Практическая работа 1. Построение основного контента OrderFragment

После добавления сворачиваемой панели инструментов в макет `OrderFragment` необходимо добавить дополнительные представления. С их помощью пользователь сможет выбрать вид пиццы и добавки (например, пармезан или масло чили); при щелчке на кнопке будет выводиться сообщение. Экран должен выглядеть примерно так:

Как видите, макет `OrderFragment` включает некоторые дополнительные представления, которыми вы еще не умеете пользоваться. Прежде чем строить макет, узнаем побольше об этих представлениях.

Выбор вида пиццы при помощи переключателя

Первое представление, которое мы используем, — группа переключателей, в которой пользователь выбирает вид пиццы.

Переключатели позволяют вывести набор вариантов, из которого выбирается один вариант, поэтому переключатели хорошо подходят для этой ситуации.

Для добавления переключателей в макет используются два элемента: `<RadioButton>` и `<RadioGroup>`. Элемент `<RadioButton>` используется для определения каждого переключателя, а элемент `<RadioGroup>` — для их группировки. Объединение переключателей в группу означает, что в любой момент времени в группе может быть установлен только один переключатель.

В приложении `Bits and Pizzas` должны отображаться переключатели Chilee (острый соус) и Funghi (грибы). Код выглядит так:

```
<RadioGroup
android:id="@+id/pizza_group"
android:layout_width="wrap_content"
android:layout_height="wrap_content">
<RadioButton android:id="@+id/radio_diavolo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Chilee" />
<RadioButton android:id="@+id/radio_funghi"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Funghi" />
</RadioGroup>
```

После того как вы определите группу и переключатели, можно написать код Kotlin для получения выбранного переключателя с использованием свойства `checkedRadioButtonId` группы. Его значением является идентификатор выбранного переключателя, или –1, если ни один переключатель не был выбран:

```
val pizzaGroup = view.findViewById<RadioGroup>(R.id.pizza_group)
val pizzaType = pizzaGroup.checkedRadioButtonId
if (pizzaType == -1) {
    //Выбранного элемента нет
} else {
    val radio = view.findViewById<RadioButton>(id)
    //Сделать что-то с переключателем
}
```

Переключатель — разновидность композитной кнопки

Во внутренней реализации переключатели наследуют от класса с именем `CompoundButton`, а тот является субклассом `Button`. `CompoundButton` представляет композитную кнопку с двумя состояниями: установленным и снятым (включенным и выключенным). Android включает другие разновидности композитных кнопок (помимо переключателей): флажки, селекторы и выключатели. Эти представления хорошо подходят для решений «да/нет»: например, «Добавить ли масло чили?» или «Хотите ли вы двойной пармезан?» Для добавления флажков, селекторов и выключателей в макеты используется код следующего вида:

```
<CheckBox
android:id="@+id/parmesan"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Parmesan" />
<Switch
android:id="@+id/switch_button"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
<ToggleButton
android:id="@+id/toggle_button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textOn="On"
android:textOff="Off" />
```

Свойство `isChecked` каждого представления можно использовать в коде Kotlin, чтобы узнать, находится ли он в установленном состоянии, как в следующем примере:

```
val parmesan = view.findViewById<CheckBox>(R.id.parmesan)
if (parmesan.isChecked) {
    //Что-то сделать
} else {
    //Сделать что-то другое
}
```

Плашка — разновидность композитной кнопки (Chips)

До настоящего момента мы рассматривали использование различных видов композитных кнопок — переключателей, флажков и селекторов. Еще одну, более гибкую разновидность композитных кнопок составляют плашки.

Это представление доступно при условии, что вы используете тему из библиотеки `Material`, например `Theme.MaterialComponents.DayNight.NoActionBar`.

Как и другие разновидности композитных кнопок, плашки используются для принятия решений «да/нет», но у них есть и другое применение: они также могут использоваться для ввода, фильтрации данных и выполнения действий.

В приложении `Bits and Pizzas` плашки будут использоваться для того, чтобы пользователь мог выбрать добавки (дополнительную порцию пармезана или масла чили) для своей пиццы. Для добавления плашек в макет используется код следующего вида:

```
<com.google.android.material.chip.Chip
android:id="@+id/parmesan"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Parmesan"
style="@style/Widget.MaterialComponents.Chip.Choice"/>
```

Ключевой частью кода плашки является атрибут `style`, управляющий ее внешним видом.

Код

```
style="@style/Widget.MaterialComponents.Chip.Choice"
```

в приведенном примере назначает для плашки стиль `Choice`, чтобы ее цвет изменялся при выборе. Другие возможные варианты — `Entry` (позволяет использовать плашки для ввода данных), `Filter` (для плашек, предназначенных для фильтрации данных) и `Action` (действует как кнопка).

Вот как выглядят плашки `Entry`, `Filter` и `Action`:

Помимо добавления одиночных плашек в макет, вы также можете группировать несколько плашек. Давайте посмотрим, как это делается.

Объединение нескольких плашек в группу

Если вы хотите, чтобы ваш макет включал несколько плашек, имеющих сходное назначение, включите их в группу плашек.

Группа плашек — разновидность группы представлений, предназначенная для аккуратного размещения нескольких плашек.

В приложении `Bits and Pizzas` используются две плашки добавок: для пармезана и для масла чили. Мы объединим их в группу следующим кодом:

```
<com.google.android.material.chip.ChipGroup
android:layout_width="wrap_content"
android:layout_height="wrap_content">
<com.google.android.material.chip.Chip
    android:id="@+id/parmesan"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Parmesan"
    style="@style/Widget.MaterialComponents.Chip.Choice"/>
<com.google.android.material.chip.Chip
    android:id="@+id/chili_oil"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Chili oil"
    style="@style/Widget.MaterialComponents.Chip.Choice"/>
</com.google.android.material.chip.ChipGroup>
```

Используйте isChecked для проверки состояния плашки

После того как вы добавите плашки в свой макет, для проверки, находится ли плашка в установленном состоянии, можно воспользоваться свойством `isChecked` каждой плашки — как и для других разновидностей композитных кнопок (селекторов, выключателей, флажков). Например, следующий код проверяет, выбрана ли плашка parmesan:

```
val parmesan = view.findViewById<Chip>(R.id.parmesan)
if (parmesan.isChecked) {
    //Что-то сделать
}
```

FAB — плавающая кнопка действия

Осталось последнее представление, о котором необходимо узнать, прежде чем мы сможем завершить макет `OrderFragment`: FAB.

FAB, или плавающая кнопка действия (Floating Action Button), круглая кнопка, которая «парит» над пользовательским интерфейсом. Она используется для привлечения внимания к стандартным или важным действиям. Как и обычную кнопку, кнопку FAB можно заставить реагировать на щелчки, назначив ей слушатель `OnClickListener` в коде Kotlin. Для добавления FAB в макет используется код следующего вида:

```
<androidx.coordinatorlayout.widget.CoordinatorLayout...>
...
<androidx.core.widget.NestedScrollView...>
    ...
</androidx.core.widget.NestedScrollView>
<com.google.android.material.floatingactionbutton.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|end"
    android:layout_margin="16dp"
    android:src="@android:drawable/ic_menu_send" />
</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

Приведенный выше код использует атрибут `layout_gravity` для закрепления FAB в нижнем углу экрана устройства с отступом 16dp.

Строка:

```
android:src="@android:drawable/ic_menu_send"
```

добавляет значок в FAB. В приведенном примере отображается один из встроенных значков Android с именем `ic_menu_send`, но вы можете использовать любую разновидность графического объекта при условии, что он помещается на FAB. Обычно кнопки FAB используются внутри координирующего макета, чтобы вы могли координировать перемещения между разными представлениями вашего макета. Рассмотрим пример.

FAB можно прикрепить к сворачиваемой панели инструментов

Кнопки FAB часто размещаются в нижнем конечном углу экрана, но их также можно прикрепить к другому представлению, например сворачиваемой панели инструментов. Когда вы это делаете, FAB перемещается со сворачиваемой панелью инструментов при ее сворачивании и расширении:

Код макета приведен ниже. Как видите, в нем используются атрибуты FAB `app:layout_anchor` и `app:layout_anchorGravity` для прикрепления FAB к нижнему конечному углу сворачиваемой панели инструментов:

```
<androidx.coordinatorlayout.widget.CoordinatorLayout...>
...
<com.google.android.material.floatingactionbutton.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_anchor="@id/collapsing_toolbar"
    app:layout_anchorGravity="bottom|end"
    android:layout_margin="16dp"
    android:src="@android:drawable/ic_menu_send" />
</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

После знакомства с переключателями, плашками, FAB и другими представлениями можно переходить к построению основного содержимого макета `OrderFragment`.

Построение макета OrderFragment

В макет фрагмента `OrderFragment` необходимо добавить представления, чтобы пользователь мог выбрать вид пиццы и включить добавки.

Макет должен выглядеть примерно так:

Весь код, необходимый для создания макета, вам уже знаком, поэтому мы можем перейти к обновлению `fragment_order.xml`. Полный код будет приведен ниже.

Полный код fragment_order.xml

Ниже приведен полный код макета фрагмента `OrderFragment`. Обновите код файла `fragment_order.xml`:

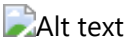
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.coordinatorlayout.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".OrderFragment">
    <com.google.android.material.appbar.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/ThemeOverlay.MaterialComponents.Dark.ActionBar">
        <com.google.android.material.appbar.CollapsingToolbarLayout
            android:layout_width="match_parent"
            android:layout_height="300dp"
            app:layout_scrollFlags="scroll|exitUntilCollapsed"
            app:contentScrim="?attr/colorPrimary">
            <ImageView
```

```
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:scaleType="centerCrop"
                android:src="@drawable/restaurant"
                app:layout_collapseMode="parallax"/>
            <com.google.android.material.appbar.MaterialToolbar
                android:id="@+id/toolbar"
                android:layout_width="match_parent"
                android:layout_height="?attr/actionBarSize"
                app:layout_collapseMode="pin" />
        </com.google.android.material.appbar.CollapsingToolbarLayout>
    </com.google.android.material.appbar.AppBarLayout>
    <androidx.core.widget.NestedScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior">
        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:padding="16dp"
            android:orientation="vertical">

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Which type of pizza would you like?" />
            <RadioGroup
                android:id="@+id/pizza_group"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content">
                <RadioButton android:id="@+id/radio_chilee"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="Chilee" />
                <RadioButton android:id="@+id/radio_funghi"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="Funghi" />
            </RadioGroup>
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Please choose any extras:" />

            <com.google.android.material.chip.ChipGroup
                android:layout_width="wrap_content"
                android:layout_height="wrap_content">
                <com.google.android.material.chip.Chip
                    android:id="@+id/parmesan"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="Parmesan"
                    style="@style/Widget.MaterialComponents.Chip.Choice"/>
                <com.google.android.material.chip.Chip
```

```
                    android:id="@+id/chili_oil"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="Chili oil"
                    style="@style/Widget.MaterialComponents.Chip.Choice"/>
            </com.google.android.material.chip.ChipGroup>
        </LinearLayout>
    </androidx.core.widget.NestedScrollView>
    <com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/fab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|end"
        android:layout_margin="16dp"
        android:src="@android:drawable/ic_menu_send" />
</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

И это весь код макета, который нам понадобится. Давайте проведем тест-драйв приложения и посмотрим, как оно выглядит.

При запуске приложения в `MainActivity` отображается фрагмент `OrderFragment`. Он включает сворачиваемую панель инструментов, как и прежде, но на этот раз в основном содержимом присутствуют текстовые представления, переключатели, плашки и FAB. Когда вы прокручиваете экран устройства, основное содержимое прокручивается вверх, и панель инструментов сворачивается. Кнопка FAB остается зафиксированной в правом конечном углу экрана.

 Alt text

Коммит

- git add .
- git commit -m "Chips, Fab"

Как заставить FAB реагировать на щелчки

Мы построили макет `OrderFragment`, но когда пользователь щелкает на кнопке FAB, ничего не происходит. Обновим код Kotlin фрагмента, чтобы кнопка FAB реагировала на щелчки. FAB должна решать две задачи:

1. Если вид пиццы не выбран, вывести сообщение. Пользователь должен выбрать, какой вид пиццы он желает заказать. Если он щелкнет на FAB, не выбрав его, на экране появляется всплывающее сообщение (`Toast`) с предложением выбрать вид пиццы.

2. Вывести заказ в отдельном сообщении. Если пользователь выбрал вид пиццы, будет выводиться сообщение с описанием заказа. Для этой цели будет использоваться другая разновидность всплывающих сообщений — `Snackbar`.

Для начала заставим FAB реагировать на щелчки.

Добавление OnClickListener к FAB

Как было сказано выше, реакция FAB на щелчки реализуется так же, как и для других видов кнопок: присоединением `OnClickListener` к FAB.

Ниже приведен код добавления `OnClickListener` к FAB в `OrderFragment`; как видите, он не отличается от кода, используемого для обычных кнопок:

```
val fab = view.findViewById<FloatingActionButton>(R.id.fab)
fab.setOnClickListener {
    //Код, выполняемый по щелчку на FAB
}
```

Теперь сделаем так, чтобы слушатель `OnClickListener` выводил сообщение, если пользователь не выбрал вид пиццы.

Проверка выбранного вида пиццы

Чтобы узнать, выбрал ли пользователь вид пиццы, можно воспользоваться свойством `checkedRadioButtonId` группы `pizzas_group`. Значение свойства содержит идентификатор выбранного переключателя, если он выбран, или –1, если пользователь еще не принял решение.

Следующий код проверяет, не щелкнул ли пользователь на FAB, не выбрав вид пиццы:

```
val fab = view.findViewById<FloatingActionButton>(R.id.fab)
fab.setOnClickListener {
    val pizzaGroup = view.findViewById<RadioGroup>(R.id.pizza_group)
    val pizzaType = pizzaGroup.checkedRadioButtonId
    if (pizzaType == -1) {
        //Пицца не выбрана, вывести сообщение
    } else {
        //Вывести заказ в другом сообщении
    }
}
```

Если пользователь не выбрал вид пиццы, на экране должно появиться всплывающее сообщение `Toast`. Посмотрим, как это делается.

Toast — простое всплывающее сообщение

Если пользователь щелкает на FAB, не выбрав вид пиццы, на экране устройства должно появиться сообщение `Toast`. `Toast` — простое всплывающее сообщение, которое предоставляет пользователю информацию и автоматически исчезает по истечении заданного времени:

Сообщение создается вызовом `Toast.makeText()`. Метод `makeText` получает три параметра: `Context` (обычно this или активность в зависимости от того, вызывается ли `Toast` из активности или фрагмента), `CharSequence!` (выводимое сообщение) и промежуток времени. Затем сообщение отображается на экране вызовом метода `show()` сообщения. Следующий пример кода на короткое время выводит сообщение `Toast` на экран:

```
val text = "Hello, I'm a toast!"
Toast.makeText(activity, text, Toast.LENGTH_SHORT).show()
```

Добавление Toast к слушателю OnClickListener

Сообщение `Toast` должно выводиться в том случае, если пользователь щелкает на FAB без выбора вида пиццы. Код решения этой задачи приведен ниже:

```
val fab = view.findViewById<FloatingActionButton>(R.id.fab)
fab.setOnClickListener {
    val pizzaGroup = view.findViewById<RadioGroup>(R.id.pizza_group)
    val pizzaType = pizzaGroup.checkedRadioButtonId
    if (pizzaType == -1) {
        val text = "You need to choose a pizza type"
        Toast.makeText(activity, text, Toast.LENGTH_LONG).show()
    } else {
        //Вывести заказ в другом сообщении
    }
}
```

Вот и все, что необходимо сделать, если пользователь не выбрал вид пиццы. Теперь напишем код для вывода заказа.

Вывод заказа в сообщении Snackbar

Если пользователь выбрал вид пиццы, то его заказ по щелчку на FAB должен выводиться во всплывающем сообщении, которое называется `Snackbar`. Сообщение `Snackbar` похоже на `Toast`, но обладает большей интерактивностью. Например, его можно смахнуть с экрана или заставить что-то делать по щелчку. Сообщения `Snackbar` создаются вызовом `Snackbar.make()`. Метод `make` получает три параметра: представление, инициирующее появление `Snackbar` (в данном случае FAB), `CharSequence`! (выводимый текст) и промежуток времени. Чтобы сообщение `Snackbar` появилось на экране, вызовите его метод `show()`.

Следующий пример на короткое время выводит сообщение на экран:

```
val text = "Hello, I'm a snackbar!"
Snackbar.make(fab, text, Snackbar.LENGTH_SHORT).show()
```

В приведенном выше коде значение LENGTH_SHORT используется для отображения Snackbar на короткое время. Другие возможные варианты — LENGTH_LONG (сообщение выводится на долгое время) и LENGTH_INDEFINITE (сообщение остается на экране неопределенно долго).

У Snackbar могут быть действия

При желании к Snackbar можно добавить действие, чтобы пользователь мог, скажем, отменить только что выполненную операцию. Для этого перед вызовом show() следует вызвать метод setAction() сообщения Snackbar. setAction получает два параметра: текст, который должен выводиться для действия, и лямбда-выражение, которое выполняется по щелчку на действии. Пример кода Snackbar с действием:

```
Snackbar.make(fab, text, Snackbar.LENGTH_SHORT)
 .setAction("Undo") {
 //Код, выполняемый по щелчку на кнопке
 }
 .show()
```

Сообщения Snackbar обычно отображаются в нижней части экрана, но вы можете переопределить это поведение при помощи метода setAnchorView() сообщения. Метод прикрепляет Snackbar к конкретному представлению, чтобы сообщение выводилось над ним. Например, такая возможность может оказаться полезной, если сообщение Snackbar должно находиться над нижней панелью навигации:

```
Snackbar.make(fab, text, Snackbar.LENGTH_SHORT)
 .setAnchorView(bottomNavBar)
 .show()
```

Код Snackbar для заказа пиццы

Теперь вы знаете, как создаются сообщения Snackbar, и мы можем написать код для вывода заказанной пиццы. Мы будем отображать сообщение Snackbar с видом пиццы, выбранным пользователем, а также всеми добавками (например, пармезан или масло чили).

Код вывода сообщения Snackbar:

```
val fab = view.findViewById<FloatingActionButton>(R.id.fab)
fab.setOnClickListener {
    val pizzaGroup = view.findViewById<RadioGroup>(R.id.pizza_group)
    val pizzaType = pizzaGroup.checkedRadioButtonId
    if (pizzaType == -1) {
        val text = "You need to choose a pizza type"
        Toast.makeText(activity, text, Toast.LENGTH_LONG).show()
    } else {
        var text = (when (pizzaType) {
```

```
            R.id.radio_diavolo -> "Diavolo pizza"
            else -> "Funghi pizza"
        })
        val parmesan = view.findViewById<Chip>(R.id.parmesan)
        text += if (parmesan.isChecked) ", extra parmesan" else ""
        val chiliOil = view.findViewById<Chip>(R.id.chili_oil)
        text += if (chiliOil.isChecked) ", extra chili oil" else ""
        Snackbar.make(fab, text, Snackbar.LENGTH_LONG).show()
    }
}
```

И это весь код Kotlin, необходимый для приложения `Bits and Pizzas`.

Рассмотрим полный код `OrderFragment.kt` и проведем тест-драйв приложения.

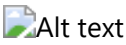
Полный код OrderFragment.kt

Ниже приведен полный код `OrderFragment.kt`; обновите свою версию:

```
package com.example.android.bitsandpizzas
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.appcompat.app.AppCompatActivity
import com.google.android.material.appbar.MaterialToolbar
import android.widget.RadioGroup
import com.google.android.material.chip.Chip
import com.google.android.material.floatingactionbutton.FloatingActionButton
import android.widget.Toast
import com.google.android.material.snackbar.Snackbar
class OrderFragment : Fragment() {
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?, savedInstanceState:
Bundle?
    ): View? {
        val view = inflater.inflate(R.layout.fragment_order, container, false)
        val toolbar = view.findViewById<MaterialToolbar>(R.id.toolbar)
        (activity as AppCompatActivity).setSupportActionBar(toolbar)
        val fab = view.findViewById<FloatingActionButton>(R.id.fab)
        fab.setOnClickListener {
            val pizzaGroup = view.findViewById<RadioGroup>(R.id.pizza_group)
            val pizzaType = pizzaGroup.checkedRadioButtonId
            if (pizzaType == -1) {
                val text = "You need to choose a pizza type"
                Toast.makeText(activity, text, Toast.LENGTH_LONG).show()
            } else {
                var text = (when (pizzaType) {
                    R.id.radio_chilee -> "Chilee pizza"
```

```
                    else -> "Funghi pizza"
                })
                val parmesan = view.findViewById<Chip>(R.id.parmesan)
                text += if (parmesan.isChecked) ", extra parmesan" else ""
                val chiliOil = view.findViewById<Chip>(R.id.chili_oil)
                text += if (chiliOil.isChecked) ", extra chili oil" else ""
                Snackbar.make(fab, text, Snackbar.LENGTH_LONG).show()
            }
        }
        return view
    }
}
```

Если щелкнуть на FAB, не выбрав вид пиццы, появляется сообщение `Toast` с предложением выбрать его. Если выбрать вид пиццы и снова щелкнуть на FAB, в нижней части экрана появляется сообщение `Snackbar` с подробной информацией о заказе.

 Alt text

Резюме

- Используйте `CoordinatorLayout` для координации анимации между представлениями.
- Используйте `AppBarLayout` для анимации панелей инструментов.
- Используйте `NestedScrollView`, если вы хотите, чтобы представления реагировали на прокрутку экрана устройства.
- Используйте `CollapsingToolbarLayout` для добавления панелей инструментов, которые сворачиваются и расширяются в соответствии с действиями пользователей.
- Чтобы определить группу переключателей, сначала создайте элемент `<RadioGroup>`, а затем поместите отдельные переключатели `<RadioButton>` в группу.
- Переключатели, выключатели, селекторы и флажки являются разновидностями композитных кнопок. Композитная кнопка может находиться в установленном или снятом состоянии.
- Плашка — более гибкая композитная кнопка, которая может использоваться для принятия решений, ввода данных пользователем, фильтрации данных и выполнения действий.
- Несколько плашек можно объединить в группу.
- Используйте FAB (плавающая кнопка действия) для отображения типичных или особенно важных действий пользователя.
- Toast — разновидность всплывающих сообщений.
- Snackbar — другая разновидность всплывающих сообщений, предназначенных для взаимодействия с пользователем.