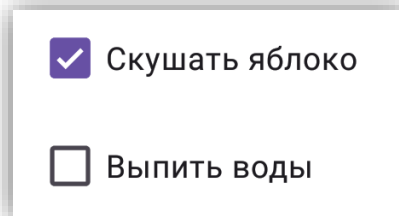


# Чекбоксы, радиокнопки, спиннеры

В данной теме будут рассмотрены элементы интерфейса, которые широко используются при написании приложений.

## Чекбоксы

Элемент управления `CheckBox` используется для двоичного выбора (да или нет):



В разметке элемент выглядит следующим образом (здесь и далее приводятся только атрибуты, которые существенны для понимания кода, и не приводятся привязки, атрибуты `layout_width` и `layout_height` и т. д.):

```
<CheckBox
    android:id="@+id/eat_apple"
    android:text="Скушать яблоко"
    android:checked="true" />
```

Атрибут `checked` является необязательным, он указывает будет ли элемент изначально включен или нет (т. е. будет ли стоять галочка при создании элемента, значения `true` / `false`).

В программном коде можно получить состояние элемента с помощью свойства `isChecked`:

```
val cbEatApple = findViewById<CheckBox>(R.id.eat_apple)
if (cbEatApple.isChecked) {
    // Действия если элемент включен
}
```

Если требуется немедленная реакция на включение или выключение элемента, то можно установить слушатель, который будет вызываться при изменении состояния элемента:

```
cbEatApple.setOnCheckedChangeListener { buttonView, isChecked ->
    // Какие-то действия
}
```

Первый параметр слушателя `buttonView` — это элемент, который вызвал срабатывание (то есть был включен или выключен), а второй `isChecked` — это новое состояние элемента. Конечно, если подключать слушатель в виде лямбда-функции, как в примере выше, то первый параметр всегда будет относиться к тому единственному элементу, к которому в коде подключен слушатель. Поэтому если однопоточных чекбоксов много, и всем им подходит один слушатель, то правильнее сделать слушателя, например, в виде анонимной функции, и подключать уже её:

```

val checkboxListener = { buttonView: CompoundButton, isChecked: Boolean ->
    when (buttonView.id) {
        R.id.eat_apple -> // Изменилось состояние чекбокса "Скушать яблоко"
        R.id.drink_water -> // Изменилось состояние чекбокса "Выпить воды"
    }
}

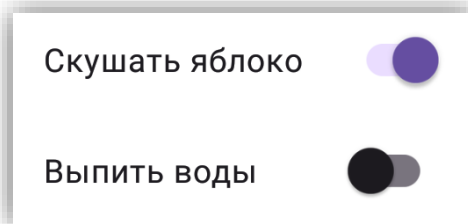
```

```

cbEatApple.setOnCheckedChangeListener(checkboxListener)
cbDrinkWater.setOnCheckedChangeListener(checkboxListener)

```

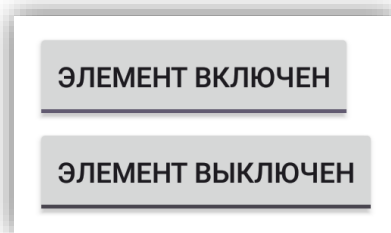
Вместо элемента `CheckBox` можно использовать элемент `SwitchMaterial` (или `SwitchCompat`, они отличаются очень незначительно):



Этот элемент больше соответствует стилю современных версий Android. Переключатель располагается сразу справа от текста, поэтому чтобы переключатели были выровнены друг под другом нужно задавать ширину для элементов `SwitchMaterial` не `wrap_content`, а либо `match_parent`, либо какое-то фиксированное значение.

В остальном данный элемент полностью аналогичен элементу `CheckBox`, работа с ним в коде также не отличается.

Ещё один элемент, который выполняет ту же роль – это `ToggleButton`. Он сочетает в себе свойства кнопки и чекбокса:



В разметке элемент выглядит следующим образом:

```

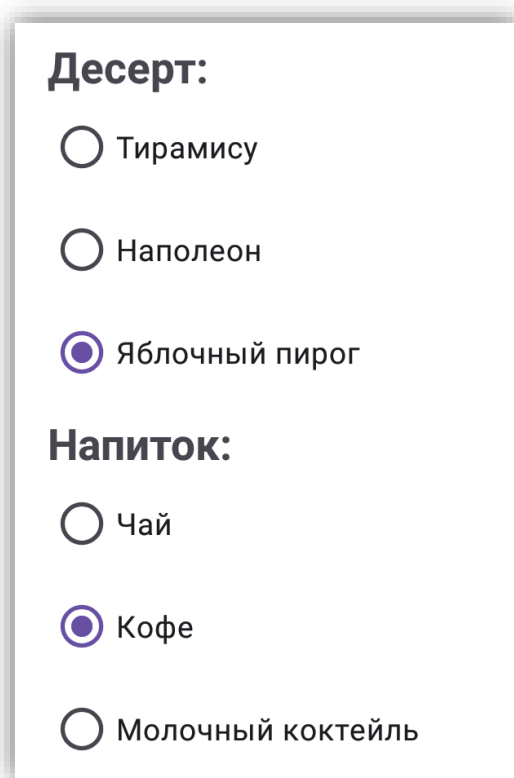
<ToggleButton
    android:textOn="Элемент включен"
    android:textOff="Элемент выключен"/>

```

Он содержит два атрибута: `textOn` содержит текст, который будет появляться на кнопке, когда она включена («нажата»), а `textOff` – когда она выключена («отжата»). В остальном работа с элементом полностью аналогична работе с чекбоксом.

## Радиокнопки

Элемент `RadioButton` используется для выбора из нескольких значений:



**Десерт:**

☐ Тирамису

☐ Наполеон

☒ Яблочный пирог

**Напиток:**

☐ Чай

☒ Кофе

☐ Молочный коктейль

В коде элемент выглядит следующим образом:

```
<RadioButton
    android:text="Яблочный пирог"
    android:checked="true"/>
```

Как и в случае с `CheckBox`, необязательный атрибут `checked` определяет будет ли элемент изначально включен или нет. Однако в случае `RadioButton` включенными может быть только один элемент.

Для того чтобы в интерфейсе могли быть несколько независимых групп радиокнопок (как в примере выше, где отдельно можно выбрать десерт, и отдельно напиток) их нужно объединить с помощью элемента `RadioGroup`:

```
<RadioGroup
    android:id="@+id/desserts"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <RadioButton
        android:id="@+id/tiramisu"
        android:text="Тирамису"/>

    <RadioButton
        android:id="@+id/napoleon"
        android:text="Наполеон"/>
```

```
<RadioButton
    android:id="@+id/apple_pipe"
    android:text="Яблочный пирог"
    android:checked="true"/>

</RadioGroup>
```

В каждой группе `RadioGroup` может быть включен только один элемент `RadioButton`.

При доступе из кода удобнее управлять не отдельными элементами `RadioButton`, а использовать для этого `RadioGroup`, в которую включены радиокнопки. Например, чтобы включить какой-то элемент, используется функция `check`, в которую передаётся идентификатор требуемого элемента:

```
val rgDesserts = findViewById<RadioGroup>(R.id.desserts)
rgDesserts.check(R.id.tiramisu)
```

При таком подходе важно чтобы в разметке у элементов `RadioButton` не использовался атрибут `checked`, в противном случае может получиться так, что выбранными окажутся сразу несколько элементов в группе. Вероятно, это баг в Android.

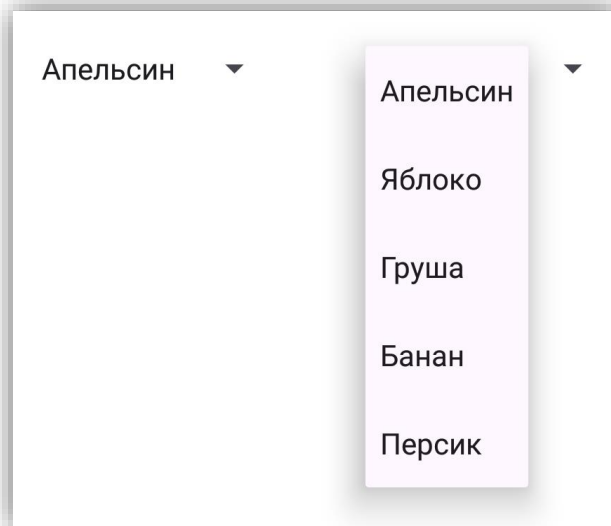
Для отслеживания выбора элемента также можно использовать слушатель у группы:

```
rgDesserts.setOnCheckedChangeListener { group, checkedId ->
    val title = when (checkedId) {
        R.id.tiramisu -> "Тирамису"
        R.id.napoleon -> "Наполеон"
        R.id.apple_pie -> "Яблочный пирог"
        else -> "Неизвестный десерт"
    }
}
```

Как и в случае с чекбоксами, можно реализовать слушатель в виде анонимной лямбда-функции и подключать его к нескольким радиогруппам. В этом случае параметр `group` будет определять какая группа вызвала срабатывание слушателя. Параметр `checkedId` содержит идентификатор в группе включенного элемента `RadioButton`.

## Спиннеры

Элемент `Spinner` представляет собой выпадающий список. В обычном состоянии он отображает выбранный элемент и стрелку справа. При нажатии на стрелку поверх элемента появляется выпадающее меню, в котором можно выбрать один из предложенных элементов:



В разметке элемент добавляется следующим образом:

```
<Spinner
    android:id="@+id/fruits"
    android:entries="@array/fruits"/>
```

Атрибут `entries` ссылается на строковый массив (в файле `res` → `values` → `strings.xml` или аналогичном), который содержит элементы выпадающего списка:

```
<string-array name="fruits">
    <item>Апельсин</item>
    <item>Яблоко</item>
    <item>Груша</item>
    <item>Банан</item>
    <item>Персик</item>
</string-array>
```

Если список элементов будет формироваться динамически во время работы программы (например, подгружаться из базы данных), то атрибут `entries` указывать не нужно. Вместо этого нужно подготовить список строк, и подключить их с помощью строкового адаптера:

```
val items = arrayOf("Апельсин", "Яблоко", "Груша", "Банан", "Персик")

val adapter = ArrayAdapter(this,
    android.R.layout.simple_spinner_item,
    items)
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)

val spinner = findViewById<Spinner>(R.id.fruits)
spinner.adapter = adapter
```

В этом примере создаётся адаптер – элемент, который берёт исходные данные (в данном случае список строк) и предоставляет его спиннеру в качестве элементов выпадающего списка. Такая схема кажется переусложнённой, но это лишь часть универсального механизма: некоторые адаптеры могут делать сложные преобразования данных,

сопоставлять данные сразу со многими элементами управления и т. д. Более подробно адаптеры будут разбираться в других темах.

Адаптер также содержит ссылки на разметку спиннера в закрытом виде, и разметку элемента выпадающего списка. В данном случае используются стандартные разметки, находящиеся в системных ресурсах (`android.R.layout.simple_spinner_item` и `android.R.layout.simple_spinner_dropdown_item`), но можно подготовить и реализовать свои разметки, чтобы создать нестандартно выглядящий элемент.

Прослушивание события смены выбранного элемента у спиннера немного сложнее, чем у предыдущих элементов: разработчики Android решили, что нужно реализовывать сразу два слушателя в одном, и если элемент выбран, и если не выбрано ничего. Поэтому создается объект-синглтон, который реализует особый интерфейс

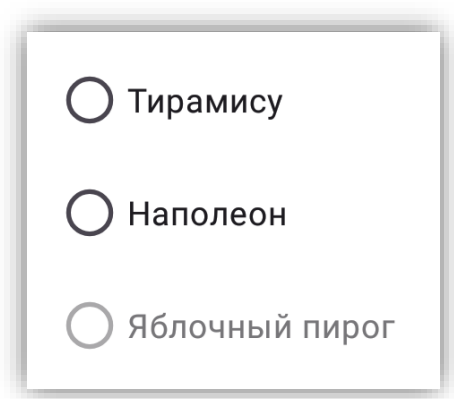
`AdapterView.OnItemSelectedListener`:

```
spinner.onItemSelectedListener = object : AdapterView.OnItemSelectedListener {  
    override fun onNothingSelected(parent: AdapterView<*>?) {  
        // Ничего не выбрано...  
    }  
  
    override fun onItemSelected(parent: AdapterView<*>?, view: View?,  
        position: Int, id: Long) {  
        // Выбран элемент с индексом position...  
    }  
}
```

Параметр `position` содержит порядковый номер элемента, он соответствует номеру элемента в исходном массиве.

## Доступность элементов

Иногда нужно чтобы какой-то элемент стал недоступен для взаимодействия с пользователем. Например, если в кафе закончились яблочные пироги, то можно заблокировать



В разметке для этого служит атрибут `enabled`:

```
<RadioButton  
    android:id="@+id/apple_pie"
```

```
android:enabled="false"  
android:text="Яблочный пирог" />
```

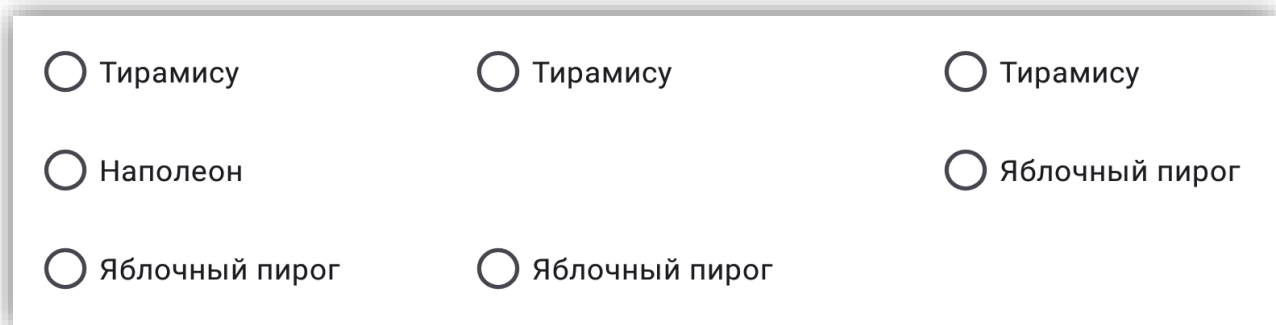
В коде программы узнать состояние элемента управления или изменить его можно с помощью свойства `isEnabled`:

```
if (spinner.isEnabled) {  
    // Действия если элемент доступен  
}  
spinner.isEnabled = false
```

Другой способ сделать элементы недоступными для пользователя – это скрыть их. Для этого в разметке служит атрибут `visibility`, он может принимать три значения:

- `visible` – элемент видим
- `invisible` – элемент скрыт, но пространство, которое он занимал, остаётся свободным
- `gone` – элемент скрыт, остальные элементы перестраиваются и занимают место скрытого элемента

Следующая иллюстрация показывает результат использования каждого из этих значений:



В коде видимость элемента можно поменять с помощью свойства с аналогичным названием `visibility`:

```
rbTiramisu.visibility = View.VISIBLE  
rbTiramisu.visibility = View.INVISIBLE  
rbTiramisu.visibility = View.GONE
```

## Задание

Разработайте приложения для регистрации пользователя. Форма регистрации должна выглядеть примерно так:

## Регистрация

Имя пользователя:

test

Пароль:

.....

Страна:

Россия

Номер телефона:

+7 (xxx) xxx xx xx

Показывать номер телефона:

☐ Всем пользователям

☒ Только друзьям

☐ Никому

☒ Согласен с условиями использования

Зарегистрироваться

Отказаться

Поле «Страна» содержит несколько стран, при выборе страны меняется телефонный код страны в поле с номером телефона.

Чекбокс «Согласен с условиями использования» делает кнопку «Зарегистрироваться» доступной или недоступной пользователю, в зависимости от того включен он или нет.

Кнопка «Отказаться» закрывает приложение.