**Приложение к уроку 2**

Приложение без интерфейса это просто код. Он понятен программистам и ничем не поможет конечным пользователям. Чтобы приложение работало с людьми, ему нужен интерфейс. Он создает диалог между программой и пользователем с помощью графики и визуальных аналогий. Отрисованные кнопки, списки и меню — понятный аналог голого программного кода. Интерфейс интерактивен — выполняет полезное действие после того, как его задействовал пользователь. Благодаря интерактивности интерфейса приложение и вы понимаете друг-друга.

Аббревиатура UX расшифровывается как User Experience (пользовательский опыт) — это ощущения пользователя, возникающие при взаимодействии с продуктом, сервисом или услугой.

А UX-дизайн — это создание полезных, простых и приятных в использовании продуктов.

UI(Интерфейс пользователя) - совокупность средств и методов, при помощи которых пользователь взаимодействует с различными, чаще всего сложными, машинами, устройствами и аппаратурой. GUI(Графический интерфе́йс по́льзователя) - разновидность пользовательского интерфейса, в котором элементы интерфейса (меню, кнопки, значки, списки и т. п.) , представленные пользователю на дисплее, исполнены в виде графических изображений. Ну и как бы в чём различие очевидно. В UI входит: GUI, текстовый интерфейс, тот же пульт к телевизору и тдтп, т. е все способы взаимодействия человека с машиной (программой) . GUI - это конкретная реализация UI. Это кнопочки, картинки и тд.

Есть хорошие интерфейсы, есть не очень. Есть удобные, есть не очень. И пользователь всегда выбирает то, что стильно и удобно. Много однообразных приложений, но разных по стилю и опциям.

Стиль приложения определяет пользователь.

 Выявление идеи и создание ТЗ. Мы детализируем идею приложения и набрасываем пути перемещения пользователя в будущем интерфейсе. Затем составляем точное ТЗ для дизайнеров и программистов.

Проектирование каркаса. Проводится разметка активных и неактивных зон, отводится место для самых важных функций и дополнительные развертки для второстепенных. Параллельно начинается работа над альтернативным макетом и прорабатывается интерфейс приложения на андроид или iOS, если это условие заказа.

Как только мы разобрались с эскизом, самое время приступить к карте экранов и понять, как же будет вести себя обычный человек при использовании будущего инструментария приложения и какое состояние будет принимать интерфейс при каждом взаимодействии пользователя с ним.

3 В студии макет **LinearLayout** представлен двумя вариантами - **Horizontal** и **Vertical**. Макет **LinearLayout** выравнивает все дочерние объекты в одном направлении — вертикально или горизонтально. Направление задается при помощи атрибута ориентации **android:orientation**:

* android:orientation="horizontal"
* android:orientation="vertical"

Все дочерние элементы помещаются в стек один за другим, так что вертикальный список компонентов будет иметь только один дочерний элемент в ряду независимо от того, насколько широким он является. Горизонтальное расположение списка будет размещать элементы в одну строку с высотой, равной высоте самого высокого дочернего элемента списка.

### Экраны

Для начала немного теории по экранам. Экран имеет такие физические характеристики как диагональ и разрешение. Диагональ – это расстояние между противоположными углами экрана, обычно измеряется в дюймах. Разрешение – кол-во точек по горизонтали и вертикали, которое экран способен отобразить, измеряется в пикселах.

**dp**или **dip**- Density-independent Pixels. Абстрактная ЕИ, позволяющая приложениям выглядеть одинаково на различных экранах и разрешениях.

**sp**- Scale-independent Pixels. То же, что и dp, только используется для размеров шрифта в View элементах

**px**– пиксел, не рекомендуется использовать т.к. на разных экранах приложение будет выглядеть по-разному.

**mm**– миллиметр, определяется по физическому размеру экрана

**in**– дюйм, определяется по физическому размеру экрана

in, mm и pt – неизменны относительно друг друга. Всегда **1 in = 25,4 mm** и **1 in = 72 pt**. Это классические единицы измерения. Т.е. задаете, например, кнопке ширину = 1 in и она должна отобразиться шириной в один дюйм, независимо от разрешения и диагонали экрана.

Что такое px, думаю, тоже понятно. Если у вас есть устройство с экраном шириной 480 px и вы создали кнопку шириной 240 px, то эта кнопка займет в ширину ровно пол-экрана. Но если вы откроете ваше приложение на устройстве с экраном с меньшим разрешением, то соотношение  изменится, например:  
- если разрешение 320х240, ширина экрана = 240 px. Кнопка займет уже не пол-экрана в ширину, а весь экран  
- если же разрешение 1280х800, ширина = 800 px. Кнопка опять же будет занимать в ширину не пол-экрана, а чуть меньше трети

А ведь экран приложения – это обычно не одна кнопка, а набор из многих элементов и все они будут расползаться или сжиматься на разных разрешениях. Поэтому использовать px при разработке НЕ рекомендуется.

Для того, чтобы избежать таких ситуаций на разных разрешениях рекомендуется использовать dp (и sp).

Кол-во пикселей в одном дюйме (dpi) (dot per inch)

Возьмем в качестве примера экран смартфона HTC Desire. Диагональ = 3,7 дюйма, разрешение = 800х480 пикселей.

Диагональ = 3,7 дюйма

Разрешение = 800х480 пикселей.

Кол-во пикселей в одном дюйме (dpi) (dot per inch)

Диагональ = = 932  
dpi = 932 / 3.7 = 252  
Т.е. в одном дюйме экрана помещается ряд из 252 пикселей.

Помимо этого dp и sp позволяют автоматически масштабировать компоненты в зависимости от размера экрана пользователя. Его можно определить, как масштабируемый px.

За степень масштабируемости отвечает **[Screen Density](http://developer.android.com/guide/practices/screens_support.html" \t "_blank)**. Это коэффициент, который используется системой для вычисления значения dp.  На текущий момент есть 5 значений этого коэффициента:  
- low (**ldpi**) = 0,75  
- medium (**mdpi**) = 1  
- tv (**tvdpi**) = 1,33  
- high (**hdpi**) = 1,5  
- extra high (**xhdpi**) = 2

Т.е. когда для экрана стоит режим **mdpi**, то**1 dp = 1 px**. Т.е. кнопка шириной 100 dp будет выглядеть также как и кнопка шириной 100 px.

Если, например, у нас экран с низким разрешением, то используется режим **ldpi**. В этом случае**1 dp = 0,75 px**. Т.е. кнопка шириной 100 dp будет выглядеть так же как кнопка шириной 75 px.

Если у нас экран с высоким разрешением, то используется режим **hdpi**или **xhdpi**. **1 dp = 1, 5 px** или **2 px**. И кнопка шириной 100 dp будет выглядеть так же как кнопка шириной 150 px или 200 px.

Т.е. при различных разрешениях используются различные Density режимы, которые позволяют приложениям масштабироваться и выглядеть если не одинаково, то, по крайне мере, похоже на всех экранах.

### Layout width и Layout height

Про ширину (layout\_width) и высоту (layout\_height) мы уже немного говорили на прошлом уроке. Мы можем указывать для них абсолютные значения, а можем использовать константы.

#### Константы

match\_parent (fill\_parent) – означает, что элемент займет всю доступную ему в родительском элементе ширину/высоту.

wrap\_content – ширина/высота элемента будет определяться его содержимым

Переименуем ConstraintLay в LinearLay, зададим ему вертикальную ориентацию.

Добавим еще две кнопки.

А теперь попытаемся эти две кнопки скомпоновать в горизонтальный лэйаут.

У разметки **LinearLayout** есть интересный атрибут **android:layout\_weight**, который назначает индивидуальный вес для дочернего элемента. Этот атрибут определяет "важность" представления и позволяет этому элементу расширяться, чтобы заполнить любое оставшееся пространство в родительском представлении. Заданный по умолчанию вес является нулевым.

Важное условие, для каждого компонента должна быть высота задана нулевая:

**android:layout\_height="0dp"**

Пробуем задать для одного

**android:layout\_weight="0"**

для остальных

**android:layout\_weight="1"**

А затем для всех **android:layout\_weight="1"**

Также можно указать атрибут **android:weightSum**. Если атрибуту присвоить значение 100, то можно указывать вес дочерних элементов в удобном виде, как в процентах. Такой способ широко используется веб-мастерами при вёрстке.

Свойство задается для всего лэйута:

**android:weightSum="100"**

а веса прописываются для каждого компонента.

Теперь смотрим, что все компоненты у нас сбились в кучу.

Для установки внутренних отступов применяется атрибут android:padding. Он устанавливает отступы контента от всех четырех сторон контейнера. Можно устанавливать отступы только от одной стороны контейнера, применяя следующие атрибуты: android:paddingLeft, android:paddingRight, android:paddingTop и android:paddingBottom.

Для установки внешних остступов используется атрибут layout\_margin. Данный атрибут имеет модификации, которые позволяют задать отступ только от одной стороны: android:layout\_marginBottom, android:layout\_marginTop, android:layout\_marginLeft и android:layout\_marginRight (отступы соответственно от нижней, верхней, левой и правой границ):

android:layout\_margin="10dp"

Пробуем сделать это на практике. Скачиваем картинки

Сначала накидываем компоненты

Пытаемся их выстроить пропорционально вручную.

Компонент кнопка. Одним из важных компонентов пользовательского интерфейса в приложения является кнопка. Она используется для выполнения различных действий пользователя.

Если вы разместили на экране кнопку и будете нажимать на неё, то ничего не произойдёт.

Необходимо написать код, который будет выполняться при нажатии. Существует несколько способов обработки нажатий на кнопку.

Относительно новый способ, специально разработанный для Android - использовать атрибут **onClick** (на панели свойств отображается как **On Click**):

android:onClick="onMyButtonClick"

Имя для события можно выбрать произвольное, но лучше не выпендриваться. Далее нужно прописать в классе активности придуманное вами имя метода, который будет обрабатывать нажатие. Метод должен быть открытым (public) и с одним параметром, использующим объект **View**. Вам нужно выучить пять слов для создания метода, а сам метод поместить в класс (если вы ещё путаетесь в структуре Java-кода, то вставьте метод перед последней фигурной скобкой):