[Как создать сustom View для Android Ч1.: Расширяем View класс и добавляем новые xml атрибуты](http://codeandlife.ru/android/1-creating-custom-android-views-part-1.html" \o "Как создать сustom View для Android. Ч1.: Расширяем View класс и добавляем новые xml атрибуты)

Введение

Это будет серия постов на тему создания пользовательского view компонента для Android. Я покажу как нарисовать содержимое компонента, как работают layouting и measuring, как реализовать view groups и как анимировать содержимое компонента. В этом посте я объясню, как расширить стандартный view, как его использовать и как создать свои собственные xml атрибуты.

Специфические задачи, специфические view

Стандартные view компоненты, которые предоставляет Android, могут использоваться для многих задач и ситуаций. Тем не менее, в наших приложениях наибольшую часть кода часто занимает конфигурация этих view для специфических задач. Этот конфигурационный код часто находится в activity вашего приложения, поэтому их сложно содержать в чистоте. А при наличии большого количества различного кода возникают сложности и с его разделением в отдельные классы.

Предположим, вы работаете над приложением, которое содержит статистику тренировок пользователя. Например, общая дистанция, общее время и так далее, в зависимости от типа тренинга. Для того чтобы отображать данные в удобном виде, вы решили их адаптировать, основываясь на их продолжительности. Если это 2378 секунд, вы хотите отображать это время как "40 минут", а если это 18550 секунд, то лучше было бы отображать на дисплей "5 часов 9 минут".

Создание custom view

Один из способов решения этой задачи - создать специализированный view, который будет должным образом обрабатывать ваш текст. Давайте начнем с создания нашего собственного класса DurationTextView, который будет расширять класс TextView.

|  |  |
| --- | --- |
| 1  2  3  4  5 | public class DurationTextView extends TextView {      public DurationTextView(Context context, AttributeSet attrs) {          super(context, attrs);      }  } |

**TextView, как и все view классы, имеет три различных конструктора: первый просто принимает контекст (Context), второй принимает контекст и набор атрибутов (AttributeSet), а третий еще дополнительно принимает стиль "по умолчанию"**. **В большинстве случаев вам достаточно реализовать второй конструктор, который показан выше.**

Теперь мы создали view  и чтобы его использовать, мы добавим его в файл разметки:

|  |  |
| --- | --- |
| 1  2  3  4 | <com.example.android.customviewp1.DurationTextView          android:id="@+id/durationview"          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content" /> |

Обратите внимание, вам необходимо полностью указывать имя вашего класса, который вы реализовали.

В данный момент этот класс практически идентичен стандартному TextView, поэтому давайте добавим ему некоторую функциональность.

Добавление логики отображения

Поскольку этот view будет отображать длительность, давайте добавим метод setDuration().

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | public void setDuration(float duration) {          int durationInMinutes = Math.round(duration / 60);          int hours = durationInMinutes / 60;          int minutes = durationInMinutes % 60;            String hourText = "";          String minuteText = "";            if (hours > 0) {              hourText = hours + (hours == 1 ? " hour " : " hours ");          }            if (minutes > 0) {              minuteText = minutes + (minutes == 1 ? " minute" : " minutes");          }            if (hours == 0 && minutes == 0) {              minuteText = "less than 1 minute";          }            String durationText = String.format(TEMPLATE, hourText + minuteText);          setText(Html.fromHtml(durationText), TextView.BufferType.SPANNABLE);      } |

Что делает этот метод? Он получает значение длительности, переводит его в текстовую строку в определенном формате, используя некую логику, а затем выводит с помощью метода setText(). В данном случае логика заключается в переводе секунд в минуты и дальнейшем разделении на  часы и минуты. Также в методе присутствует логика для корректного отображения одной минуты: "1 minute" вместо "1 minute(s)".

В конце метода отформатированное время преобразуется в строку с помощью шаблона.

**private static final String TEMPLATE = "Duration: <strong>%s</strong>";**

Шаблон начинается со слова "Duration", а далее жирными буквами отображается отформатированная строка.

Для простоты я использовал в коде строковые литералы. В нормальном коде, эти строковые литералы  лучше перенести в strings.xml, это позволяет в дальнейшем локализовать их.

Использование полученного view

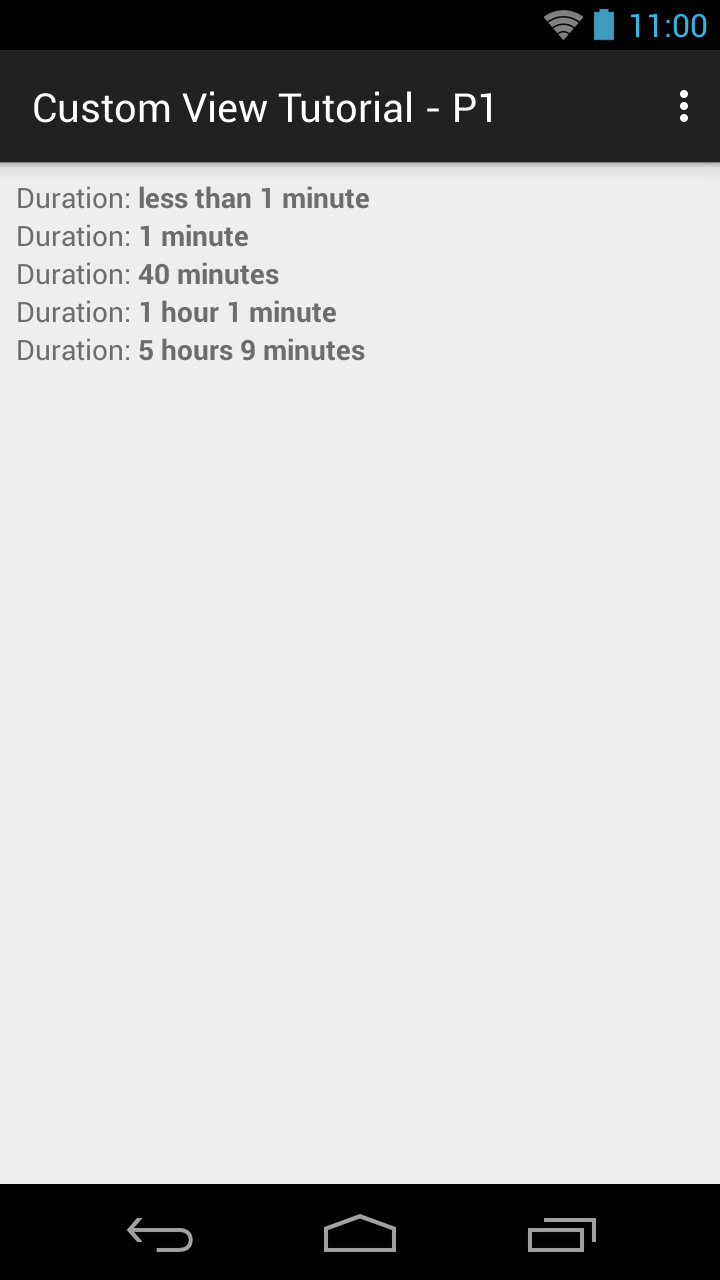
Попробуем использовать наш view. В xml файл разметки добавим 5 custom View.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | <com.example.android.customviewp1.DurationTextView          android:id="@+id/durationview1"          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content" />        <com.example.android.customviewp1.DurationTextView          android:id="@+id/durationview2"          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content" />        <com.example.android.customviewp1.DurationTextView          android:id="@+id/durationview3"          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content" />        <com.example.android.customviewp1.DurationTextView          android:id="@+id/durationview4"          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content" />        <com.example.android.customviewp1.DurationTextView          android:id="@+id/durationview5"          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content" /> |

А в activity в методе onCreate() после метода setContentView() добавим следующие строки.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | DurationTextView durationView1 = (DurationTextView) findViewById(R.id.durationview1);  durationView1.setDuration(25);    DurationTextView durationView2 = (DurationTextView) findViewById(R.id.durationview2);  durationView2.setDuration(78);    DurationTextView durationView3 = (DurationTextView) findViewById(R.id.durationview3);  durationView3.setDuration(2378);    DurationTextView durationView4 = (DurationTextView) findViewById(R.id.durationview4);  durationView4.setDuration(3670);    DurationTextView durationView5 = (DurationTextView) findViewById(R.id.durationview5);  durationView5.setDuration(18550); |

Соберем проект и загрузим его в эмулятор. Полученный результат можно видеть на изображении ниже.



Добавление xml атрибутов

Если бы мы могли задавать шаблон, то созданный нами компонент был бы более универсальным. По идее можно добавить метод, который позволит нам устанавливать шаблон, однако это не так просто. Если подумать, длительности нам нужно устанавливать динамически, а шаблон скорее всего будет все время одинаковым. Поэтому вместо метода, давайте добавим нашему custom view новые xml атрибуты.

**В первую очередь нужно создать в папке values файл attrs.xml, где мы сможем определить атрибуты.** Для этого view мы добавим один атрибут - template типа string. Выглядеть это будет так:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <?xml version="1.0" encoding="utf-8"?>  <resources>      <declare-styleable name="TemplateTextView">          <attr name="template" format="string" />      </declare-styleable>  </resources> |

Мы записали тег declare-styleable и задали имя TemplateTextView. Имя может быть произвольным, но обычно задают имя, совпадающее с именем view. В данном случае я не задаю имя DurationTextView, потому что планирую использовать template атрибут еще в другом view.

Далее мы определили новый атрибут, установив ему имя template и формат string. Помимо string существует множество других форматов, например, Color, Integer, Boolean, Reference и так далее.

Использовать новый атрибут в layout xml можно так.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | <xml version="1.0" encoding="utf-8">  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"      xmlns:tools="http://schemas.android.com/tools"      android:layout\_width="match\_parent"      android:layout\_height="match\_parent"      android:orientation="vertical"      android:padding="8dp"      tools:context=".MainActivity"      xmlns:app= "http://schemas.android.com/apk/res-auto">        <com.example.android.customviewp1.DurationTextView          android:id="@+id/durationview1"          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content"          app:template="%s was spent running"/>    </LinearLayout> |

Обратите внимание, в корневом элементе мы добавляем такую строчку.

**xmlns:app= "http://schemas.android.com/apk/res-auto"**

Это позволяет нам использовать атрибуты, которые определены в файле attrs.xml, в нашем layout. В данном случае мы задали префикс "app", хотя он может быть произвольным.

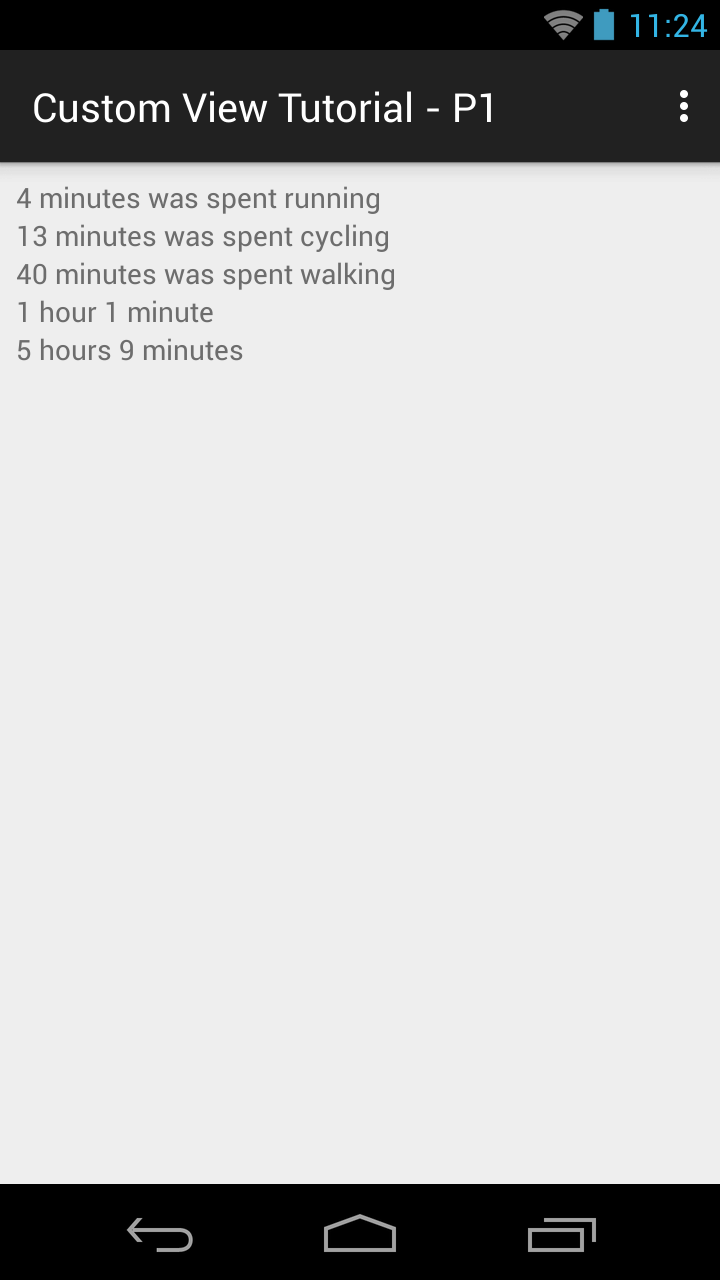
Для того чтобы использовать новый атрибут, нужно получить его значение в нашем custom view. Сначала заменяем приватную статическую константу "TEMPLATE" переменной "template".  Затем добавляем в наш конструктор следующий код.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | private String template;        public DurationTextView(Context context, AttributeSet attrs) {          super(context, attrs);          TypedArray attributes = context.obtainStyledAttributes(attrs, R.styleable.TemplateTextView);          template = attributes.getString(R.styleable.TemplateTextView\_template);            if (template == null || !template.contains("%s")) {              template = "%s";          }            attributes.recycle();      } |

В первой строке мы получаем набор атрибутов, который содержит все атрибуты, существующие в xml файле. Метод obtainStyledAttributes() делает две основные вещи. Во-первых, он фильтрует все атрибуты из attrs с помощью контекста, применяет стили и resolves reference to values. Во-вторых, он возвращает только те атрибуты, которые вы определили. Они задаются вторым аргументом, который представляет собой массив ссылок на требуемые атрибуты. В данном случае в R.styleable.TemplateTextView у нас только один атрибут R.attr.template, который мы создали в файле attrs.xml.

Далее мы используем массив attributes, чтобы получить значение атрибута. Если значение шаблона не было определено или не содержит "%s", мы устанавливаем template = "%s". И в заключении нужно не забыть освободить ресурсы с помощью функции recycle().

После некоторых изменений в layout, приложение будет выглядеть так:



Здесь я установил шаблон для первых трех custom view

Большая часть view в Android имеет методы, которые позволяют устанавливать значения XML атрибутов в коде. Возможно вам тоже понадобиться добавить какой-нибудь метод, это зависит от того, как вы будете использовать view.

В следующей части мы посмотрим как нарисовать свой собственное содержимое view компонента и сделаем [view для отображения графика](http://codeandlife.ru/android/2-creating-custom-android-views-part-2.html).

Исходники к статье можно скачать c [GitHub](https://github.com/CodeAndLife/Custom-view-1" \t "_blank).    
По материалам сайта [Jayway](http://www.jayway.com/2012/06/25/creating-custom-android-views-part-1-extending-standard-views-and-adding-new-xml-attributes/" \t "_blank)Вольный перевод - Pavel Bobkov.