            editText?.setText(s)
            Toast.makeText(applicationContext, "First RadioButton", Toast.LENGTH_SHORT)
                .show()
        })

        radioBtn2.setOnClickListener(View.OnClickListener {
            val s = "Greeatings!"
            editText?.setText(s)
            Toast.makeText(applicationContext, "Second RadioButton", Toast.LENGTH_SHORT)
                .show()
        })

        toogleBtn.setOnCheckedChangeListener { buttonView, isChecked ->
            if (isChecked) {
                // The toggle is enabled/checked
                Toast.makeText(applicationContext, "Toggle on", Toast.LENGTH_SHORT).show()


                editText.setBackgroundColor(Color.GREEN)
            } else {
                // The toggle is disabled
                Toast.makeText(applicationContext, "Toggle off", Toast.LENGTH_SHORT).show()
                editText.setBackgroundColor(Color.GRAY)
            }
        }

        checkBox?.setOnCheckedChangeListener(object : CompoundButton.OnCheckedChangeListener {
            override fun onCheckedChanged(buttonView: CompoundButton, isChecked: Boolean) {
                if (checkBox!!.isChecked) {
                    Toast.makeText(applicationContext, "checked ", Toast.LENGTH_SHORT).show()
                }
                else{
                    Toast.makeText(applicationContext, "unCkecked", Toast.LENGTH_SHORT).show()
                }
            }
        })
    }
}
```

Полную версию приложения можно посмотреть здесь

Начать тур для пользователя на этой странице