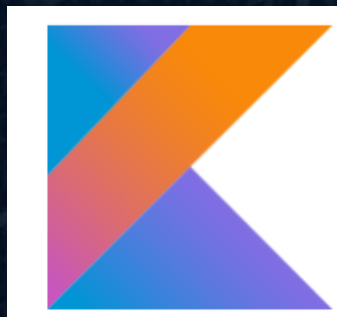




Икономически университет -
Варна

Катедра „Информатика“

Нейтив мобилни приложения



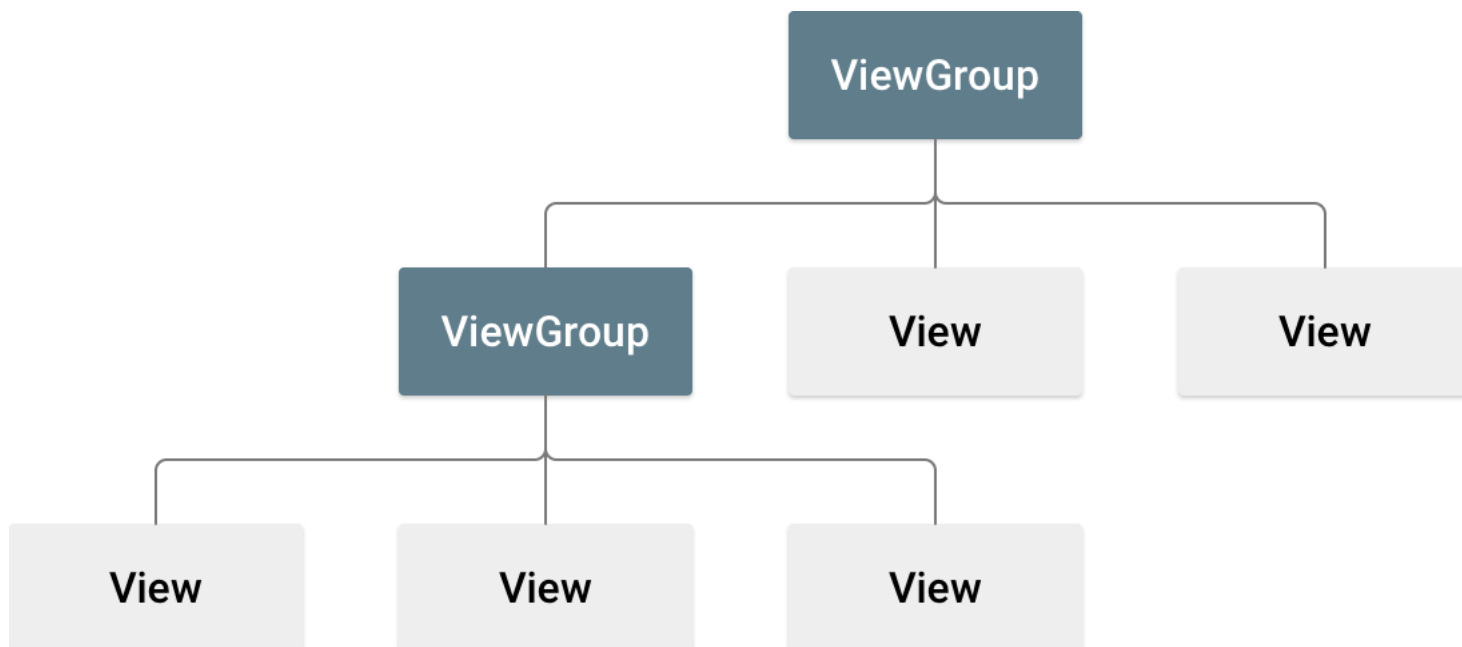
Потребителски
интерфейс

Изглед (Layout)

- Определя визуалната структура на потребителския интерфейс (напр. activity)
- Може да се дефинира по два начина:
 - Като елементи в XML
 - Или чрез създаване на елементи по време на изпълнение
- Android ви дава възможност да използвате един от двата или и двата начина за управление на потребителския интерфейс

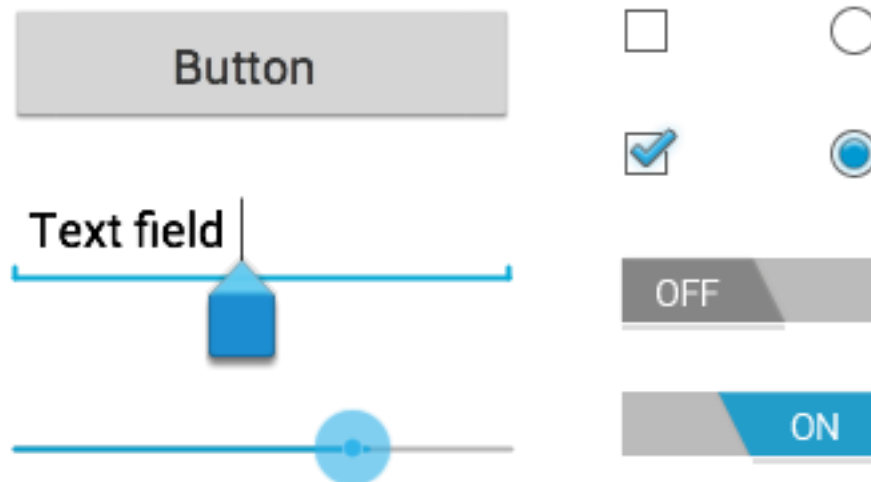
Изглед (Layout)

- Всички елементи на изгледа се създават като се използва йерархия от контроли (View) и групиращи елементи (ViewGroup)



Контроли (View)

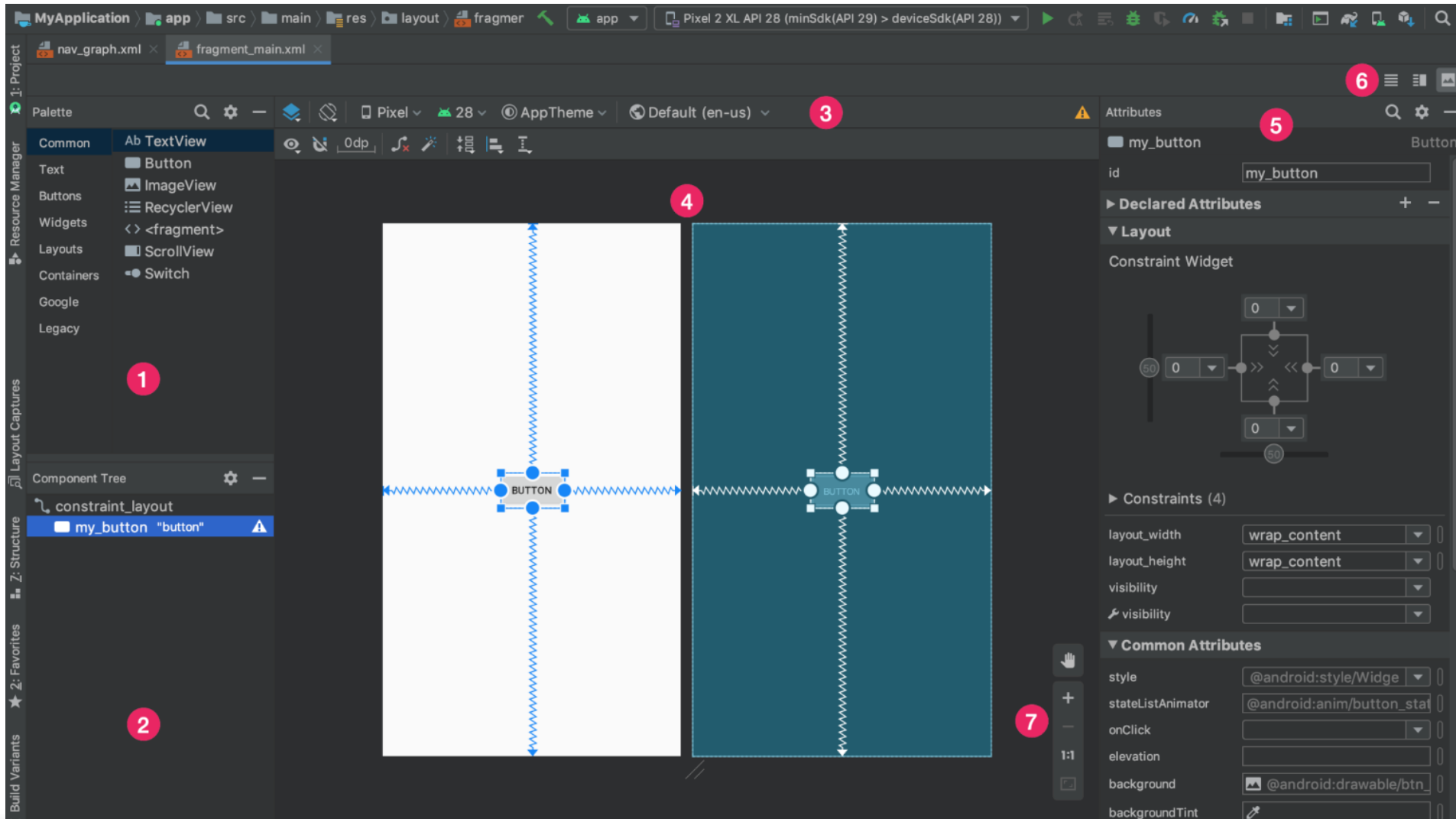
- Интерактивни компоненти на потребителския интерфейс на приложението
- Примери – TextView, Button, ImageView, ScrollView, Switch и др.



Linear Layout

- Групиращ контрол, който подрежда съдържащите се контроли в една посока
 - вертикално или хоризонтално
 - задава се с `android:orientation`
- Съдържащите се контроли се подреждат един след друг
- Взема предвид полетата (`margins`) между съдържащите се контроли, а също и подравняването им (`gravity`)

Layout Editor



Layout Editor

1. Контроли и групиращи елементи
2. Йерархия на компоненти в изгледа
3. Конфигуриране на изгледа в Design Editor
4. Редактиране на изгледа
5. Атрибутите на текущо избрания контрол или групиращ елемент
6. Режими на показване - Code, Design, Split
7. Оразмеряване на Design Editor

Зареждане на XML ресурсите

- За зареждане на layout ресурсите от кода на приложението е необходимо да се използва методът `OnCreate()` и `setContentView(R.layout.изглед)`

```
override fun onCreate (savedInstanceState:  
Bundle?) {
```

```
    super.onCreate(savedInstanceState)
```

```
    setContentView(R.layout.activity_main)
```

```
}
```


Атрибути

- Всеки изглед, както и контрол (View) поддържа свои собствени XML атрибути
- Някои от тях естествено са специфични само за даден контрол, например:
 - TextView поддържа атрибут `textSize`
- Други пък са общи за всички контроли:
 - `android:id="@+id/button"`
 - Връзка между Kotlin код и контрол по `id`
 - `val my_button: Button = findViewById(R.id.button);`

Атрибути

- XML layout атрибути - layout_xxxx
- Всички контроли трябва да дефинират задължително ширина и дължина:
 - layout_width
 - layout_height
- При това много често се използва една от следните две константи:
 - wrap_content – съдържанието на контрола
 - match_parent – съдържанието на родителя

Позиция в Layout

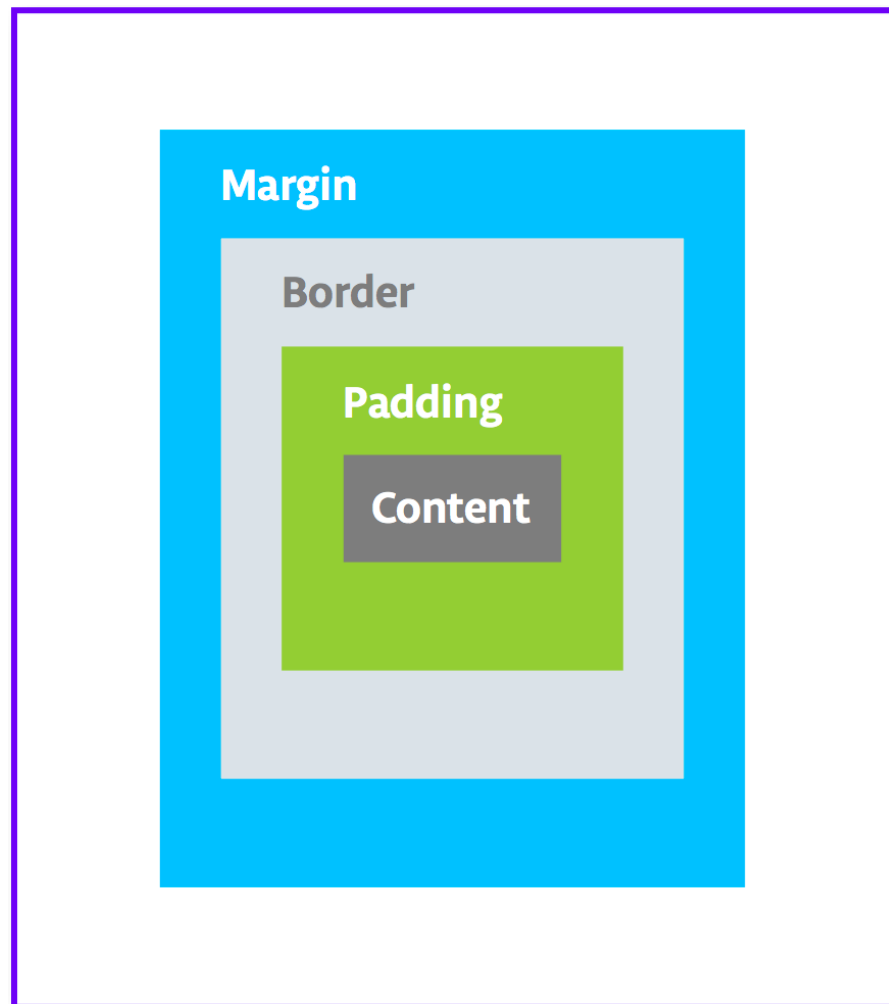
- Всеки контрол има местоположение, което се определя от двойка координати (ляво и горе) и две размерности – width и height
- Единицата за местоположение и размерност е пиксел
- Методите `getLeft()` и `getTop()` връщат позицията на дадения контрол съотнесена към тази на неговия родител.

Размер в Layout

- Контролите имат 2 двойки стойности за ширина и дължина
 - measured width и measured height
 - Определят колко голям иска да бъде контролът
 - Достъп – `getMeasuredWidth()`, `getMeasuredHeight()`
 - width and height
 - Определят действителния размер на контрола
 - Достъп - `getWidth()` и `getHeight()`
- При определяне на размера е от значение отстоянието - `setPadding(int, int, int, int)`

Padding vs Margin атрибути


- `android:padding`
 - разстоянието между рамката и съдържанието
- `android:layout_margin`
 - разстоянието между рамките на контрола и неговия родител
- `right/left` vs `start/end`



Атрибут Visibility

- Показва или скрива контроли в изгледа
- Възможни стойности:
 - visible – показва контрола
 - invisible – скрива контрола, като заема място
 - gone - скрива контрола, като не заема място
- Стойността на атрибута може да се променя и през програмния код, чрез View
 - Пример: `editText.visibility = View.GONE;`

Design-time атрибути

- Тези атрибути се използват и прилагат само по време на дизайн, но не и по време на изпълнение (тогава се игнорират)
- В XML кода се предхождат от `tools namespace`.
- В дизайнера се обозначават с 
- Примери:
 - `tools:layout_editor_absoluteY`
 - `tools:text`

TextView

- Контрол за визуализиране на текст.
- Атрибути:
 - text
 - textAppearance
 - textSize – sp(scalable pixels)
 - textColor
 - textAlignment (за по-стари API - gravity)
 - fontFamily
 - lineSpacingMultiplier
 - style

Бутон

- Използва се от потребителя за реализиране на определено действие
- Бутонът може да бъде визуализиран с текст, икона или и двете.



- Пример:

```
<Button
```

```
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
```

```
    android:text="@string/button_text" />
```

Бутон

- ImageButton

```
<ImageButton
```

```
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
```

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

```
... />
```

- Бутон с текст и икона:

```
    android:text="@string/button_text"
```

```
    android:drawableLeft="@drawable/button_icon"
```

Бутон – click събитие

- В XML чрез атрибута android:onClick
 - android:onClick="clickHandlerFunction"
 - clickHandlerFunction е функция в кода
- В програмния код - setOnClickListener
 - android:id="@+id/button"
 - val my_button: Button =
findViewById(R.id.button);
 - my_button.setOnClickListener {
 // действие
}

Бутон - стилове

- Използване на потребителски стил

```
<Button ...  
    style="@style/button_style"  
    android:text="@string/button_text" />
```
- Задаване на нов стил в ресурсите-style.xml

```
<style name="button_style">  
    <item name="android:textColor">#00FF00</item>  
</style>
```


ImageView

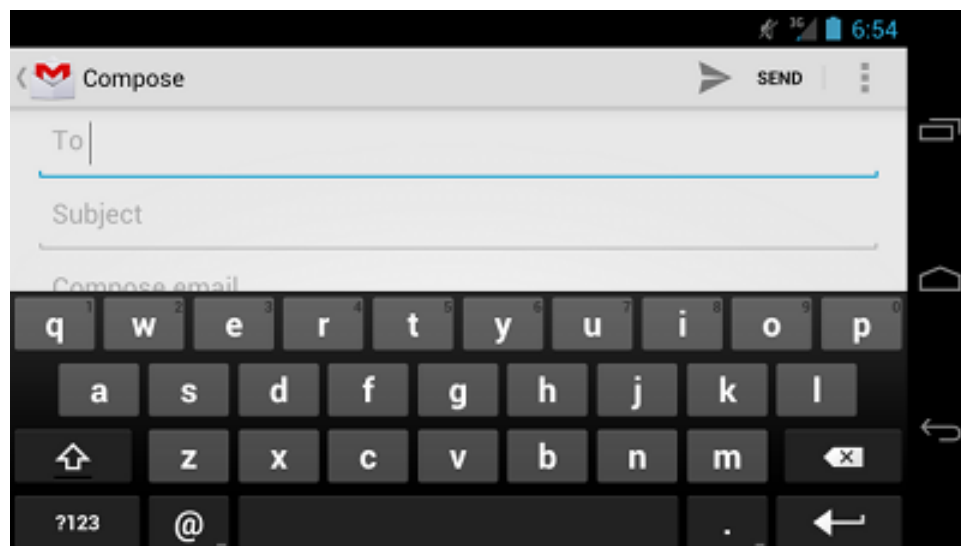
- Използва се за добавяне на drawable или bitmap ресурси, чрез атрибута android:src
 - android:src="@drawable/име на картинката"
- Може да се добавя динамично чрез метод
 - setImageResource()
- За да покажем картинка само в дизайн режим се използва tools:src
- За обратна съвместимост - app:srcCompat
 - vectorDrawables.useSupportLibrary = true

ScrollView

- Групиращ контрол, който се използва за визуализация на дълъг текст или набор от изображения, като позволява превъртане
- Той може да има само 1 контрол в себе си. За визуализация на повече контроли следва да се използва друг групиращ контрол (например `LinearLayout`) и в него да се разположат другите контроли

Текстови полета

- Дават възможност за въвеждане на текст
 - едноредови и многоредови
- Автоматично визуализират клавиатурата
- Атрибут – Hint



Тип на клавиатурата

- Използва се атрибута `android:inputType`
 - `text`
 - `textEmailAddress` – добавя символът `@`
 - `textUri` - добавя символът `/`
 - `Number`
 - `phone`

`<EditText`

`android:inputType="textEmailAddress />"`

Атрибут android:inputType

- Някои възможни стойности:

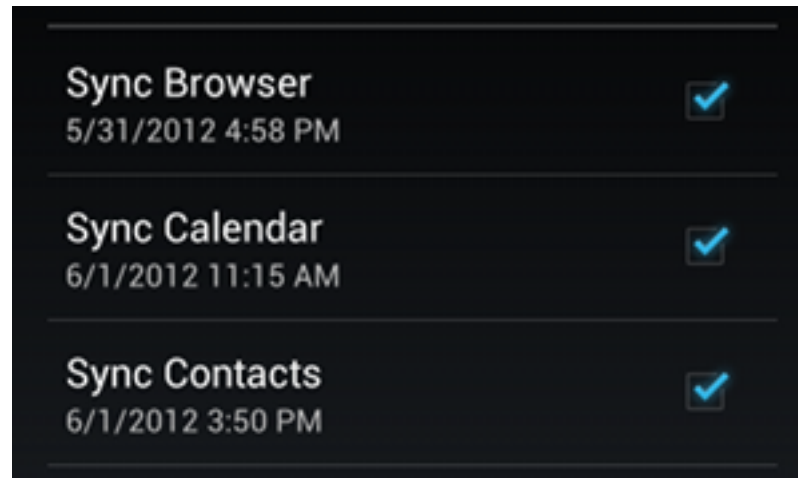
- textCapSentences
- textCapWords
- textAutoCorrect
- textPassword
- textMultiLine

<EditText ...

```
    android:inputType="textPostalAddress|  
        textCapWords|textNoSuggestions" />
```

Checkbox

- Обикновено тези контроли се нареждат във вертикален списък



- Всеки контрол се управлява отделно като за всеки трябва да бъде дефинирано събитие click

Radio бутони

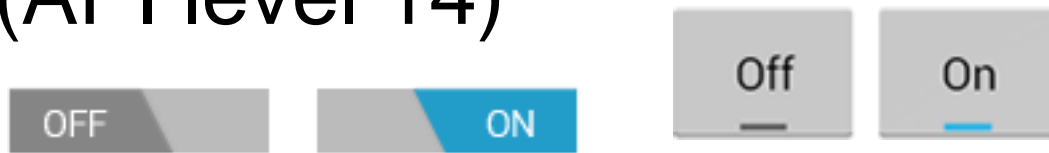
- Групират се с помощта на RadioGroup

```
<RadioGroup ...  
    android:orientation="vertical">  
    <RadioButton ...  
        android:text="@string/pirates" />  
    <RadioButton ... />  
</RadioGroup>
```

- Ако не е необходимо стойностите да се показват една до друга за визуализирането им се препоръчва използването на spinner

Toggle бутон

- Позволява на потребителя да промени настройка като избере между 2 състояния
- Switch контрол се използва от Android 4.0 (API level 14)



- За ръчна промяна:
 - Свойството `isChecked`
 - Методът `toggle()`

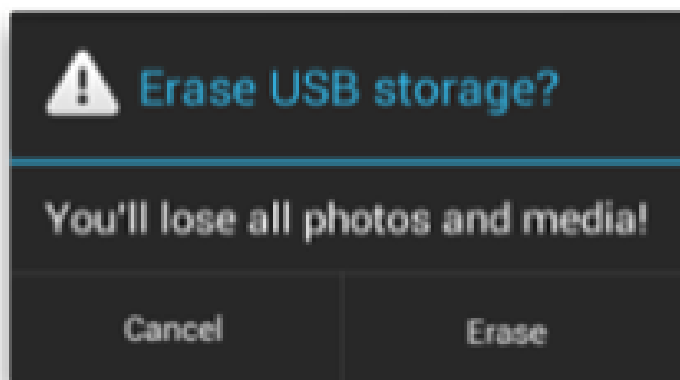
Spinner

- Бърз начин за избор на една стойност от набор от стойности
- Опциите се зареждат с Adapter – ArrayAdapter, CursorAdapter

```
val spinner: Spinner = findViewById(R.id.spinner)
var spinner_adapter = ArrayAdapter.createFromResource(this,
    R.array.planets_array, android.R.layout.simple_spinner_item)
spinner_adapter.setDropDownViewResource(android.R.layout.simple
    _spinner_dropdown_item)
spinner.adapter = spinner_adapter;
```

Диалогов прозорец

- Малък модален прозорец, който подтиква потребителя да вземе решение или да въведе допълнителна информация преди да може да продължи.



- Използват се следните класове:
 - AlertDialog, DatePickerDialog, TimePickerDialog

Диалогов прозорец AlertDialog

- Може да показва заглавие, до три бутона, списък с избираеми елементи или персонализирано оформление
- За създаване се използва обект от клас `AlertDialog.Builder` и метод `create()`

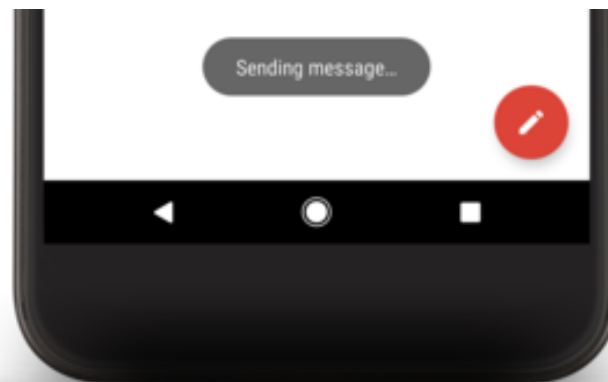
```
val builder = AlertDialog.Builder(this);  
val dialog = builder.create();
```
- За оформление се ползват методите `setTitle()`, `setMessage()`, `setItems()` и др.

Pickers

- Предоставя контроли за избиране на всяка част от :
 - Time - hour, minute, AM/PM
 - Date - month, day, year
- Контролира потребителския избор:
 - Избор на валидна дата и час
 - Правилно форматиране
 - Настройка на системната променлива locale

Toast

- Предоставя обратна връзка за операция в малък изскачащ прозорец.
- Запълва само пространството, което е необходимо за съобщението, а текущата активност остава видима и интерактивна.
- Автоматично изчезва след показване.



Toast

- За създаване се използва методът `makeText()`, който има 3 параметъра:
 - Context
 - Текстът, който следва да се покаже
 - Продължителността на показване
- Пример: `Toast.makeText(this, "Hello!", Toast.LENGTH_LONG).show()` // Методът `show()` се използва за визуализация.
- Методът `setGravity()` се използва за задаване на позицията му

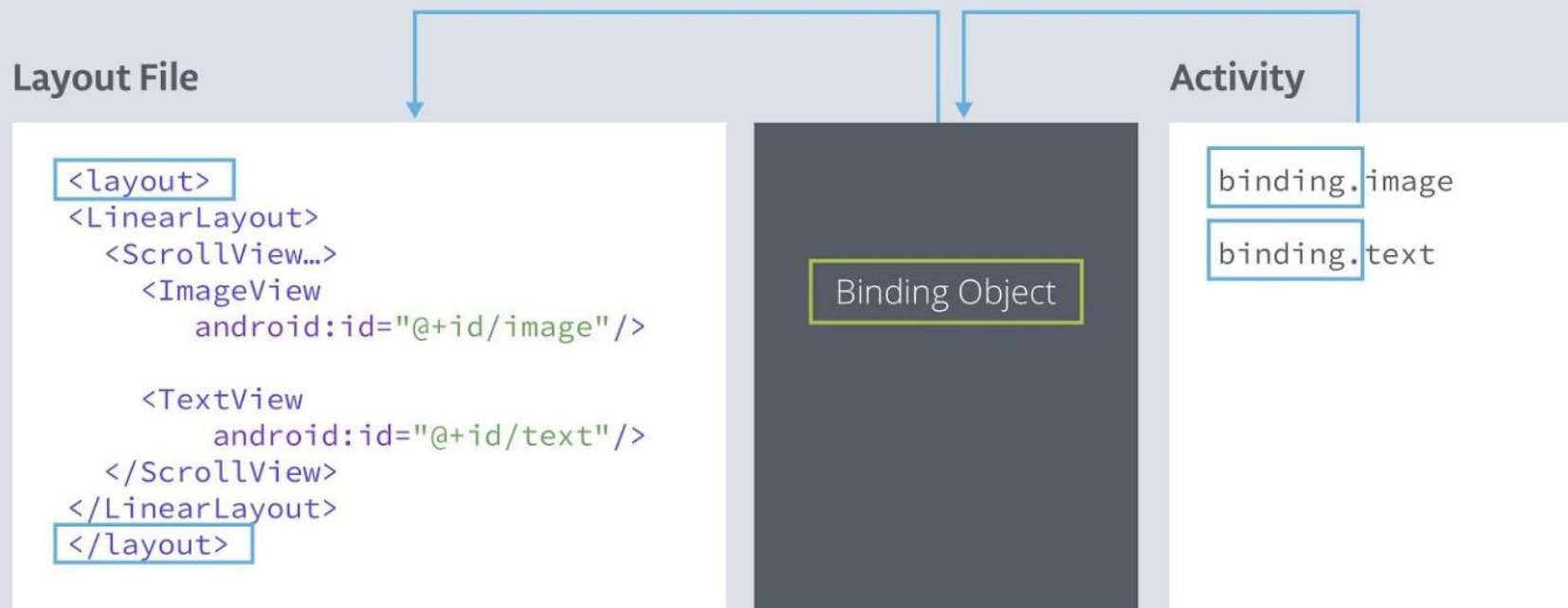
Data Binding

- Тази техника се използва, за да намали времето за търсене (`findViewById()`) на контроли при по-сложни йерархии
- Предимства:
 - по-кратък и по-лесен за четене/поддръжка код
 - данните и контролите са ясно разделени
 - Android проследява йерархията на контролите само веднъж – при зареждане
 - компилаторът проверява типовете данни

Data Binding – заместване на findViewById()

- Активиране в android секцията на модула
build.gradle: buildFeatures {dataBinding true}
- Използване на <layout> в XML .
- Дефиниране на binding променлива: private
lateinit var binding: ActivityMainBinding
- Замяна на setContentView с : binding =
DataBindingUtil.setContentView(this,
R.layout.activity_main)
- Замяна на findViewById() с binding.*

Data Binding – заместване на findViewById()



Data Binding – връзка между контроли и data класове

- Да се създаде data class.
- Да се добави `<data>` блок в `<layout>`.
- Да се дефинира `<variable>` с
 - `<variable name="..." type="..." />`
- В MainActivity да се създаде променлива от създадения клас, която да се свърже с променливата в XML
- На контрол да се зададе променливата дефинирана в `<data>` блока с `"@={ ... }"`

Навигация

- За да се използва библиотеката за навигация в Android е необходимо:
 - В build.gradle на ниво проект в ext да се добави navigationVersion = "2.3.5", като последната версия може да се намери тук
 - В build.gradle на ниво модул в dependencies да се добавят:
 - navigation-fragment-ktx
 - navigation-ui-ktx

Навигационен граф

- Навигационният граф определя възможните пътища от една навигационна точка до друга.
- Навигационните точки могат да бъдат: фрагменти, activities или други компоненти
- Създава се в ресурсите в папка navigation
- Освен навигационни точки в навигационния граф могат да се добавят и преходи(actions), които свързват точките

Navigation host фрагмент

- Navigation host фрагмент служи като контейнер за фрагментите в навигационния граф и често се казва NavHostFragment.
- При движение на потребителя между различни навигационни точки, дефинирани в навигационния граф, navigation host фрагментът разменя фрагментите и създава и управлява стека на фрагментите.
- Дефинира се като фрагмент с name = androidx.navigation.fragment.NavHostFragment

Реализиране на преход

- За да се определи към коя навигационна точка да се премине се използва метода `findNavController().navigate()` на контрола, който е бил натиснат, като на този метод се предава `id` на прехода, дефиниран в навигационния граф. Например:

```
view.findNavController().navigate(R.id.action_titleFragment_to_gameFragment)
```


Back бутон

- Системен бутон, който се използва за навигация в хронологичен ред на поява на екраните.
- За разлика от Up бутона, Back бутонът може да излезе от приложението или да пренасочи към друго
- В някои случаи има поведение, което не е пряко свързано с навигацията.
- Атрибути `popUpTo` и `popUpToInclusive`

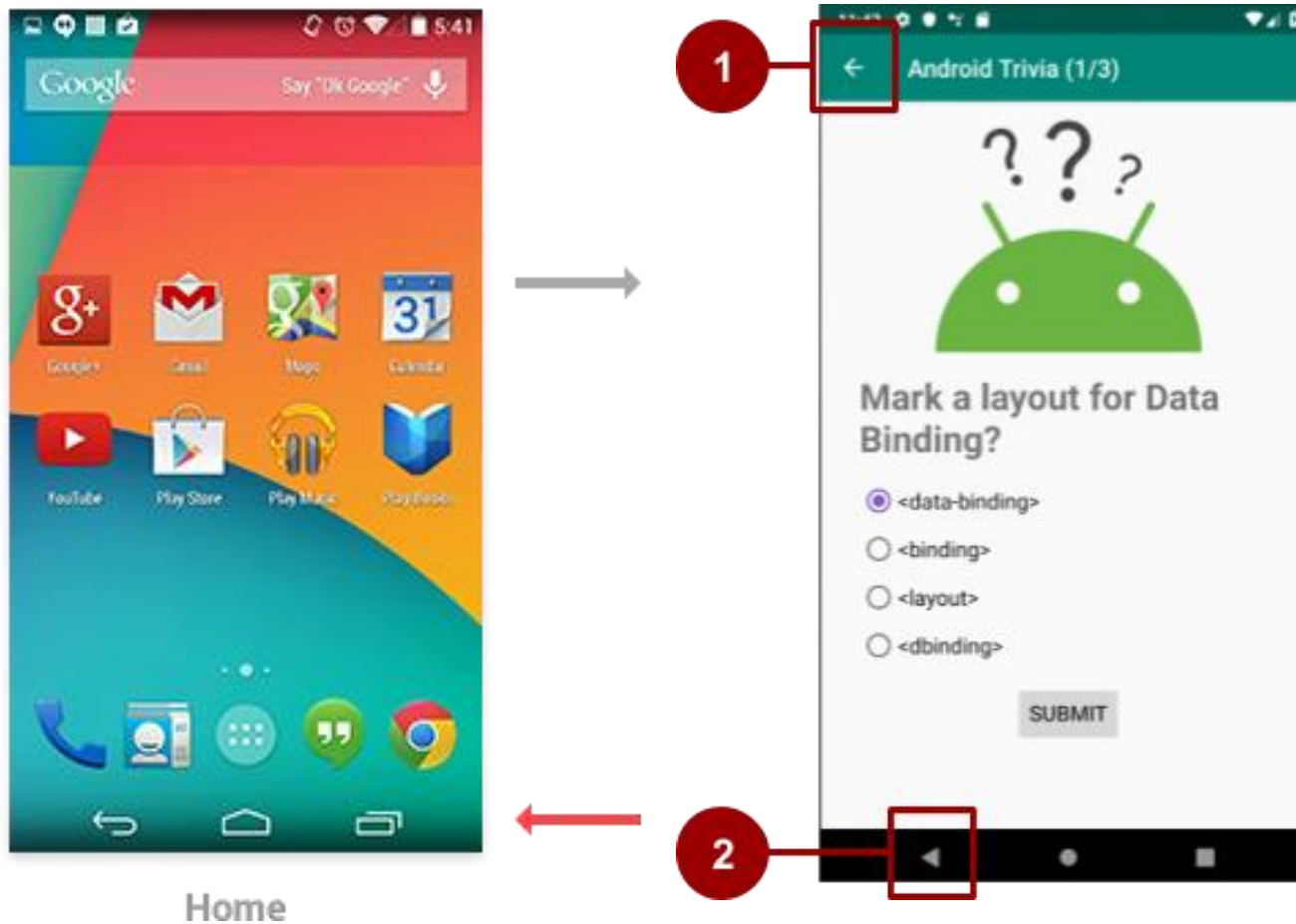


Back button


Up бутон

- Част е от app bar-а (action bar) и се състои от стрелка наляво.
- Използва се за навигация в приложението в зависимост от йерархичната организация на екраните
- Отсъства на началния екран на прилож.
- NavigationUI библиотеката на навигационния контролер дава възможност за визуализиране на up бутон и за реализиране на неговото поведение


Up vs. Back

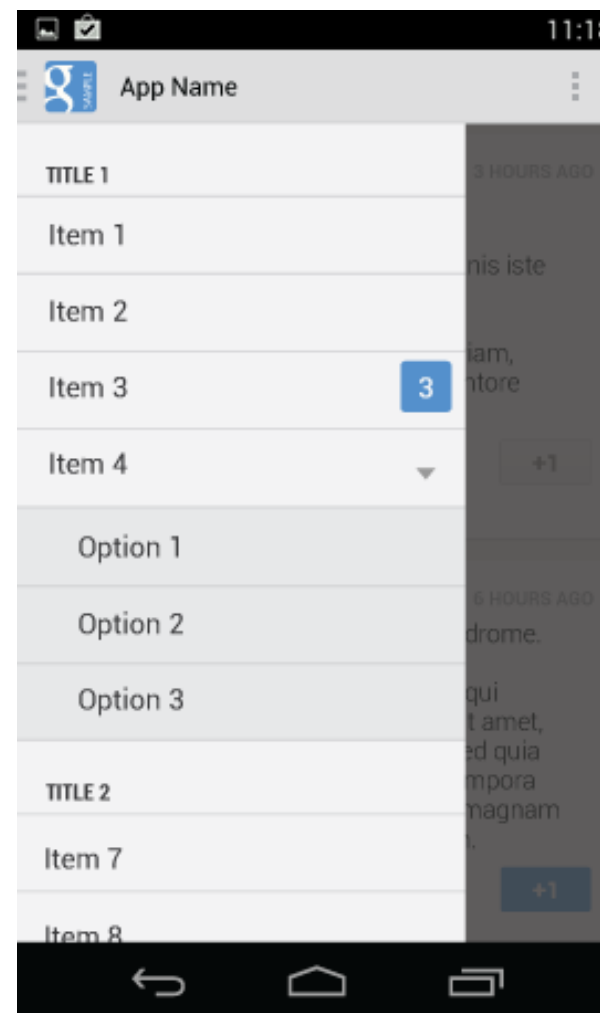


Options menu

- Достъп се получава от app bar 
- Да се създаде файл в ресурсите от тип Menu, в който да се добавят отделни елементи на менюто, чиито ID-та да съвпадат с тези на фрагментите, които ще зареждат
- `setHasOptionsMenu(true)` – активира менюто
- Предефиниране на `onOptionsItemSelected()` – за връзка с дизайна на менюто
- Предефиниране на `onOptionsItemSelected()` – за избор на елемент от менюто

Drawer

- Панел, който се показва в лявата част на екрана и показва основните възможности за навигация на приложението
- Може да бъде показан:
 - чрез приплъзване от някой от краищата на екрана (ляв)
 - чрез избор на 



Drawer

- В build.gradle на ниво модул в dependencies:
 - implementation "com.google.android.material:material:\$supportlibVersion"
- Да се създаде файл в ресурсите от тип Menu, в който да се добавят отделни елементи на менюто, чиито ID-та да съвпадат с тези на фрагментите, които ще зареждат
- Добавяне към даден фрагмент/activity:
 - <androidx.drawerlayout.widget.DrawerLayout>
 - <com.google.android.material.navigation.NavigationView>
- `NavigationUI.setupWithNavController()`
- `NavigationUI.setupActionBarWithNavController()`

Навигация с табове

- В основния екран на приложението се използва `BottomNavigationView`, който задава визуалната структура на табовете
- Функционалността на всеки таб се реализира в отделен фрагмент
- В `MainActivity` се използва методът `setOnNavigationItemSelectedListener`, за да се определи какво се случва при избор на някой от табовете.

Навигация между приложения

- Android приложенията могат да се активират взаимно и така потребителят може да навигира между тях
- Intent – механизъм на дадено приложение да заяви, че иска помощ от друго за реализиране на дадено действие.
- Implicit intent - текущото приложение декларира действие, което трябва да бъде реализирано от друго приложение (например камера или email клиент)

Навигация между приложения

- Ако няколко приложения могат да обработят декларираното в `implicit intent` действие, на потребителя се показва диалог за избор
- Всеки `implicit intent` има `ACTION`, което описва типа действие, което трябва да се реализира. Тези действия се дефинират в `Intent` класа. Примери: `ACTION_VIEW`, `ACTION_EDIT`, `ACTION_DIAL` и др.