**Conheça o RecyclerView**

https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html

A Google lançou mais um componente muito bacana pro Android. O RecyclerView, que assim como o ListView, tem a missão de exibir uma listagem de dados para o usuário.

Quais são as vantagens deste componente:

* Ele é mais modularizado.  
  Isso significa que você consegue colocar comportamentos nele que você não conseguiria usando ListView.   
  Por exemplo: É EXTREMAMENTE simples, sua lista virar uma grid, usando o RecyclerView.
* Ele é mais performático.   
  O RecyclerView, te obriga a implementar o padrão ViewHolder que impede algumas chamadas do método findViewById desnecessariamente.  
  Internamente ele tem um nível de abstração maior entre modelo(sua lista de dados) e view(o Adapter), ações como animações não vão ficar a cargo do Adapter deixando o componente mais performático.

Resumindo ele é mais fácil de usar e é mais rápido.

Vamos ao código pra entender esse componente na prática.

**Android Studio**

Diferente dos outros artigos desse blog, desta vez vou usar o Android Studio pra desenvolver o exemplo.

Se você ainda usa o Eclipse, é melhor já ir se aventurando no Android Studio, pois ele é a IDE oficial do Android.

Apesar da mudança não ser tão leve(as IDE’s são bem diferentes), ela é necessária.

Depois que você estiver adaptado vai valer a pena, o Android Studio tem muitos recursos bacanas.

Um desses recursos legais é o Gradle, que gerencia as suas dependências.

No arquivo Gradle do seu projeto adicione a dependência do RecyclerView.

dependencies {

compile **'com.android.support:recyclerview-v7:22.2.0'**}

Pronto, agora já podemos usar o RecyclerView no nosso projeto.

**Projeto de exemplo**

No nosso projeto, vamos fazer uma listagem de carros.

A nossa tela vai ter apenas o RecyclerView que vai mostrar os dados. Abaixo da lista teremos dois botões, o primeiro pra mudar o tipo de layout(Grid e Lista) e o segundo vai adicionar um novo carro a listagem.

Então a primeira classe que vamos escrever é o nosso modelo, o Car:

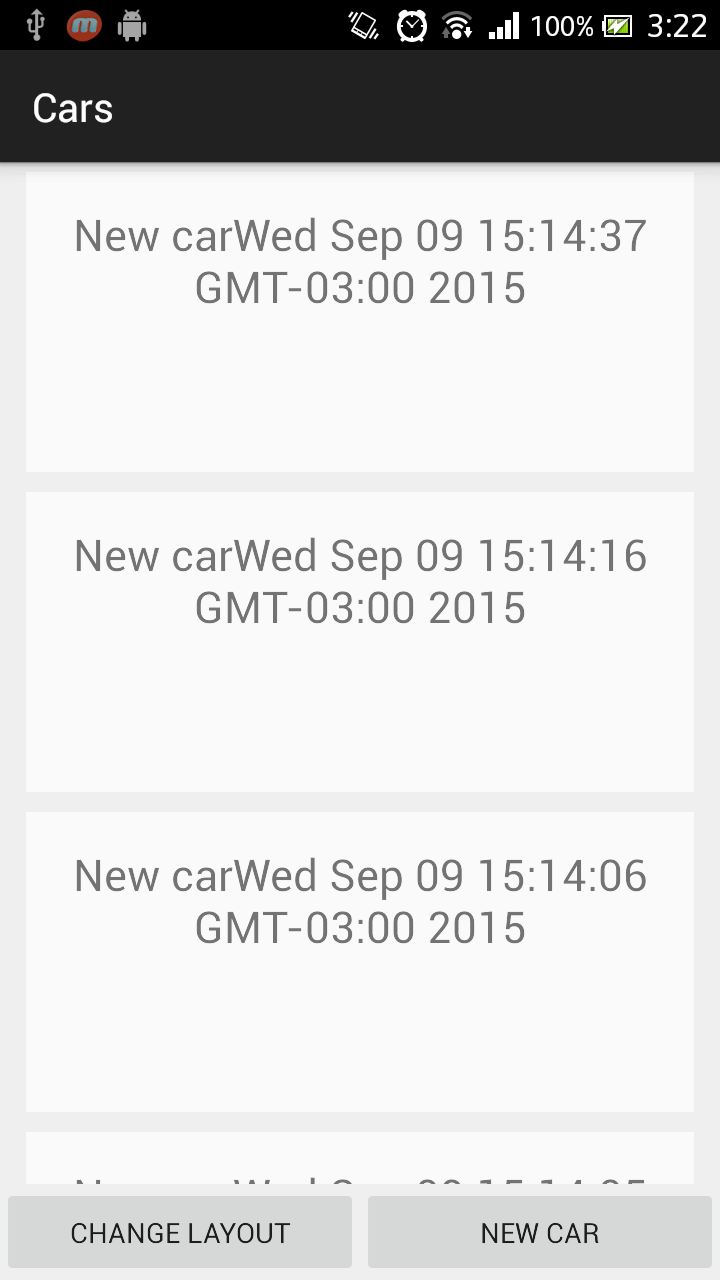
**public class** Car {   
   
 **private** String **name**;  
 **private** String **description**;  
  
 **public** Car() {  
 }  
  
 **public** Car(String name, String description) {  
 **this**.**name** = name;  
 **this**.**description** = description;  
 }  
  
 **public** String getName() {  
 **return name**;  
 }  
  
 **public void** setName(String name) {  
 **this**.**name** = name;  
 }  
  
 **public** String getDescription() {  
 **return description**;  
 }  
  
 **public void** setDescription(String description) {  
 **this**.**description** = description;  
 }  
  
 @Override  
 **public** String toString() {  
 **return name** + **" "** + **description**;  
 }  
}

No final das contas o objetivo é exibir uma listagem desses dados.

Agora vamos apresentar o layout da nossa aplicação.

<**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:background="#efefef"  
 android:orientation="vertical"**>  
  
 <**android.support.v7.widget.RecyclerView  
 android:layout\_weight="1"  
 android:id="@+id/my\_recycler\_view"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:layout\_centerInParent="true"  
 android:layout\_marginLeft="8dp"  
 android:layout\_marginRight="8dp"  
 android:scrollbars="vertical"** />  
  
 <**LinearLayout  
 android:layout\_weight="0"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"**>  
  
 <**Button  
 android:id="@+id/button\_change\_layout"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="1"  
 android:text="Change Layout"** />  
  
 <**Button  
 android:id="@+id/button\_new\_car"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="1"  
 android:text="New Car"** />  
  
 </**LinearLayout**>  
  
</**LinearLayout**>

O layout acima irá gerar a seguinte tela abaixo:



A tela é basicamente uma lista, com dois botões.

A lista é criada com o nosso componente **RecyclerView**.

E os botões irão demonstrar as funcionalidades:

* Como alterar o layout do RecycleView do tipo lista para Grid e vice-versa.
* Como acrescentar um item animadamente.

Bom, então vamos mostrar a nossa Activity que consome esse layout que foi mostrado.

**public class** CarList **extends** AppCompatActivity **implements** CarListAdapter.OnDataSelected {  
  
 **private** List<Car> **cars** = **new** ArrayList<Car>();  
 **private** RecyclerView **recyclerView**;  
 **private** RecyclerView.Adapter **adapter**;  
 **private** LinearLayoutManager **linearLayoutManager**;  
 **private** GridLayoutManager **gridLayoutManager**;  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_main***);  
  
 createFakeCars();  
  
 **recyclerView** = (RecyclerView) findViewById(R.id.***my\_recycler\_view***);  
 **recyclerView**.setHasFixedSize(**true**);  
  
 **linearLayoutManager** = **new** LinearLayoutManager(**this**);  
 **linearLayoutManager**.setOrientation(LinearLayoutManager.***VERTICAL***);  
  
 **gridLayoutManager** = **new** GridLayoutManager(**this**,2);  
  
 **recyclerView**.setLayoutManager(**linearLayoutManager**);  
  
 **adapter** = **new** CarListAdapter(**this**,**this**,**cars**);  
 **recyclerView**.setAdapter(**adapter**);  
   
 findViewById(R.id.***button\_new\_car***).setOnClickListener(**new** View.OnClickListener() {  
  
 @Override  
 **public void** onClick(View v) {  
 **recyclerView**.scrollToPosition(0);  
 Car newCar = **new** Car(**"New car"** + **new** Date().toString(), **"Description"**);  
 **cars**.add(0, newCar);  
 **adapter**.notifyItemInserted(0);  
 }  
  
 });  
  
 findViewById(R.id.***button\_change\_layout***).setOnClickListener(**new** View.OnClickListener() {  
  
 @Override  
 **public void** onClick(View v) {  
 **if**(**recyclerView**.getLayoutManager() **instanceof** GridLayoutManager) {  
 **recyclerView**.setLayoutManager(**linearLayoutManager**);  
 } **else** {  
 **recyclerView**.setLayoutManager(**gridLayoutManager**);  
 }  
 }  
 });  
  
 }  
  
 @Override  
 **public void** onDataSelected(View view, **int** position) {  
 Car selectedItem = **cars**.get(position);  
 Toast.*makeText*(**this**, **"Selected Item: "** + selectedItem.toString(), Toast.***LENGTH\_SHORT***).show();  
 **cars**.remove(position);  
 **adapter**.notifyItemRemoved(position);  
 }  
  
 **private void** createFakeCars() {  
 **for**(**int** i = 0; i < 10; i ++) {  
 Car sampleCar = **new** Car();  
 sampleCar.setName(**"Car "** + i);  
 sampleCar.setDescription(**"Description: "** + i);  
 **cars**.add(sampleCar);  
 }  
 }  
}

**Explicações do código**

No onCreate da nossa Activty, primeiro preenchemos a Lista de Carros com dados falsos, fazemos isso através do método createFakeCar().

Depois recuperamos a referência do RecycleView.

Com o objeto em mãos executamos o método setHasFixed do RecycleView, que ajuda a ganhar desempenho. Uma vez que estamos dizendo que os elementos da lista terão tamanho fixo.

Em seguida, criamos dois LayoutManager. Um do tipo LinearLayoutManager e outro do tipo GridLayoutManager.

Mas o que é isso?

O LayoutManager é responsável por definir o jeito que os dados vão ser apresentados.

Com o LinearLayoutManager, estamos assegurando que a RecyclerView vai se comportar como uma Lista (caso a orientação seja vertical) ou como um Gallery (caso a orientação seja horizontal).

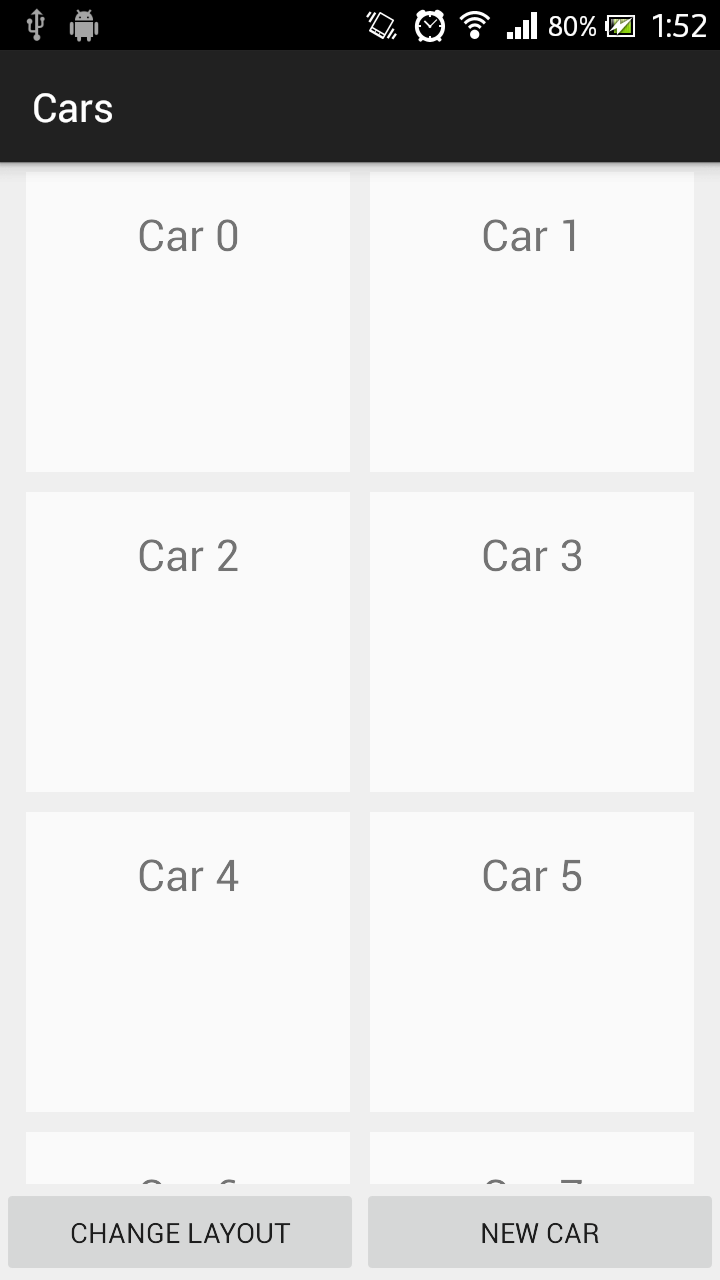
Ex:

**linearLayoutManager**.setOrientation(LinearLayoutManager.***VERTICAL***);

**linearLayoutManager**.setOrientation(LinearLayoutManager. HORIZONTAL);

Já com o GridLayoutManager, vamos definir que o RecyclerView apresente os itens tal como uma Grid.

Veja como fica o RecyclerView no modo Grid:



Depois de criarmos os LayoutManagers, escolhemos iniciar a nossa tela com o LinearLayoutManager.

Pra configurar isso, basta usar o método setLayoutManager do RecyclerView, passando o LayoutManager desejado como parâmetro.

Mais a frente no código, criamos um adapter passando a lista de Carros(Irei falar desse adapter mais a frente) .

Após a criação do adapter, basta vincula-lo ao nosso RecyclerView, através do método setAdapter.

Por último, colocamos as ações nos botões.

A primeira ação de botão apresentada, é a de adicionar um novo carro a lista animadamente.   
Para isso, colocamos o RecyclerView para mover para o topo da lista, através do método scrollToPosition.

Depois criamos e adicionamos um objeto do tipo Car na nossa lista.

Em seguida, acionamos o método notifyItemInserted do adapter, passando como parâmetro a posição onde o item foi inserido, no nosso caso 0.

A segunda ação de botão é a de trocar o estilo do RecyclerView.

Veja que é um código bem simples.

Apenas pegamos o getLayoutManager que esta vigente no RecyclerView através do método getLayoutManager. Se ele for do tipo Grid, configuramos o LinearLayoutManager, caso contrário configuramos o GridLayoutManager no nosso RecyclerView.

**RecyclerView.Adapter**

Recapitulando a função do adapter:

O Adapter é a estrutura que **recebe os dados que vão ser exibidos** e devolve a **View preenchida de cada item**.

Nesse nosso exemplo vamos exibir uma lista de carros.

Então o nosso Adapter vai receber no seu construtor uma Lista de carros e internamente deverá ser capaz de gerar uma View para cada item dessa lista.

Para isso precisamos criar um Layout que representará o carro na lista.

Decidimos somente mostrar o nome do carro, então o Layout abaixo atende bem. Repare que ele só possui um TextView dentro de alguns LinearLayouts que vão dar um efeito de caixa.

<**LinearLayout  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 android:id="@+id/card\_view"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_gravity="center"** >  
  
 <**LinearLayout  
 android:background="#fafafa"  
 android:layout\_margin="5dp"  
 android:minHeight="150dp"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:orientation="vertical"**>  
  
  
 <**TextView  
 android:gravity="center"  
 android:layout\_gravity="center"  
 android:layout\_marginTop="@dimen/activity\_vertical\_margin"  
 android:layout\_marginLeft="@dimen/activity\_vertical\_margin"  
 android:layout\_marginRight="@dimen/activity\_vertical\_margin"  
 android:textSize="22sp"  
 android:id="@+id/info\_text"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"** />  
 </**LinearLayout**>  
</**LinearLayout**>

Beleza, mas qual a diferença deste Adapter para o Adapter que usamos no ListView?

Vamos as diferenças:

O **RecyclerView.Adapter** nos obriga a implementar os seguintes métodos:

**Int getItemCount():** Retorna a quantidade de itens que ele vai gerenciar no total. Basta passar o size, da lista de itens que será passada no seu construtor.

**CarListAdapter.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType):**Método do padrão ViewHolder. Esse padrão garante que teremos um objeto que irá guardar as referências da nossa View.

Fazemos isso por uma questão de desempenho. Ao fazer cache da View não precisamos acionar o método findViewById inúmeras vezes toda vez que o usuário rolar pela nossa lista.

Como resultado teremos uma lista muito mais fluída.

É neste método em questão, que esse objeto ViewHolder vai ser criado. Aqui inflamos a View que vai representar o item da lista e passamos ela como parâmetro para o construtor do ViewHolder que vai gerenciar quando usar os findViewById.

**void onBindViewHolder(CarListAdapter.ViewHolder holder, int position):**   
Outro método do padrão ViewHolder.  
Veja que esse método tem dois parâmetros:

* O ViewHolder já carregado com as Views.
* E a posição do item que devemos mostrar.

Com esses dados em mãos, pegamos o carro pela posição.

Pegamos a View do ViewHolder, e a populamos com os dados do carro.

Esse método é chamado internamente no Android.

Abaixo segue a listagem completa do nosso Adapter:

**public class** CarListAdapter **extends** RecyclerView.Adapter<CarListAdapter.ViewHolder> {  
  
 **private** List<Car> **cars**;  
 **private** Context **context**;  
 **private** OnDataSelected **onDataSelected**;  
  
 **public** CarListAdapter(Context context, OnDataSelected onDataSelected, List<Car> cars) {  
 **this**.**context** = context;  
 **this**.**onDataSelected** = onDataSelected;  
 **this**.**cars** = cars;  
 }  
  
 @Override  
 **public** CarListAdapter.ViewHolder onCreateViewHolder(ViewGroup parent,**int** viewType) {  
 View view = LayoutInflater.*from*(parent.getContext()).inflate(R.layout.***car\_layout***, parent, **false**);  
  
 ViewHolder viewHolder = **new** ViewHolder(view);  
 **return** viewHolder;  
 }  
  
 @Override  
 **public void** onBindViewHolder(ViewHolder holder, **int** position) {  
 Car car = **cars**.get(position);  
 holder.**textViewTitleCar**.setText(car.getName());  
 }  
  
 @Override  
 **public int** getItemCount() {  
 **return cars**.size();  
 }  
  
 **private void** treatOnDataSelectedIfNecessary(View view, **int** position) {  
 **if**(**onDataSelected** != **null**) {  
 **onDataSelected**.onDataSelected(view, position);  
 }  
 }  
  
 **public static interface** OnDataSelected {  
  
 **public void** onDataSelected(View view, **int** position);  
  
 }  
  
 **public class** ViewHolder **extends** RecyclerView.ViewHolder {  
  
 **public** TextView **textViewTitleCar**;  
  
 **public** ViewHolder(View view) {  
 **super**(view);  
 view.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 treatOnDataSelectedIfNecessary(v,getAdapterPosition());  
 }  
 });  
  
 **textViewTitleCar** = (TextView)view.findViewById(R.id.***info\_text***);  
 }  
 }  
}

**Opa! Mas oque é esse OnDataSelected?**

Pois é, tivemos que criar essa interface para resolver um novo problema.

Diferente do ListView, o RecyclerView **não** possui o método setOnItemClickListener. É nele que passamos o Listener que tratará o evento de clique em algum item da lista.

Desse modo para colocar um evento de clique no nosso item. Devemos configurar a ação diretamente nos itens do Adapter.

A ação é representada pela nossa interface OnDataSelected que nos força a implementação do método onDataSelected(View view, int position).

Os parâmetros deste método são respectivamente: A View recém clicada e a posição do item selecionado.

Atentando ao código quem implementou essa nossa interface foi a Activity.

E a ação que confiramos foi a de remover o item recém selecionado (Volte na listagem da Activity e acompanhe).

No construtor do ViewHolder, pegamos a View do car. E atribuímos a ela o OnClickListener, dentro do onClick colocamos que caso tenhamos um OnDataSelected no nosso Adapter, ele deverá ser executado.

Apesar de ter ficado um pouco mais complexo. É uma solução padronizada para tratar ações diferentes em um mesmo item da lista.

Por exemplo: Se o item da nossa Lista tivesse dois TextViews e cada TextView tivesse uma ação diferente. Não teríamos problemas em atender esse cenário. Bastaria acrescentar mais um método abstrato na interface OnDataSelected, e atribui-lo como ação desse segundo TextView.