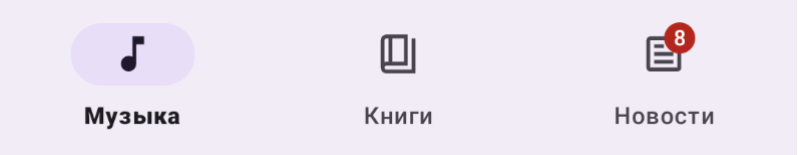
Вкладки

Современные приложения часто содержат разнородную информацию, которую желательно разделить по логике содержимого. Да и самой информации обычно бывает много, она не всегда умещается на одном экране. Использование вкладок позволяет структурировать информацию по типу.

В приложениях на Android вкладки бывают нескольких типов, для каждого из них есть свои рекомендации по использованию.

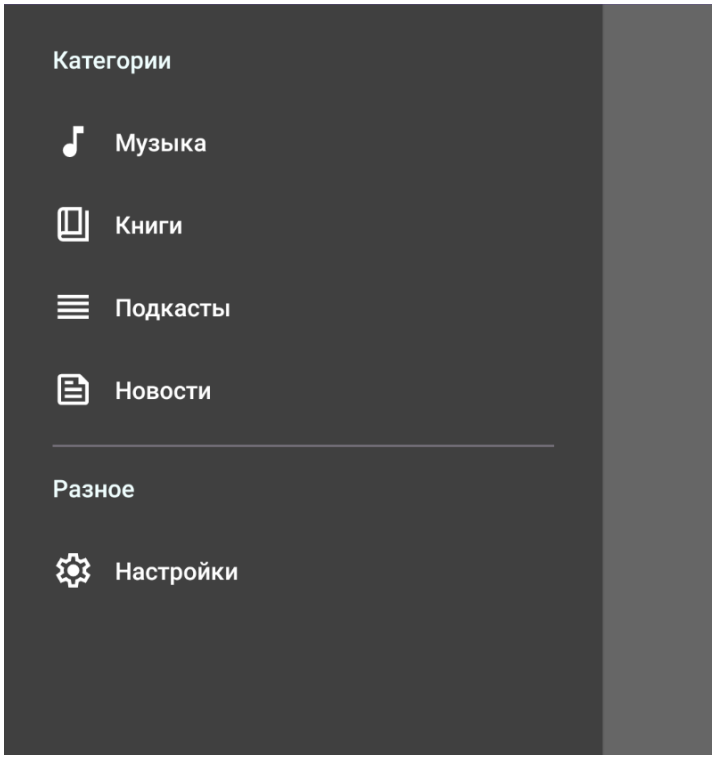
В **нижней** части экрана обычно располагается панель навигации `BottomNavigationView`, которая обеспечивает навигацию по ключевым разделам приложения:



Google предлагает ряд рекомендаций, следование которым сделает панель удобной и привычной для пользователя:

- она должна содержать от 3 до 5 элементов;
- её следует использовать для разделов верхнего уровня, которые должны быть доступны из любой точки приложения – кнопки на панели навигации не должны меняться в зависимости от раздела;
- не следует использовать кнопки на панели для небольших задач, например, для просмотра одного сообщения или письма;
- не нужно помещать на панель кнопку для открытия настроек или информации о программе;
- значки должны быть одного контрастного цвета, не нужно их раскрашивать.

В **левой** части экрана может располагаться боковая панель `Navigation Drawer`, которая открывается свайпом от левого края экрана или нажатием кнопки «бургер» на верхней панели:



Рекомендуется использовать боковую панель:

- если в приложении 5 и более элементов верхнего уровня;
- если в приложении 2 или более уровня иерархии (например, в почтовом приложении могут быть папки, и одновременно с ними – виртуальные метки);
- для замены навигационной панели на больших экранах.

Наконец, в **верхней** части экрана могут располагаться вкладки – они служат для упорядочивания контента, находящегося на одном уровне иерархии:

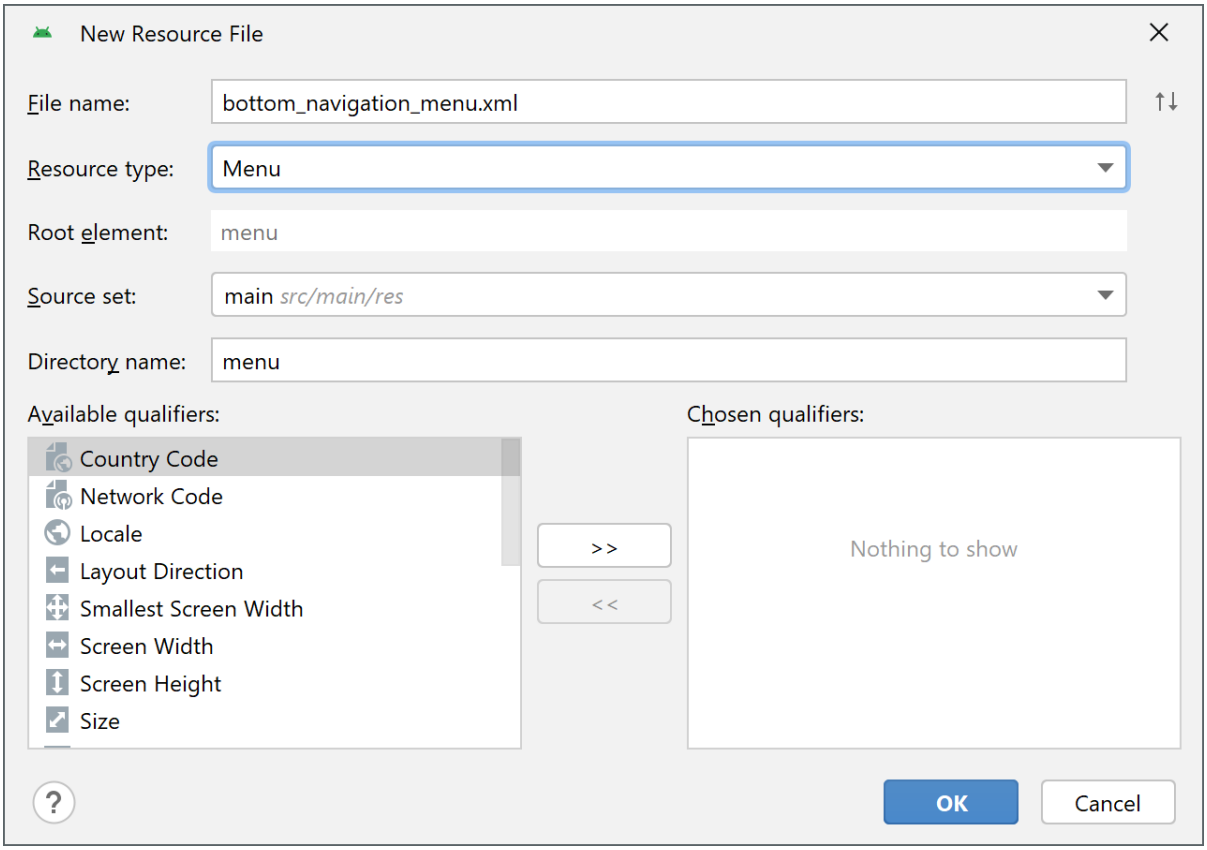
В данной теме будут изучены навигационная панель и панель вкладок.

Навигационная панель BottomNavigationView

Размещать элемент `BottomNavigationView` нужно в нижней части экрана, это можно сделать с помощью разных контейнеров, вот пример для `ConstraintsLayout`:

```
<com.google.android.material.bottomnavigation.BottomNavigationView
    android:id="@+id/bottomNav"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    app:menu="@menu/bottom_navigation_menu"/>
```

Атрибут `menu` указывает на XML-файл со списком кнопок, которые будут отображены на панели. Чтобы создать такой элемент, нужно щёлкнуть правой кнопкой на папке *res* и выбрать *New → Android Resource File*, и в пункте *Resource type* выбрать значение *Menu*:



Среда создаст папку *menu* (если её ещё не было) и разместит там XML-файл. В него нужно добавить пункты `item`, каждый из которых соответствует кнопке на панели навигации:

```
<menu ...>
    <item
        android:id="@+id/menu_music"
        android:icon="@drawable/music"
        android:title="@string/music"/>
    <item
        android:id="@+id/menu_books"
        android:icon="@drawable/book"
        android:title="@string/books"/>
    <item
        android:id="@+id/menu_news"
        android:icon="@drawable/news"
        android:title="@string/news"/>
</menu>
```

Каждый элемент `item` может содержать атрибуты:

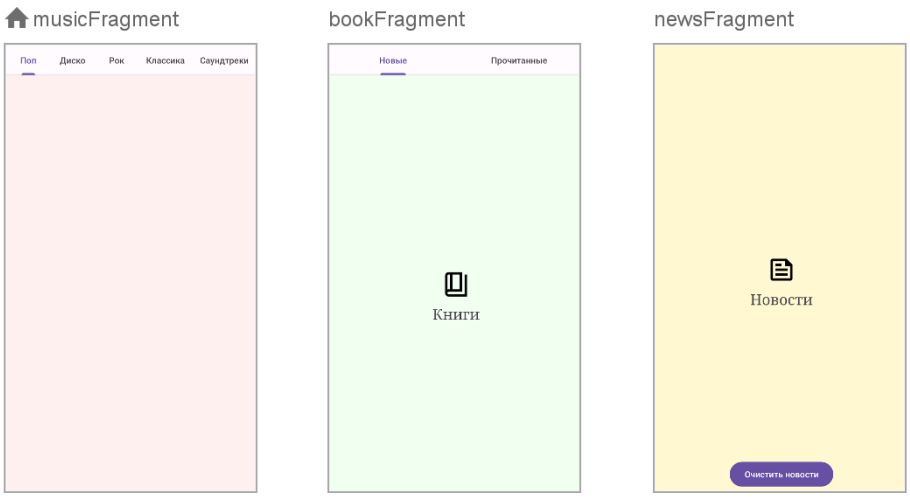
- `id` – с его помощью в коде программы можно будет определить какая кнопка была нажата;
- `icon` – значок, который будет отображаться на кнопке;
- `title` – краткая текстовая подпись к значку;
- `enabled` – доступна ли кнопка для нажатия пользователем.

В коде программы нужно подключить слушатель нажатия кнопки, чтобы реагировать на действия пользователя:

```
val navBar = findViewById<BottomNavigationView>(R.id.bottomNav)
navBar.setOnItemSelectedListener {
    when (it.itemId) {
```

```
        R.id.menu_music -> {
            navCtl.navigate(R.id.musicFragment)
            true
        }
        R.id.menu_books -> {
            navCtl.navigate(R.id.bookFragment)
            true
        }
        R.id.menu_news -> {
            navCtl.navigate(R.id.newsFragment)
            true
        }
        else -> false
    }
}
```

В данном примере при нажатии кнопки происходит навигация на соответствующий фрагмент, как это было описано в одной из предыдущих тем. Поскольку при использовании навигационной панели пользователь может переходить между фрагментами в произвольном порядке, достаточно просто добавить фрагменты в навигационный граф без создания связей между ними:



Вместо фрагментов можно использовать другие методы отображения контента.

Если при нажатии кнопки программа осуществляет переход, то слушатель должен вернуть значение `true`, в противном случае `false`.

Если пользователь нажимает на кнопку, которая уже выбрана, то вызывается событие `setOnItemReselectedListener`, при необходимости можно обработать и такую ситуацию.

На кнопке может отображаться специальный маркер – бейдж, который используется для привлечения внимания пользователя при наличии нового контента в соответствующем разделе:

Чтобы добавить такой бейдж к кнопке, нужно вызвать метод `getOrCreateBadge` у навигационной панели – он возвращает объект бейджа для указанной кнопки. Свойство

`number` содержит число, которое появляется в бейдже, а свойство `isVisible` определяет будет ли бейдж видим:

```
val badge = navBar.getOrCreateBadge(R.id.menu_news)
badge.number++
badge.isVisible = true
```

Панель вкладок TabLayout

Верхние вкладки формируются элементом `TabLayout`, и как правило размещаются в верхней части экрана:

```
<com.google.android.material.tabs.TabLayout
    android:id="@+id/tabs"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:tabMode="scrollable">

    <com.google.android.material.tabs.TabItem
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/tab_music_pop"/>
    ...
```

Пункты панели вкладок формируются элементами `TabItem`, которым можно задать текст и/или значок.

Панель вкладок может быть фиксированной (если вкладок немного и они полностью помещаются на экран) или прокручиваемой (вкладки могут уходить за правый край экрана) – это поведение определяется свойством `tabMode`, которое может принимать значение `fixed` или `scrollable`.

Для реакции на нажатие на вкладку может использоваться слушатель, который обрабатывает сразу три события:

```
tabs.addOnTabSelectedListener(object : TabLayout.OnTabSelectedListener {

    override fun onTabSelected(tab: TabLayout.Tab?) {
        // Вкладка выбрана
    }

    override fun onTabReselected(tab: TabLayout.Tab?) {
        // Вкладка выбрана повторно
    }

    override fun onTabUnselected(tab: TabLayout.Tab?) {
        // Вкладка перестала быть выбранной
    }
})
```

Кажется логичным определить что за вкладка была выбрана с помощью параметра `tab`, ведь у него есть `id`, однако идентификатор всегда будет -1. Вместо этого используется свойство `position`, которое содержит порядковый номер вкладки начиная с 0:

```
override fun onTabSelected(tab: TabLayout.Tab?) {
    if ((tab?.position ?: -1) == 1)
        ...
}
```

Однако более удобный способ отображения контента – это использование элемента `ViewPager2`, который автоматически отображает фрагменты и позволяет переключаться между ними с помощью свайпа. Разместить его можно следующим образом:

```
<androidx.viewpager2.widget.ViewPager2
    android:id="@+id/pager"
    .../>
```

Как и списки, ViewPager2 требует адаптера. К счастью адаптер весьма прост и содержит всего две функции:

```
class MusicViewPagerAdapter(val fragments: List<Fragment>, fragment: Fragment) :
    FragmentStateAdapter(fragment) {

    override fun getItemCount(): Int {
        return fragments.size
    }

    override fun createFragment(position: Int): Fragment {
        return fragments[position]
    }
}
```

Адаптер должен переопределять функции:

- `getItemCount` возвращает количество фрагментов, которые `ViewPager2` может отобразить;
- `createFragment` возвращает сам фрагмент с указанным индексом.

Хотя вторая функция называется `createFragment`, создавать фрагмент вовсе не обязательно, можно вернуть уже существующий фрагмент. В примере выше адаптер получает в качестве параметра такой список фрагментов `fragments`, и потом просто возвращает очередной фрагмент с нужным индексом. Создать такой адаптер можно примерно так:

```
val adapter = MusicViewPagerAdapter(listOf(
        MusicInsideFragment(resources.getString(R.string.tab_music_pop)),
        MusicInsideFragment(resources.getString(R.string.tab_music_disco)),
        MusicInsideFragment(resources.getString(R.string.tab_music_rock)),
        MusicInsideFragment(resources.getString(R.string.tab_music_classic)),
        MusicInsideFragment(resources.getString(R.string.tab_music_score)),
        MusicInsideFragment(resources.getString(R.string.tab_music_instrument))
    ), this)
pager.adapter = adapter
```

Здесь в адаптер передаётся список фрагментов, каждый из которых просто копия фрагмента `MusicInsideFragment`, и получает разные строковые параметры.

Помимо адаптера нужно создать ещё объект типа `TabLayoutMediator`, который будет служить передаточным механизмом между вкладками `TabLayout` и слайдером фрагментов `ViewPager2`:

```
TabLayoutMediator(tabs, pages) { tab, position ->
    when (position) {
        0 -> tab.text = resources.getString(R.string.tab_music_pop)
        1 -> tab.text = resources.getString(R.string.tab_music_disco)
        2 -> tab.text = resources.getString(R.string.tab_music_rock)
        3 -> tab.text = resources.getString(R.string.tab_music_classic)
        4 -> tab.text = resources.getString(R.string.tab_music_score)
        5 -> tab.text = resources.getString(R.string.tab_music_instrument)
    }
}.attach()
```

При переключении вкладки он будет автоматически перемещать слайдер фрагментов на нужный фрагмент, а если пользователь свайпит фрагменты – будет синхронизировать выделенную вкладку.

При использовании `TabLayoutMediator` требуется динамически устанавливать название (и, по желанию) значок каждой вкладки.

Как и кнопки навигационной панели, вкладки тоже могут иметь бейджи:

```
val badge = tab.getOrCreateBadge()
badge.number = 1
```

Задание

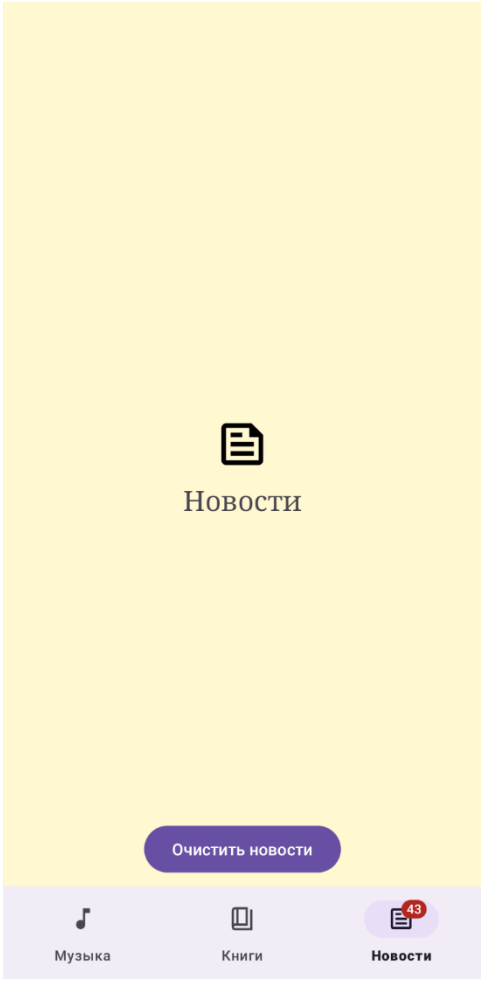
Напишите приложение-суперапп, которое, как и многие другие супераппы, сочетает в себе малосочетаемое: музыку, книги и новости.

В нижней части приложения должен располагаться элемент `BottomNavigationView` с кнопками разделов: *Музыка*, *Книги*, *Новости*. При нажатии одной из кнопок происходит переход в соответствующий раздел. Можно реализовать разделы на фрагментах с навигационным графом, или другим способом.

В разделе *Музыка* в верхней части должен располагаться элемент `TabLayout` со вкладками, соответствующим разным жанрам музыки. Вкладок должно быть достаточно много, чтобы они не помещались на экран, элемент `TabLayout` должен работать в режиме `scrollable`. Содержимое экрана должно располагаться в слайдере `ViewPager2`, каждая вкладка связана с соответствующим фрагментом (можно использовать один фрагмент и передавать ему информацию о жанре на выбранной вкладке):

В разделе *Книги* в верхней части должен располагаться элемент `TabLayout` со вкладками *Новое* и *Прочитанное*, элемент `TabLayout` должен работать в режиме `fixed`. Содержимое экрана должно располагаться во фрагменте, в котором каким-либо образом отображается информация о выбранной вкладке:

При запуске программы каждые несколько секунд должно увеличиваться количество непрочитанных новостей в бейдже на кнопке *Новости*:



Если зайти в раздел *Новости*, то количество новостей прекращает увеличиваться, а если уйти из раздела – то снова начинает расти. В нижней части раздела располагается кнопка *Очистить новости*, которая сбрасывает счётчик в 0, и скрывает его. Когда количество новостей снова начинает увеличиваться – счётчик должен снова появиться.

Это пример программы, можно выбрать какую-то свою тематику приложения, но описанный функционал должен быть реализован и там.

Таймер обратного отсчёта

Для того чтобы выполнять какое-то действие через регулярные промежутки времени можно использовать таймер обратного отсчёта `CountDownTimer`. Создаётся он немного необычно – как синглтон, потому что класс `CountDownTimer` является абстрактным, и для его создания необходимо определить недостающие методы:

```
val timer = object : CountDownTimer(60_000, 2_000) {
    override fun onTick(millisUntilFinished: Long) {
        // Вызывается каждые 2 секунды
    }
    override fun onFinish() {
        // Вызывается через 60 секунд
    }
}
timer.start()
```

В этом примере создаётся таймер, который в течение 60 секунд будет каждые 2 секунды вызывать метод `onTick`, а в конце вызовет метод `onFinish` и прекратит свою работу. Эти методы вызываются в основном потоке программы, поэтому они имеют доступ к элементам

управления, но также это значит что они будут замедлять выполнение основного потока, поэтому не рекомендуется выполнять в них какие-либо тяжёлые вычисления, код должен быть максимально простым и эффективным.