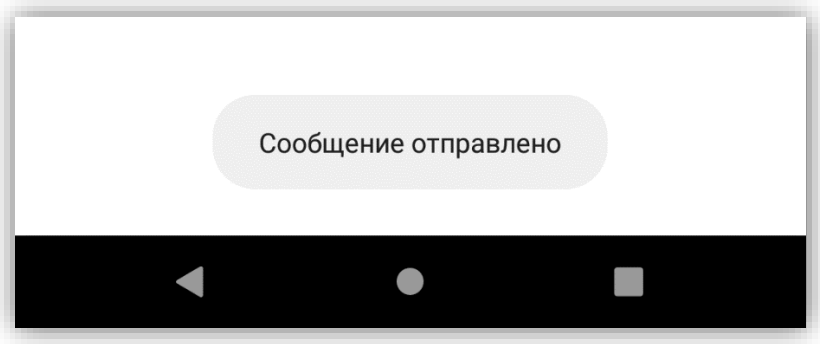
# Сообщения Toast и Snackbar. SeekBar

Иногда требуется показать пользователю краткое сообщение – как правило, результат операции, например, «Сообщение отправлено» или «Папка удалена». Для этого используются два системных компонента: простой `Toast` и обладающий продвинутыми возможностями `Snackbar`.

## Сообщения Toast

Компонент `Toast` показывает простое сообщение:




Для отображения сообщения нужно создать объект с помощью метода `makeText` в классе `Toast` и отобразить его с помощью метода `show`:

```
val toast = Toast.makeText(this, "Сообщение отправлено", Toast.LENGTH_SHORT)
toast.show()
```

Первый параметр метода `makeText` – контекст, поскольку сообщения часто показываются в активности или фрагменте, то как правило можно использовать ссылку `this`.

Второй параметр – это текст сообщения, который будет отображён. Здесь можно использовать как обычную строку, так и ссылку на строковый ресурс. Не следует использовать жёстко закодированные строки, это может усложнить локализацию приложения. Максимальная длина сообщения ограничена двумя строками, поэтому лучше использовать короткие сообщения.

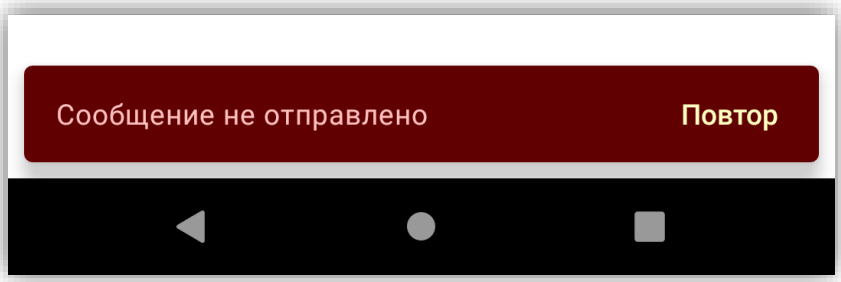
Третий параметр – это продолжительность отображения сообщения. Допустимы два параметра, определяемые константами: `Toast.LENGTH_SHORT` для краткого времени, и `Toast.LENGTH_LONG` для более продолжительного.

Можно объединить вызовы в цепочку, чтобы не создавать новую переменную:

```
Toast.makeText(this, "Сообщение отправлено", Toast.LENGTH_SHORT).show()
```

## Сообщения Snackbar

Компонент `Snackbar` тоже умеет показывать сообщения, как и `Toast`, но в отличие от него может использовать пользовательские цвета для текста и фона, в него можно добавлять кнопки для быстрых действий:




Для отображения простого сообщения нужно создать объект с помощью метода `make` класса `Snackbar` и отобразить его с помощью метода `show`:

```
val clMain = findViewById<ConstraintLayout>(R.id.main)
val sb = Snackbar.make(clMain, "Сообщение отправлено", Snackbar.LENGTH_SHORT)
sb.show()
```

Как видно из примера, в целом работа с сообщениями `Snackbar` похожа на работу с `Toast`, однако первый параметр представляет собой не контекст активности, а один из её элементов. В данном случае используется корневой контейнер `ConstraintLayout`, в котором уже располагаются все элементы управления, но можно использовать и любой другой элемент. Дело в том, что `Snackbar` предпочитает отображаться внутри контейнера `CoordinatorLayout`, который используется в Material Design, но если этот контейнер в активности отсутствует, то `Snackbar` будет подниматься вверх по иерархии элементов, пока не дойдёт до самой активности, и отобразится уже в ней. Поэтому в качестве первого параметра метода `make` можно использовать любой элемент управления в активности.

Продолжительность отображения сообщения схожа с тем что предлагает компонент Toast (`Snackbar.LENGTH_SHORT` и `Snackbar.LENGTH_LONG`), однако добавился ещё один вариант: бесконечное отображение сообщения, для этого служит константа `Snackbar.LENGTH_INDEFINITE`. Такое сообщение будет оставаться на экране до тех пор, пока пользователь не смахнёт его или не нажмёт кнопку действия, или же само приложение не вызовет у сообщения метод `dismiss` для удаления сообщения.

Если хочется использовать нестандартные цвета сообщения (например, красное для ошибок и зелёное для успешного выполнения операции), можно использовать методы `setBackgroundTint` для цвета фона и `setTextColor` для цвета текста сообщения:

```
val sb = Snackbar.make(...)
sb.setBackgroundTint(0xff600000.toInt())
sb.setTextColor(0xffffc0c0.toInt())
sb.show()
```

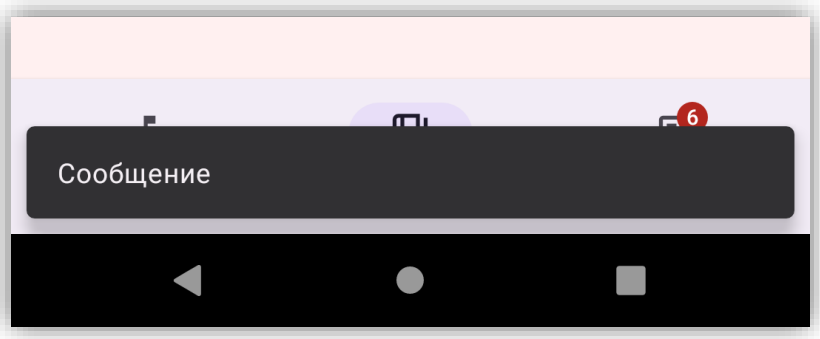
Цвет представляет собой целое число в формате `0xAARRGGBB`, причем альфа-канал в данном случае опускать нельзя, потому что иначе цвет получится прозрачным и результат будет совсем не таким как ожидалось. Однако полный цвет в формате `0xAARRGGBB` воспринимается компилятором как число типа `Long`, а методы ожидают число типа `Int`, поэтому используется явное преобразование типа с помощью метода `toInt`.

Для добавления кнопки действия используется метод `setAction`:

```
val sb = Snackbar.make(...)
sb.setAction("Повтор") {
    // Действия при нажатии на кнопку "Повтор"
}
sb.setActionTextColor(0xfffffffc0.toInt())
sb.show()
```

Цвет текста на кнопке действия можно задать с помощью метода `setActionTextColor`.

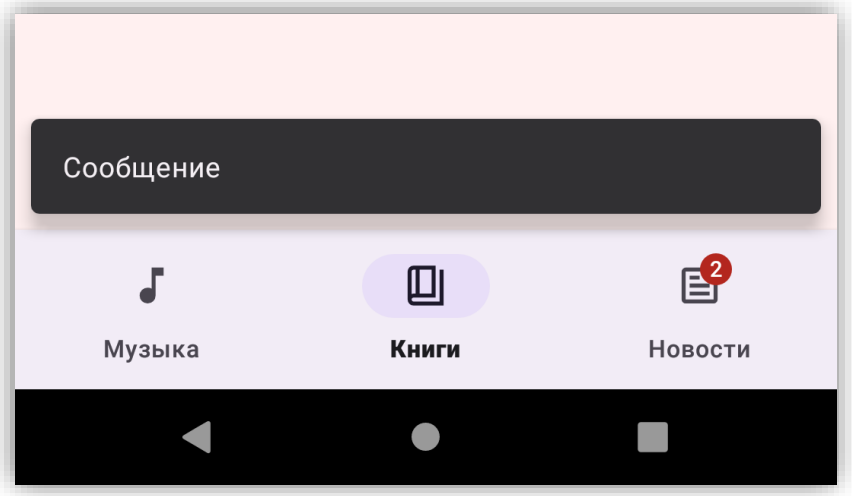
Иногда бывает полезно чтобы сообщение отображалось не в самом низу экрана, а над каким-то элементом. Например, если в нижней части экрана находится элемент `BottomNavigationView` с кнопками, и сообщение отобразится в нижней части экрана, то получится не очень красиво:




Поэтому лучше указать чтобы сообщение отображалось над элементом `BottomNavigationView`, для этого используется метод `setAnchorView`, которому в качестве параметра передаётся идентификатор элемента (или сам объект элемента) над которым будет показано сообщение:

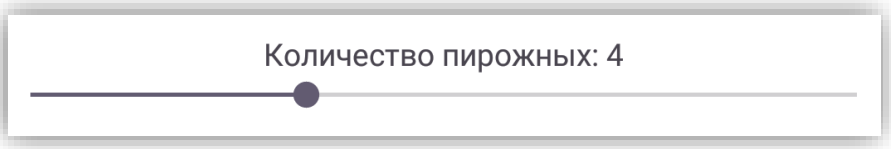
```
val sb = Snackbar.make(...)
sb.setAnchorView(R.id.bottomNav)
sb.show()
```

Результат будет следующий:



## Seekbar

Элемент управления `Seekbar` представляет собой ползунок, который пользователь может перемещать влево или вправо, при этом меняется внутренне значение элемента:




В разметке элемент добавляется следующим образом:

```
<SeekBar
    ...
    android:min="1"
    android:max="10"
    android:progress="3" />
```

Здесь `min` – это значение крайнего левого положения ползунка, `max` – значение крайнего правого, а `progress` – это текущее положение ползунка. Если значение `min` не задано, то оно равно нулю.

В коде узнать текущее (или изменить) значение элемента можно с помощью свойства `progress`:

```
val sbAmount = findViewById<SeekBar>(R.id.amount)
val amount = sbAmount.progress
```

Можно отслеживать изменение значения с помощью слушателя – он, увы, тоже не очень простой, и состоит из трёх вложенных методов:

```
val listener = object : SeekBar.OnSeekBarChangeListener {
    override fun onProgressChanged(seekBar: SeekBar?,
    progress: Int, fromUser: Boolean) {
        // Изменилось значение
    }

    override fun onStartTrackingTouch(seekBar: SeekBar?) {
        // Началось перетаскивание ползунка
    }

    override fun onStopTrackingTouch(seekBar: SeekBar?) {
        // Закончилось перетаскивание ползунка
    }
}

sbAmount.setOnSeekBarChangeListener(listener)
```

Не обязательно писать код во всех трёх методах, можно использовать только `onProgressChanged`, но присутствовать все три метода должны обязательно.

## Задание

Напишите приложение для тестирования компонента `Snackbar`.

Приложение должно позволять пользователю вводить текст сообщения и текст кнопки действия. Если текст кнопки действия не введён, то кнопка действия не должна отображаться.

Пользователь может изменять цвет фона, цвет текста сообщения и цвет текста кнопки действия с помощью ползунков `SeekBar`, каждый компонент (R, G и B) по отдельности.

При нажатии кнопки «Показать» отображается сообщение `Snackbar` с учётом всех заданных пользователем параметров.

При запуске приложения текстовые поля с текстами сообщения и кнопки действия, а также ползунки с цветами, должны содержать некоторые заранее заданные значения, чтобы можно было запустить демонстрацию `Snackbar` даже без изменения значений.

Активность приложения должна выглядеть примерно так: