3. User Interface(s) in Android.

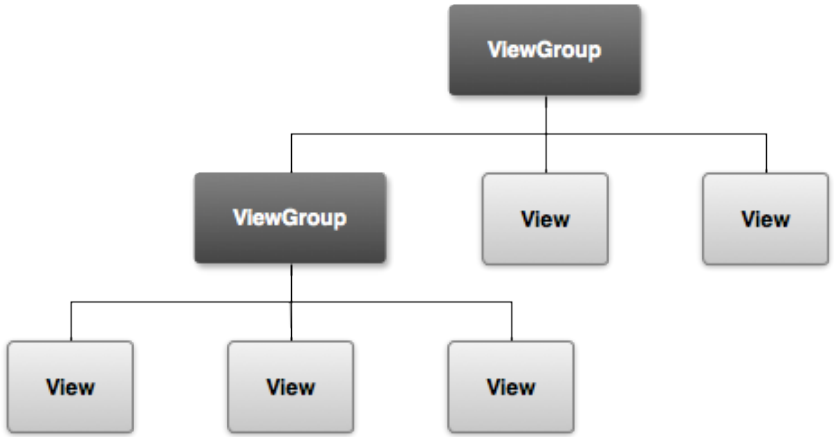
Вместо съдържание:

- Запознаване с основните UI building-blocks в Android (чрез Android Studio).
- Какво можем да правим с тях?
- А защо не да си направим и custom ;)

Бърз поглед на Android UI.

- Потребителският интерфейс в Android се изгражда от два основни типа обекти View и ViewGroup;
- View обектите (по точно widgets наследници на View) са тези с които потребителят взаимодейства;
- ViewGroup обектите (по точно, обектите наследници) са тези които „съдържат“ други View или ViewGroup обекти; Тоест това са контейнерите определящи оформлението (layout) на потребителският интерфейс;
- В sdk на Android има достатъчно на брой наследници на View и ViewGroup класовете с които можем да използваме за UI на нашето приложение;

Бърз поглед на Android UI cont.



- Йерархия от View (и ViewGroup) обекти определя UI на всеки компонент;
- Колкото по простичка е тази йерархия, толквоа по „леки" са нашите компоненти;
- Самите View и ViewGroup обекти, можем да съсздаваме използвайки Java code;
- Значително по лесно обаче е да използваме .xml, тъй като това ни дава възможност за значително по четимо оформление (layout); А и е много близък до добре познатия ни HTML(5) markup ;);
- Оибкновено името на даден .xml елемент, представящ даден View клас е същото като на този клас – например:
- ViewGroup => LinearLayout class => <LinearLayout> .xml
- View => Button class => <Button> .xml
- Примерен LinearLayout със няколко компонента:

Бърз поглед на Android UI cont.

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <Button
        android:id="@+id/_btn_01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I'm a clickable thing 1..." />
    <TextView
        android:id="@+id/_txt_view_01"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="I'm a simpe text view;" />
    <Button
        android:id="@+id/_btn_02"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="I'm a clickable thing 2..." />

</LinearLayout>
```

Layout(s) – какво са те?

- Вместо определение – контейнери (ViewGroups), определящи структурата на UI за компоненти като Acitivities (или widgets);
- Обикновено ги дефинираме като .xml markup;
- Можем и чрез код (hard it is…);
- Това се отнася както ViewGroups, така и за Views;
- Обикновено default състоянията и на двата типа обекти, дефинираме в .xml; Докато чрез код променяме тези състояние и „управляваме“ нашите UI компоненти;
- Така представянето е разделено от логиката (VC частта от MVC);
- Както стана дума речника на .xml markup-а с който дефинираме View (UI) елементи, следва доста близко имената на View класовете и техните методи; Тоест името на даден .xml елемент съвпада с името на View класа, който представя, а името на негов атрибут съвпада с името на метод в този клас;

Layout(s) – какво са те? cont.

- Layout(s) записваме (обикновено) в res/layout папката на нашия проект;
- 'res' right-click => New => Layout Resource File
- Попълваме име на файла и вида на layout-а (LinearLayout, RelativeLayout);
- Всеки layout файл трябва да съдържа точно един root елемент който е ViewGroup /или View обект (с това внимаваме, тъй като обикновенно не се cast-ва към ViewGroup и проблемите почват още в design mode на Android Stuido)/;
- След като имаме root контейнера, можем да добавим (допишем или довлечем) останалите си UI обкети;

Layout(s) – load it.

- Имаме layout, попълнен със Views; Как да го „заредим“?
- При компилиране, всеки layout бива компилиран във View ресурс;
- Във Activity.onCreate();
- Използвайки R.layout.<our_layout_file_name>
- Извикваме setContentView(R.layout. …);
- onCreate() е извикван от Android, когато нашето Activity бива стартирано (преди всички останали callback методи);

Layout(s) – параметри (атрибути).

- Или по друг начин казано – какво най-често ще използваме като атрибути на View и ViewGroup обектите?
- Всеки от тях си има свое собствено м-во от атрибути. Някой специфични за конкретен View клас (и неговите обекти) и биват наследени от неговите наследници;
- Други са общи за всички View обкети – the mighty ID atribute ;)
- Трети пък са т.нар. Layout параметри – т.е. такива описващи ориентацията на даден View обект в рамките на съдържащия го ViewGroup обкет;

the mighty ID atribute;

- Всеки View обект (и ViewGroup обектите, като наследници на View), може да притежата int ID, което еднозначно го идентифицира във View йерархията;
- В .xml markup-а обикновено го описаваме като String стойност за атрибута id; След компилиране обаче, стойността му е int;
- .xml синтаксис: android:id="@+id/my_view"
- @id – това трябва да бъде интерпретурано като ID ресурс от .xml парсера;
- + - това се интерпретира като нов ID ресурс;
- След компилация, нашият така зададен нов ресурс се намира във файла R.java (R.id.my_view);
- Освен ID ресурсите които ние дефинираме, Android също разполага с такива. В .xml към тях се обръщаме (ги реферираме, указваме) последният начин: android:id="@android:id/<some-android-resource-id>"
- Разполагайки с resource ids за нашите Views, в кода можем да ги използваме, разчитайки на findViewById(R.id. …) метода, който да интанциира нашият View обект, разчитайки на .xml markup-а (обиновено в onCreate() метода на Activity);
- ID атрибутите, са особено важни когато използваме RelativeLayout. Там съставните обекти могат да определелят положението си, спрямо останалите, за което се използват ID атрибутите;

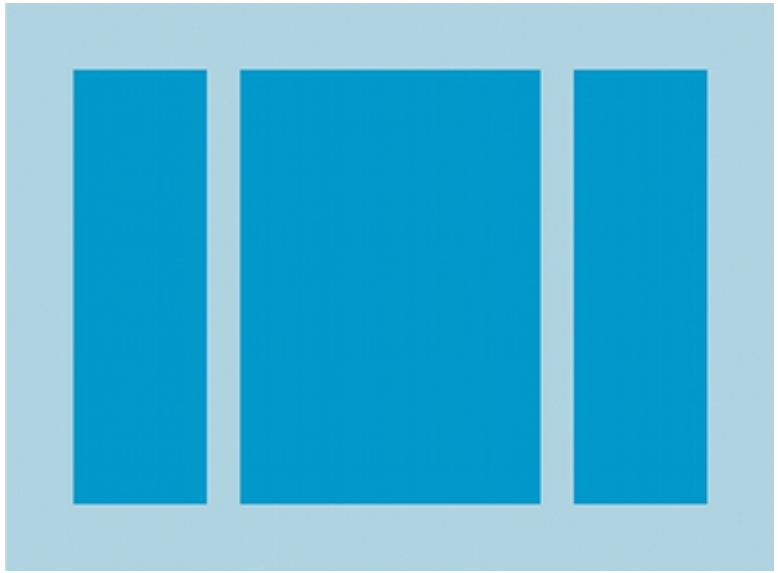
Layout(s) – параметри (атрибути). cont.

- Атрибутите имащи вида layout_<thing> в .xml markup-a, определят „разположение“ за дадения View обект, спрямо ViewGroup обекта в който той се намира;
- Всички ViewGroup обекти дефинират _width и _height layout атрибути и всички View обекти, съдържащи се в тях са задължени да ги дефинират;
- Можем да укажем _width и _height с точни размери (например в px/dp), но това не е препоръчително;
- Най-често се използват следните:
- wrap_content – View обекта ни е толкова голям, колкото и неговото съдържание;
- match_parent (fill_parent) – View обекта ни е толкова голям, колкото този който го съдържа (т.е. колкото му позволи да бъде голям);

Layout(s) – параметри (атрибути). cont.

- Още няколко, но вече от страна на Java Code :)
- Position – getLeft(), getTop(), getRight(), getBottom();
- Тъй като нашите View обекти са правоъгълници, тяхното положение се определя от двойката (left/X, top/Y) координати и двойката (width, height). Горните са методите с които можем да ги вземем (спрямо съдържащият го родител са тези координати);
- Size – getMeasuredWidth(), getMeasuredHeight(), getWidth(), getHeight();
- Разликата – колкого големи искаме да бъдем и колко всъщност големи сме начертани ;) /отчитайки padding и други ограничения/;

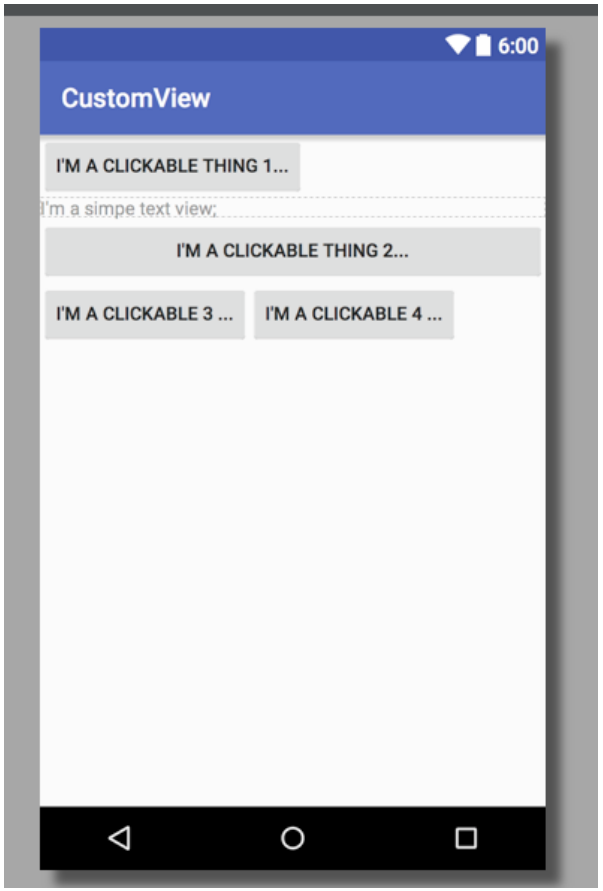
Два основни layout(s).



- LinearLayout – организира своите View обкети в редица (хоризонтална или вертикална); Създава scrollbar, ако дължината им е по голяма от тази на екрана;
- RelativeLayout – мястото на един View обект се определя, спрямо друг View обект или родителя; Това е валидно за всеки View обект във RelativeLayout;

Quick Quiz ;)

- Използвайки .xml markup-а от слайд #4, можете ли да създатете layout изглеждащ така:



User Input. Some Input Controls.

Тип на контрола	Опитане	Class
Button	Tap, Click, Press от страна на потребителя;	Button
Text field	Текстово поле, позволяващо редактиране на текст	EditText
Checkbox	On/Off селектор, управляван от потребителя; Както в HTML;	CheckBox
Radio button	On/Off флаг, но в група от опции; Само една опция може да бъде избрана; Както в HTMLl	RadioGroup || RadioButton
Toggle button	On/Off бутон, показващ състоянията си;	ToggleButton
Spinner	Списък, появяващ се, който позволява на потребителите да изберат една от възможните опции;	Spinner

Input Controls. Some notes...

- Можем да експеремтираме с изброениете, използвайки Design режима на Android Studio или пишейки .xml markup на ръка;
  Последнто по-бързо ще ни убеди, че .xml markup-а не е толкова страшен и обемен /разполагаме с доста подсказки от страна на Android Stuido/;
- Все пак едно-две важни неща, касаещи всеки View обект в нашият layout;
- За целта ще използваме (отново) Button View;

Not only for Button(s) ...

```
<Button
    android:id="@+id/_btn_05"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"

    android:text="@string/_btn_05_txt"
||
    android:src="@drawable/_btn_05_icon"
/>
```

- @string/_btn_01_txt – взима текста на нашият бутон от res/values/strings.xml ресурса на проекта;
- @drawable/_btn_01_icon – съответно нашия бутон ще има икона, намираща се във файла res/drawable/_btn_01_icon.png;
- Така като е написано – или текст или икона;
- Ако искаме и двете android:src => android:drawableLeft;

Not only for Button(s) ...

- Как отговаряме на click събития ?
- Първи вариант - .xml атрибут:
- Атрибут: android:onClick="btn_05_click"
- След което в Activity-то, съдържащато този layout трябва да имаме:
- public void btn_05_click(View view) { … } метод;
- Втори вариант – onClickListener();
- Тоест веднага след „инстанциирането" на нашия View обкет, трябва да му прикачим и onClickListener();

Not only for Button(s) ...

```
Button _btn_05 = (Button) findViewById(R.id._btn_05);

_btn_05.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // do something upon user click ...
    }
});
```

What about custom Views? Live code :-

- Както спонехме, всичко което правим с помощтта на .xml markup, касаещо нашите View и ViewGroup обекти, можем и през Java code;
- Тоест целта ни е да си направим custom view ;)
- По точно, целта ни е да видим основните стъпки, през които трябва да минем за да осъществим това;

What about custom Views? Live code :-

- За да създадем наплъно cutom View, трабва да:
- Наследим View /не е изненада, всичко видимо произлиза от него :)/
- Напишем „подходящ“ конструктор;
- Добавим подходящи setters/getters (modifiers);
- Със сигурност обаче, ще трябва да предефинираме onMeasure() и onDraw() методите на View класа;
- Поведението по подразбиране на onDraw() е: върши нищо;
- Поведението по подразбиране на onMeasure() е: установява 100x100 px големина на нашият View обкет /т.е. едва ли ще ни е достатъчно това :)/;
- Примерен проект: TryCust.zip;