

```

    android:layout_height="match_parent"
    android:fitsSystemWindows="true" tools:openDrawer="start">>
```

<include
 layout="@layout/app_bar_main"
 android:layout_width="match_parent"
 android:layout_height="match_parent" />

<android.support.design.widget.NavigationView
 android:id="@+id/nav_view"
 android:layout_width="wrap_content"
 android:layout_height="match_parent"
 android:layout_gravity="start"
 app:headerLayout="@layout/nav_header_main"
 app:menu="@menu/activity_main_drawer" />

```
</android.support.v4.widget.DrawerLayout>
```

res/layout/nav_header_main.xml: вид, который будет отображаться в верхней части drawer (бокового меню).

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  

    android:layout_width="match_parent"  

    android:layout_height="@dimen/nav_header_height"  

    android:background="@drawable/side_nav_bar"  

    android:paddingBottom="@dimen/activity_vertical_margin"  

    android:paddingLeft="@dimen/activity_horizontal_margin"  

    android:paddingRight="@dimen/activity_horizontal_margin"  

    android:paddingTop="@dimen/activity_vertical_margin"  

    android:theme="@style/ThemeOverlay.AppCompat.Dark"  

    android:orientation="vertical"  

    android:gravity="bottom">
```

<ImageView
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:paddingTop="@dimen/nav_header_vertical_spacing"
 android:src="@android:drawable/sym_def_app_icon"
 android:id="@+id/imageView" />

<TextView
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:paddingTop="@dimen/nav_header_vertical_spacing"
 android:text="Android Studio" android:textAppearance="@style/TextAppearance.
 AppCompat.Body1" />

<TextView
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="android.studio@android.com"
 android:id="@+id/textView" />

```
</LinearLayout>
```

res/layout/app_bar_main.xml – слой абстракции для панели инструментов, отделяющий ее от содержимого:

```

<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
```

```

xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    tools:context="eu.rekisoft.playground.MainActivity">

    <android.support.design.widget.AppBarLayout
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:theme="@style/AppTheme.AppBarOverlay">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            app:popupTheme="@style/AppTheme.PopupOverlay" />

    </android.support.design.widget.AppBarLayout>

    <include layout="@layout/content_main"/>

    <android.support.design.widget.FloatingActionButton
        android:id="@+id/fab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|end"
        android:layout_margin="@dimen/fab_margin"
        android:src="@android:drawable/ic_dialog_email" />

</android.support.design.widget.CoordinatorLayout>

```

res/layout/content_main.xml – реальное содержимое активности только для демонстрирования, здесь вы поместите свой обычный layout.xml:

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:showIn="@layout/app_bar_main"
    tools:context="eu.rekisoft.playground.MainActivity">

    <TextView
        android:text="Hello World!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

```

Определите файл меню как *res/menu/activity_main_drawer.xml*:

```

<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <group android:checkableBehavior="single">
        <item
            android:id="@+id/nav_camera"
            android:icon="@drawable/ic_menu_camera"
            android:title="Import" />
        <item
            android:id="@+id/nav_gallery"
            android:icon="@drawable/ic_menu_gallery"
            android:title="Gallery" />
        <item
            android:id="@+id/nav_slideshow"
            android:icon="@drawable/ic_menu_slideshow"
            android:title="Slideshow" />
        <item
            android:id="@+id/nav_manage"
            android:icon="@drawable/ic_menu_manage"
            android:title="Tools" />
    </group>

    <item android:title="Communicate">
        <menu>
            <item
                android:id="@+id/nav_share"
                android:icon="@drawable/ic_menu_share"
                android:title="Share" />
            <item
                android:id="@+id/nav_send"
                android:icon="@drawable/ic_menu_send"
                android:title="Send" />
        </menu>
    </item>
</menu>

```

И, наконец, java/main/eu/rekisoft/playground/MainActivity.java:

```

public class MainActivity extends AppCompatActivity
    implements NavigationView.OnNavigationItemSelected {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                    .setAction("Action", null).show();
            }
        });
    }
}

```

```

DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
    this, drawer, toolbar, R.string.navigation_drawer_open, R.string.
    navigation_drawer_close);
drawer.setDrawerListener(toggle);
toggle.syncState();

NavigationView navigationView = (NavigationView) findViewById(R.id.nav_view);
navigationView.setNavigationItemSelectedListener(this);
}

@Override
public void onBackPressed() {
    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
    if (drawer.isDrawerOpen(GravityCompat.START)) {
        drawer.closeDrawer(GravityCompat.START);
    } else {
        super.onBackPressed();
    }
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Раздуваем меню; при этом добавляются элементы на панель действий, если она
    // присутствует.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Здесь обрабатываются нажатия на элементы панели действий. Панель действий
    // будет автоматически обрабатывать щелчки на кнопке Home/Up, пока
    // вы указываете родительскую активность в AndroidManifest.xml.
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
        return true;
    }

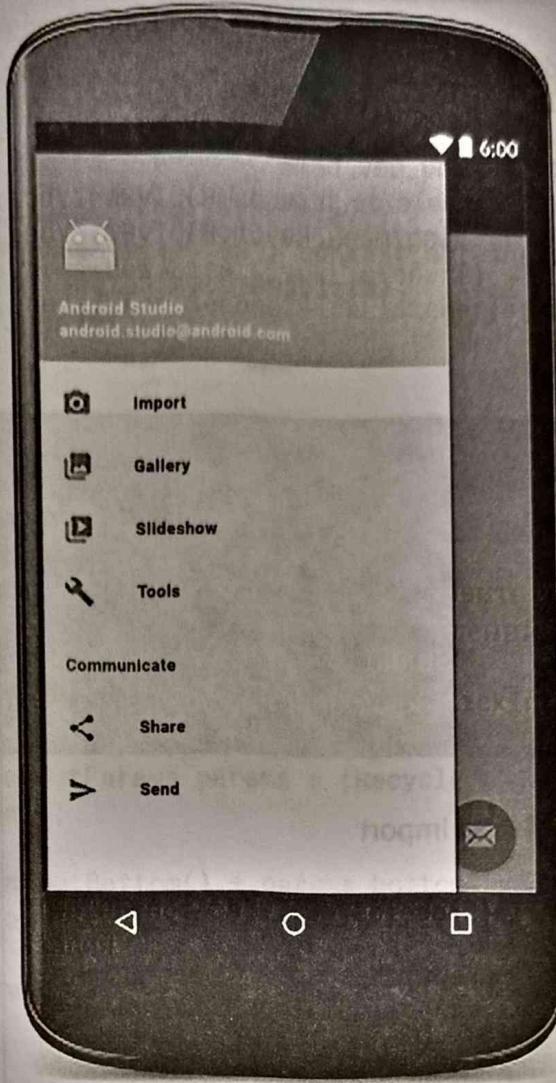
    return super.onOptionsItemSelected(item);
}

@SuppressWarnings("StatementWithEmptyBody")
@Override
public boolean onNavigationItemSelected(MenuItem item) {
    // Здесь обрабатываются нажатия на элементы навигационного представления
    switch(item.getItemId()) {/*...*/}

    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
    drawer.closeDrawer(GravityCompat.START);
    return true;
}
}

```

Он будет выглядеть следующим образом:



15.2. Добавить подчеркивание в элементы меню

Каждая группа заканчивается разделителем строк. Если у каждого пункта меню будет своя группа, вы получите желаемый графический результат. Это будет работать только в том случае, если разные группы имеют разные android:id. Кроме того, в файле menu.xml следует указать следующие параметры: android:checkable="true" для одиночного пункта и android:checkableBehavior="single" для группы пунктов.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

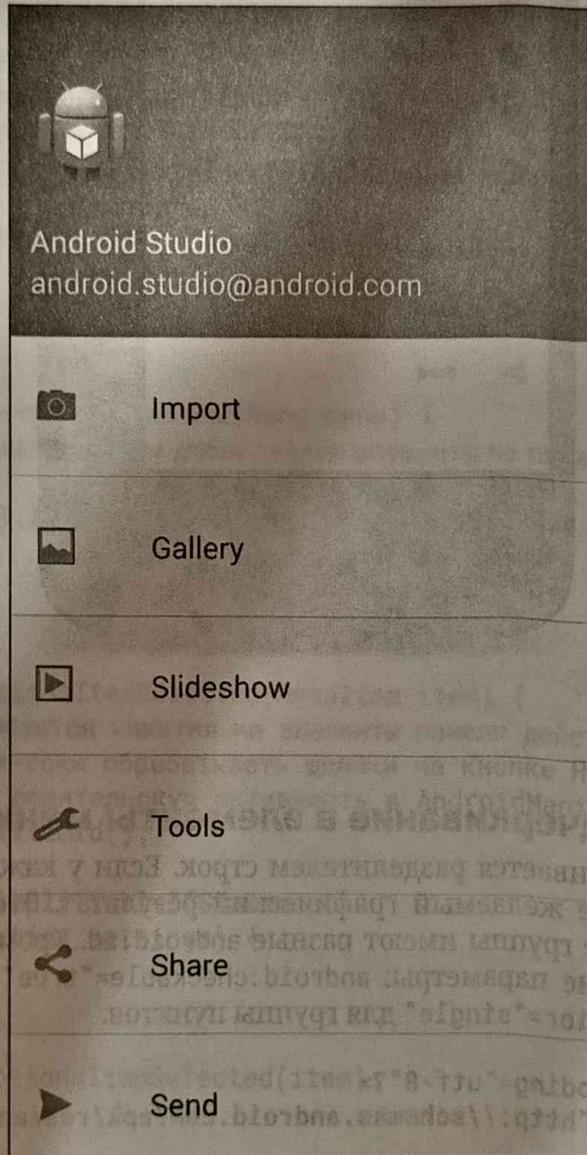
    <item
        android:id="@+id/pos_item_help"
        android:checkable="true"
        android:title="Help" />
    <item
        android:id="@+id/pos_item_pos"
        android:checkable="true"
        android:title="POS" />
    <item
        android:id="@+id/pos_item_orders"
        android:checkable="true"
        android:title="Orders" />
```

```

<group
    android:id="@+id/group"
    android:checkableBehavior="single">
    <item
        android:id="@+id/menu_nav_home"
        android:icon="@drawable/ic_home_black_24dp"
        android:title="@string/menu_nav_home" />
</group>

.....
</menu>

```



15.3. Добавление разделителей в меню

Обратитесь к RecyclerView в NavigationView и добавьте к нему ItemDecoration.

```

NavigationView navigationView = (NavigationView) findViewById(R.id.nav_view);
NavigationMenuView navMenuView = (NavigationMenuView) navigationView.getChildAt(0);
navMenuView.addItemDecoration(new DividerItemDecoration(this));

```

Код для DividerItemDecoration:

```

public class DividerItemDecoration extends RecyclerView.ItemDecoration {

```

```

private static final int[] ATTRS = new int[]{android.R.attr.listDivider};

private Drawable mDivider;

public DividerItemDecoration(Context context) {
    final TypedArray styledAttributes = context.obtainStyledAttributes(ATTRS);
    mDivider = styledAttributes.getDrawable(0);
    styledAttributes.recycle();
}

@Override
public void onDraw(Canvas c, RecyclerView parent, RecyclerView.State state) {
    int left = parent.getPaddingLeft();
    int right = parent.getWidth() - parent.getPaddingRight();

    int childCount = parent.getChildCount();
    for (int i = 1; i < childCount; i++) {
        View child = parent.getChildAt(i);

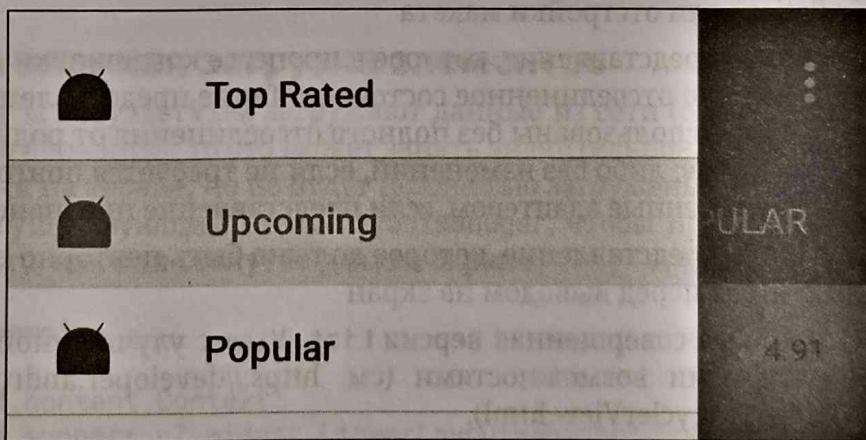
        RecyclerView.LayoutParams params = (RecyclerView.LayoutParams) child.
        getLayoutParams();

        int top = child.getBottom() + params.bottomMargin;
        int bottom = top + mDivider.getIntrinsicHeight();

        mDivider.setBounds(left, top, right, bottom); mDivider.draw(c);
    }
}
}

```

Вот как это выглядит:



15.4. Добавление разделителя меню с использованием стандартного DividerItemDecoration

Просто используйте стандартный класс DividerItemDecoration:

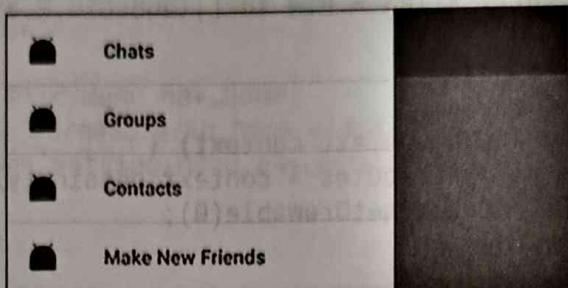
```

NavigationView navigationView = (NavigationView) findViewById(R.id.navigation);
NavigationMenuView navMenuView = (NavigationMenuView) navigationView.getChildAt(0);

navMenuView.addItemDecoration(new
    DividerItemDecoration(context, DividerItemDecoration.VERTICAL));

```

Предварительный просмотр:



Глава 16. RecyclerView

Параметр	Подробнее
Adapter	Подкласс RecyclerView.Adapter, отвечающий за предоставление представлений, которые представляют элементы в наборе данных
Position	Положение элемента данных в адаптере
Index	Индекс присоединенного дочернего представления, используемый в вызове getChildAt(int). Противоположность привязке к позиции
Binding	Процесс подготовки дочернего представления к отображению данных, соответствующих позиции в адаптере
Recycle (view)	Представление, ранее использовавшееся для отображения данных для определенной позиции адаптера, может быть помещено в кэш для последующего повторного использования для повторного отображения данных того же типа. Это может значительно повысить производительность за счет отсутствия первоначального «раздувания» (inflation) или отстройки макета
Scrap (view)	Дочернее представление, которое в процессе компоновки перешло во временно отсоединенное состояние. Такие представления могут быть повторно использованы без полного отсоединения от родительского RecyclerView: либо без изменений, если не требуется повторная привязка, либо измененные адаптером, если представление признано грязным (dirty)
Dirty (view)	Дочернее представление, которое должно быть повторно привязано адаптером перед выводом на экран

RecyclerView – это более совершенная версия List View с улучшенной производительностью и дополнительными возможностями (см. <https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html>).

16.1. Добавление RecyclerView

Добавьте зависимость, как описано в разделе Remark, а затем добавьте RecyclerView в свой макет:

```
<android.support.v7.widget.RecyclerView
    android:id="@+id/my_recycler_view"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
```

После добавления виджета RecyclerView в макет необходимо получить дескриптор к объекту, подключить его к менеджеру макетов и прикрепить адаптер для отображения данных:

```
mRecyclerView = (RecyclerView) findViewById(R.id.my_recycler_view);

// установить менеджер компоновки (в данном примере LinearLayoutManager)
mLayoutManager = new LinearLayoutManager(getApplicationContext());
mRecyclerView.setLayoutManager(mLayoutManager);

// указать адаптер
mAdapter = new MyAdapter(myDataset); mRecyclerView.setAdapter(mAdapter);
```

Или просто настройте менеджер компоновки из xml, добавив следующие строки:

```
xmlns:app="http://schemas.android.com/apk/res-auto"
app:layoutManager="android.support.v7.widget.LinearLayoutManager"
```

Если вы знаете, что изменение содержимого RecyclerView не приведет к изменению размера макета RecyclerView, то для повышения производительности компонента используйте следующий код. Если RecyclerView имеет фиксированный размер, то известно, что сам RecyclerView не будет изменять размеры за счет своих дочерних элементов, поэтому он вообще не вызывает запрос компоновки. Он просто обрабатывает изменение сам, если аннулировать то, что является родителем – координатор, макет или что-либо еще (этот метод можно использовать даже до установки LayoutManager и Adapter):

```
mRecyclerView.setHasFixedSize(true);
```

RecyclerView предоставляет для использования эти встроенные менеджеры компоновки. Таким образом, с его помощью можно создать список, сетку и ступенчатую сетку.

1. LinearLayoutManager отображает элементы в вертикальном или горизонтальном прокручиваемом списке.
2. GridLayoutManager отображает элементы в сетке.
3. StaggeredGridLayoutManager отображает элементы в сетке, расположенной в шахматном порядке.

16.2. Более плавная загрузка элементов

Если элементы RecyclerView загружают данные из сети (обычно изображения) или выполняют другую обработку, то это может занять значительное время, и в итоге элементы будут отображаться на экране, но не будут полностью загружены. Чтобы избежать этого, можно расширить существующий LinearLayoutManager, чтобы предварительно загружать ряд элементов до того, как они станут видны на экране:

```
package com.example;

import android.content.Context;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.OrientationHelper;
import android.support.v7.widget.RecyclerView;

/**
 * LinearLayoutManager, который предварительно загружает элементы за пределами экрана.
 * Предварительная загрузка полезна в ситуациях, когда загрузка элементов может занять
 * некоторое время. Обычно это связано с наличием в них карт, изображений
 * или других элементов, требующих завершения сетевые запросов, прежде чем
 * они будут отображены.
 * По умолчанию в этом макете загружается одна дополнительная страница,
 * страница - пиксельная мера, эквивалентная экранному размеру Recycler view
```

```

* Это можно изменить с помощью соответствующего конструктора
* или через метод {@link #setPages(int)}.
*/
public class PreLoadingLinearLayoutManager extends LinearLayoutManager {
    private int mPages = 1;
    private OrientationHelper mOrientationHelper;

    public PreLoadingLinearLayoutManager(final Context context) {
        super(context);
    }

    public PreLoadingLinearLayoutManager(final Context context, final int pages) {
        super(context);
        this.mPages = pages;
    }

    public PreLoadingLinearLayoutManager(final Context context, final int orientation, final boolean reverseLayout) {
        super(context, orientation, reverseLayout);
    }

    @Override
    public void setOrientation(final int orientation) {
        super.setOrientation(orientation);
        mOrientationHelper = null;
    }

    /**
     * Установите количество страниц макета, которые будут предварительно загружены
     * за пределами экрана,
     * страница - это пиксельная мера, эквивалентная экранному размеру
     * Recycler view.
     * @param pages - количество страниц; может быть {@code 0} для отключения
     * предварительной загрузки
     */
    public void setPages(final int pages) {
        this.mPages = pages;
    }

    @Override
    protected int getExtraLayoutSpace(final RecyclerView.State state) {
        if (mOrientationHelper == null) {
            mOrientationHelper = OrientationHelper.createOrientationHelper(this,
                getOrientation());
        }
        return mOrientationHelper.getTotalSpace() * mPages;
    }
}

```

16.3. RecyclerView с привязкой данных

Здесь представлен общий класс ViewHolder, который можно использовать с любым макетом DataBinding. Представлен экземпляр конкретного класса ViewDataBinding, который создается с использованием «раздутого» объекта View и класса утилиты DataBindingUtil.

```

import android.databinding.DataBindingUtil;
import android.support.v7.widget.RecyclerView;
import android.view.View;

```

```
public class BindingViewHolder<T> extends RecyclerView.ViewHolder {
    private final T binding;

    public BindingViewHolder(View itemView) {
        super(itemView);
        binding = (T) DataBindingUtil.bind(itemView);
    }

    public T getBinding() {
        return binding;
    }
}
```

После создания этого класса вы можете использовать **<layout>** в файле макета, чтобы включить привязку к базе данных для этого макета следующим образом:

file name: my_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">

    <data>
        <variable
            name="item"
            type="ItemModel" />
    </data>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:text="@{item.itemLabel}" />
    </LinearLayout>
</layout>
```

и вот ваш пример dataModel:

```
public class ItemModel {
    public String itemLabel;
}
```

По умолчанию библиотека Android Data Binding генерирует класс `ViewDataBinding` на основе имени файла макета, преобразуя его в регистр Pascal и добавляя к нему суффикс "Binding". В нашем примере это будет `MyItemBinding` для файла макета `my_item.xml`. Этот класс `Binding` также будет иметь setter-метод для установки объекта, определенного как данные в файле макета (`ItemModel` для данного примера).

Теперь, когда у нас есть все необходимые компоненты, мы можем реализовать наш адаптер следующим образом:

```
class MyAdapter extends RecyclerView.Adapter<BindingViewHolder<MyItemBinding>>{
    ArrayList<ItemModel> items = new ArrayList<>();

    public MyAdapter(ArrayList<ItemModel> items) {
        this.items = items;
    }

    @Override
    public BindingViewHolder<MyItemBinding> onCreateViewHolder(ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.my_item, parent, false);
        return new BindingViewHolder<MyItemBinding>(view);
    }

    @Override
    public void onBindViewHolder(BindingViewHolder<MyItemBinding> holder, int position) {
        MyItemBinding binding = holder.getBinding();
        binding.setItem(items.get(position));
    }

    @Override
    public int getItemCount() {
        return items.size();
    }
}
```

```

    }

    @Override public BindingViewHolder<MyItemBinding> onCreateViewHolder(ViewGroup
        parent, int viewType) {
        return new
        BindingViewHolder<>(LayoutInflater.from(parent.getContext()).inflate(R.layout.
        my_item, parent, false));
    }

    @Override public void onBindViewHolder(BindingViewHolder<ItemModel> holder, int
        position) {
        holder.getBinding().setItemModel(items.get(position));
        holder.getBinding().executePendingBindings();
    }

    @Override public int getItemCount() {
        return items.size();
    }
}

```

16.4. Анимация изменения данных

RecyclerView будет выполнять соответствующую анимацию, если используется любой из методов “notify”, кроме notifyDataSetChanged; к ним относятся notifyDataSetChanged, notifyItemInserted, notifyDataSetChanged, notifyDataSetChanged и т. д.

Адаптер должен расширять этот класс вместо RecyclerView.Adapter.

```

import android.support.annotation.NonNull;
import android.support.v7.widget.RecyclerView;

import java.util.List;

public abstract class AnimatedRecyclerAdapter<T, VH extends RecyclerView.
ViewHolder>
    extends RecyclerView.Adapter<VH> {
    protected List<T> models;
    protected AnimatedRecyclerAdapter(@NonNull List<T> models) {
        this.models = models;
    }

    //Установка новых моделей
    public void setModels(@NonNull final List<T> models) {
        applyAndAnimateRemovals(models);
        applyAndAnimateAdditions(models);
        applyAndAnimateMovedItems(models);
    }

    //Удаление элемента в позиции и уведомление об изменениях
    private T removeItem(int position) {
        final T model = models.remove(position);
        notifyItemRemoved(position);
        return model;
    }

    //Добавить элемент в позицию и уведомить об изменениях
    private void addItem(int position, T model) {
        models.add(position, model);
        notifyItemInserted(position);
    }
}

```

```

//Перемещение элемента из позиции fromPosition в позицию toPosition
и уведомление об изменениях
private void moveItem(int fromPosition, int toPosition) {
    final T model = models.remove(fromPosition);
    models.add(toPosition, model);
    notifyItemMoved(fromPosition, toPosition);
}

//Удаление элементов, которые больше не существуют в новых моделях
private void applyAndAnimateRemovals(@NonNull final List<T> newTs) {
    for (int i = models.size() - 1; i >= 0; i--) {
        final T model = models.get(i);
        if (!newTs.contains(model))
            removeItem(i);
    }
}

//Добавить элементы, которых нет в старых моделях
private void applyAndAnimateAdditions(@NonNull final List<T> newTs) {
    for (int i = 0, count = newTs.size(); i < count; i++) {
        final T model = newTs.get(i);
        if (!models.contains(model))
            addItem(i, model);
    }
}

//Перемещение элементов, изменивших свое положение
private void applyAndAnimateMovedItems(@NonNull final List<T> newTs) {
    for (int toPosition = newTs.size() - 1; toPosition >= 0; toPosition--) {
        final T model = newTs.get(toPosition);
        final int fromPosition = models.indexOf(model);
        if (fromPosition >= 0 && fromPosition != toPosition) {
            moveItem(fromPosition, toPosition);
        }
    }
}
}

```

НЕ СЛЕДУЕТ использовать один и тот же List для setModels и List в адаптере.

Вы объявляете модели (models) как глобальные переменные. DataModel – это только фиктивный класс.

```

private List<DataModel> models;
private YourAdapter adapter;

```

Инициализируйте модели перед передачей их адаптеру. YourAdapter – это реализация AnimatedRecyclerAdapter.

```

models = new ArrayList<>();
// Добавление моделей
models.add(new DataModel());
// Не передавайте модели напрямую, иначе при изменении глобальных моделей
// вы также будете модифицировать модели в адаптере.
// adapter = new YourAdapter(models); <- Это неправильно.
adapter = new YourAdapter(new ArrayList(models));

```

Вызовите эту функцию после обновления глобальных моделей.

```
adapter.setModels(new ArrayList(models));
```

Если вы не переопределите equals, то все сравнения будут выполняться по ссылке.

Пример с использованием SortedList

В Android вскоре после появления RecyclerView появился класс SortedList. Этот класс обрабатывает все вызовы метода 'notify' к RecyclerView.Adapter, чтобы обеспечить надлежащую анимацию, и даже позволяет пакетно обрабатывать несколько изменений, чтобы избежать дрожания анимации.

```
import android.support.v7.util.SortedList;
import android.support.v7.widget.RecyclerView;
import android.support.v7.widget.util.SortedListAdapterCallback;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import java.util.List;

public class MyAdapter extends RecyclerView.Adapter<MyAdapter.ViewHolder> {

    private SortedList<DataModel> mSortedList;

    class ViewHolder extends RecyclerView.ViewHolder {

        TextView text;
        CheckBox checkBox;

        ViewHolder(View itemView) {
            super(itemView);
            //Инициализируйте здесь свой код...
        }

        void setDataModel(DataModel model) {
            //Обновление пользовательского интерфейса с переданной сюда моделью
            //данных...
            text.setText(model.getText());
            checkBox.setChecked(model.isChecked());
        }
    }

    public MyAdapter() {
        mSortedList = new SortedList<>(DataModel.class, new
        SortedListAdapterCallback<DataModel>(this) {
            @Override
            public int compare(DataModel o1, DataModel o2) {
                //Это вызывается для определения порядка между объектами в массиве.
                if (o1.someValue() < o2.someValue()) {
                    return -1;
                } else if (o1.someValue() > o2.someValue()) {
                    return 1;
                } else {
                    return 0;
                }
            }
        });
    }

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext())
            .inflate(R.layout.item_view, parent, false);
        return new ViewHolder(view);
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int position) {
        DataModel model = mSortedList.get(position);
        holder.setDataModel(model);
    }

    @Override
    public int getItemCount() {
        return mSortedList.size();
    }
}
```

```
    }

    @Override
    public boolean areContentsTheSame(DataModel oldItem, DataModel newItem) {
        //Это нужно для того, чтобы проверить, изменилось ли содержимое данного
        //объекта. Эти элементы считаются одинаковыми только в том случае,
        //если функция areItemsTheSame() вернула true.

        //Если возвращается false, то вызывается функция onBindViewHolder()
        //с указанием держателя, содержащего элемент, и его позиции.
        return oldItem.getText().equals(newItem.getText()) && oldItem.
            isChecked() == newItem.isChecked();
    }

    @Override
    public boolean areItemsTheSame(DataModel item1, DataModel item2) {
        //Проверяется, не являются ли эти два элемента одинаковыми. Если нет,
        //то он добавляется в список, в противном случае проверяется,
        //не изменилось ли содержимое.
        return item1.equals(item2);
    }
}

@Override
public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    View itemView = //Иницируйте здесь представление вашего элемента.
    return new ViewHolder(itemView);
}

@Override
public void onBindViewHolder(ViewHolder holder, int position) {
    //Просто обновите держатель объектом из отсортированного списка с заданной
    //позиции
    DataModel model = mSortedList.get(position);
    if (model != null) {
        holder.setDataModel(model);
    }
}

@Override
public int getItemCount() {
    return mSortedList.size();
}

public void resetList(List<DataModel> models) {
    //Если вы выполняете несколько изменений, используйте методы пакетной обработки
    //для обеспечения надлежащей анимации.
    mSortedList.beginBatchedUpdates();
    mSortedList.clear();
    mSortedList.addAll(models);
    mSortedList.endBatchedUpdates();
}

//Следующие методы изменяют набор данных и автоматически обрабатывают вызов
//соответствующего метода 'notify' на адаптере.
public void addModel(DataModel model) {
    mSortedList.add(model);
}
```

```

public void addModels(List<DataModel> models) {
    mSortedList.addAll(models);
}

public void clear() {
    mSortedList.clear();
}

public void removeModel(DataModel model) {
    mSortedList.remove(model);
}

public void removeModelAt(int i) {
    mSortedList.removeItemAt(i);
}
}
}

```

16.5. Всплывающее меню с RecyclerView

Разместите нижеприведенный код в своем ViewHolder.

Примечание: в данном коде используется событие щелчка btnExpand, для всего события щелчка recyclerView можно установить слушателя (listener) на объект itemView.

```

public class MyViewHolder extends RecyclerView.ViewHolder{
    CardView cv;
    TextView recordName, visibleFile, date, time;
    Button btnIn, btnExpand;

    public MyViewHolder(final View itemView) {
        super(itemView);

        cv = (CardView)itemView.findViewById(R.id.cardview);
        recordName = (TextView)itemView.findViewById(R.id.tv_record);
        visibleFile = (TextView)itemView.findViewById(R.id.visible_file);
        date = (TextView)itemView.findViewById(R.id.date);
        time = (TextView)itemView.findViewById(R.id.time);
        btnIn = (Button)itemView.findViewById(R.id.btn_in_out);

        btnExpand = (Button) itemView.findViewById(R.id.btn_expand);

        btnExpand.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                PopupMenu popup = new PopupMenu(btnExpand.getContext(), itemView);

                popup.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
                    @Override
                    public boolean onMenuItemClick(MenuItem item) {
                        switch (item.getItemId()) {
                            case R.id.action_delete: moveFile(recordName.getText().toString(), getAdapterPosition());
                                return true;
                            case R.id.action_play:
                                String valueOfPath = recordName.getText().toString();
                                Intent intent = new Intent();
                                intent.setAction(android.content.Intent.ACTION_VIEW);

```

```
File file = new File(valueOfPath);
intent.setDataAndType(Uri.fromFile(file), "audio/*");
context.startActivity(intent);
return true;
case R.id.action_share:
    String valueOfPath = recordName.getText().toString();
    File filee = new File(valueOfPath);
    try {
        Intent sendIntent = new Intent();
        sendIntent.setAction(Intent.ACTION_SEND);
        sendIntent.setType("audio/*");
        sendIntent.putExtra(Intent.EXTRA_STREAM, Uri.
fromFile(filee));
        context.startActivity(sendIntent);
    } catch (NoSuchMethodError | IllegalArgumentException |
NullPointerException e) {
        e.printStackTrace();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return true;
default:
    return false;
}
});
// здесь вы можете "раздуть" свое меню
popup.inflate(R.menu.my_menu_item);
popup.setGravity(Gravity.RIGHT);
// если вы хотите получить иконку с пунктами меню, то напишите этот блок
try-catch.
try {
    Field mFieldPopup=popup.getClass().getDeclaredField("mPopup");
    mFieldPopup.setAccessible(true);
    MenuPopupHelper mPopup = (MenuPopupHelper) mFieldPopup.get(popup);
    mPopup.setForceShowIcon(true);
} catch (Exception e) {
}
popup.show();
}
});
```

Альтернативный способ отображения иконок в меню:

```
try {
    Field[] fields = popup.getClass().getDeclaredFields();
    for (Field field : fields) {
        if ("mPopup".equals(field.getName())) {
            field.setAccessible(true);
            Object menuPopupHelper = field.get(popup);
            Class<?> classPopupHelper = Class.forName(menuPopupHelper.getClass().
                getName());
```

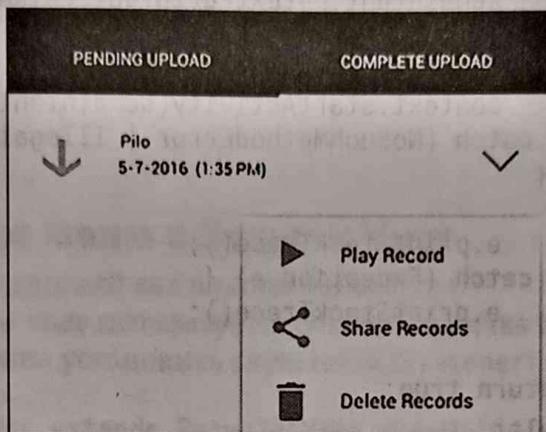
```

        Method setForceIcons = classPopupHelper.getMethod("setForceShowIcon",
        boolean.class);
        setForceIcons.invoke(menuPopupHelper, true);
        break;
    }
}
} catch (Exception e) {

}

```

Ниже приведен результат:



16.6. Использование нескольких ViewHolders с ItemType

Иногда в RecyclerView необходимо использовать несколько типов представлений для отображения в списке, показываемом в пользовательском интерфейсе, и для каждого представления требуется свой layout xml для «раздувания».

Для решения этой проблемы можно использовать различные ViewHolders в одном адаптере, используя специальный метод в RecyclerView – `getItemViewType(int position)`.

Ниже приведен пример использования двух ViewHolders.

1. ViewHolder для отображения элементов списка.
2. ViewHolder для отображения нескольких представлений заголовков.

```

@Override
public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    View itemView = LayoutInflater.from(context).inflate(viewType, parent, false);
    return ViewHolder.create(itemView, viewType);
}

@Override
public void onBindViewHolder(RecyclerView.ViewHolder holder, int position) {
    final Item model = this.items.get(position);
    ((ViewHolder) holder).bind(model);
}

@Override
public int getItemViewType(int position) {
    return inSearchState ? R.layout.item_header : R.layout.item_entry;
}

abstract class ViewHolder {
    abstract void bind(Item model);
}

public static ViewHolder create(View v, int viewType) {

```

```

        return viewType == R.layout.item_header ? new HeaderViewHolder(v) :new
        EntryViewHolder(v);
    }

}

static class EntryViewHolder extends ViewHolder {
    private View v;

    public EntryViewHolder(View v) {
        this.v = v;
    }

    @Override public void bind(Item model) {
        // Привязка данных элемента к представлению ввода
    }
}

static class HeaderViewHolder extends ViewHolder {
    private View v;

    public HeaderViewHolder(View v) {
        this.v = v;
    }

    @Override public void bind(Item model) {
        // Привязка данных элемента к представлению заголовка.
    }
}

```

16.7. Фильтрация элементов внутри RecyclerView с помощью SearchView

Добавим метод фильтрации в RecyclerView.Adapter:

```

public void filter(String text) {
    if(text.isEmpty()){
        items.clear();
        items.addAll(itemsCopy);
    } else{
        ArrayList<PhoneBookItem> result = new ArrayList<>();
        text = text.toLowerCase();
        for(PhoneBookItem item: itemsCopy){
            //сопоставление по имени или телефону
            if(item.name.toLowerCase().contains(text) ||
item.phone.toLowerCase().contains(text)){
                result.add(item);
            }
        }
        notifyDataSetChanged();
    }
}

```

itemsCopy инициализируется в конструкторе адаптера по типу itemsCopy.addAll(items). В этом случае достаточно вызвать фильтр из OnQueryTextListener из SearchView:

```
searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
```

```

@Override
public boolean onQueryTextSubmit(String query) {
    adapter.filter(query);
    return true;
}

@Override
public boolean onQueryTextChange(String newText) {
    adapter.filter(newText);
    return true;
}
);

```

16.8. Перетаскивание и пролистывание с помощью RecyclerView

Функции пролистывания и перетаскивания можно реализовать с помощью RecyclerView без использования сторонних библиотек.

Достаточно воспользоваться классом `ItemTouchHelper`, входящим в состав библиотеки поддержки RecyclerView.

Инициализируйте `ItemTouchHelper` с обратным вызовом `SimpleCallback` и, в зависимости от того, какую функциональность вы поддерживаете, переопределите `onMove(RecyclerView, ViewHolder, ViewHolder)` и/или `onSwiped(ViewHolder, int)` и, наконец, прикрепите к вашему RecyclerView.

```

ItemTouchHelper.SimpleCallback simpleItemTouchCallback = new ItemTouchHelper.SimpleCallback(0, ItemTouchHelper.LEFT | ItemTouchHelper.RIGHT) {

    @Override
    public void onSwiped(RecyclerView.ViewHolder viewHolder, int swipeDir) {
        // удалить элемент из адаптера
    }

    @Override
    public boolean onMove(RecyclerView recyclerView, RecyclerView.ViewHolder viewHolder, RecyclerView.ViewHolder target) {
        final int fromPos = viewHolder.getAdapterPosition();
        final int toPos = target.getAdapterPosition();
        // перемещение элемента `fromPos` в `toPos` в адаптере.
        return true; // true, если перемещение произошло, false в противном случае
    }
};

ItemTouchHelper itemTouchHelper = new ItemTouchHelper(simpleItemTouchCallback);
itemTouchHelper.attachToRecyclerView(recyclerView);

```

Следует отметить, что конструктор `SimpleCallback` применяет одну и ту же стратегию пролистывания ко всем элементам RecyclerView. В любом случае можно обновить направление пролистывания по умолчанию для конкретных элементов, просто переопределив метод `getSwipeDirs(RecyclerView, ViewHolder)`.

Предположим, например, что наш RecyclerView содержит `HeaderViewHolder`, и мы явно не хотим применять к нему пролистывание. Достаточно переопределить `getSwipeDirs` следующим образом:

```

@Override
public int getSwipeDirs(RecyclerView recyclerView, RecyclerView.ViewHolder viewHolder) {
    if (viewHolder instanceof HeaderViewHolder) {

```

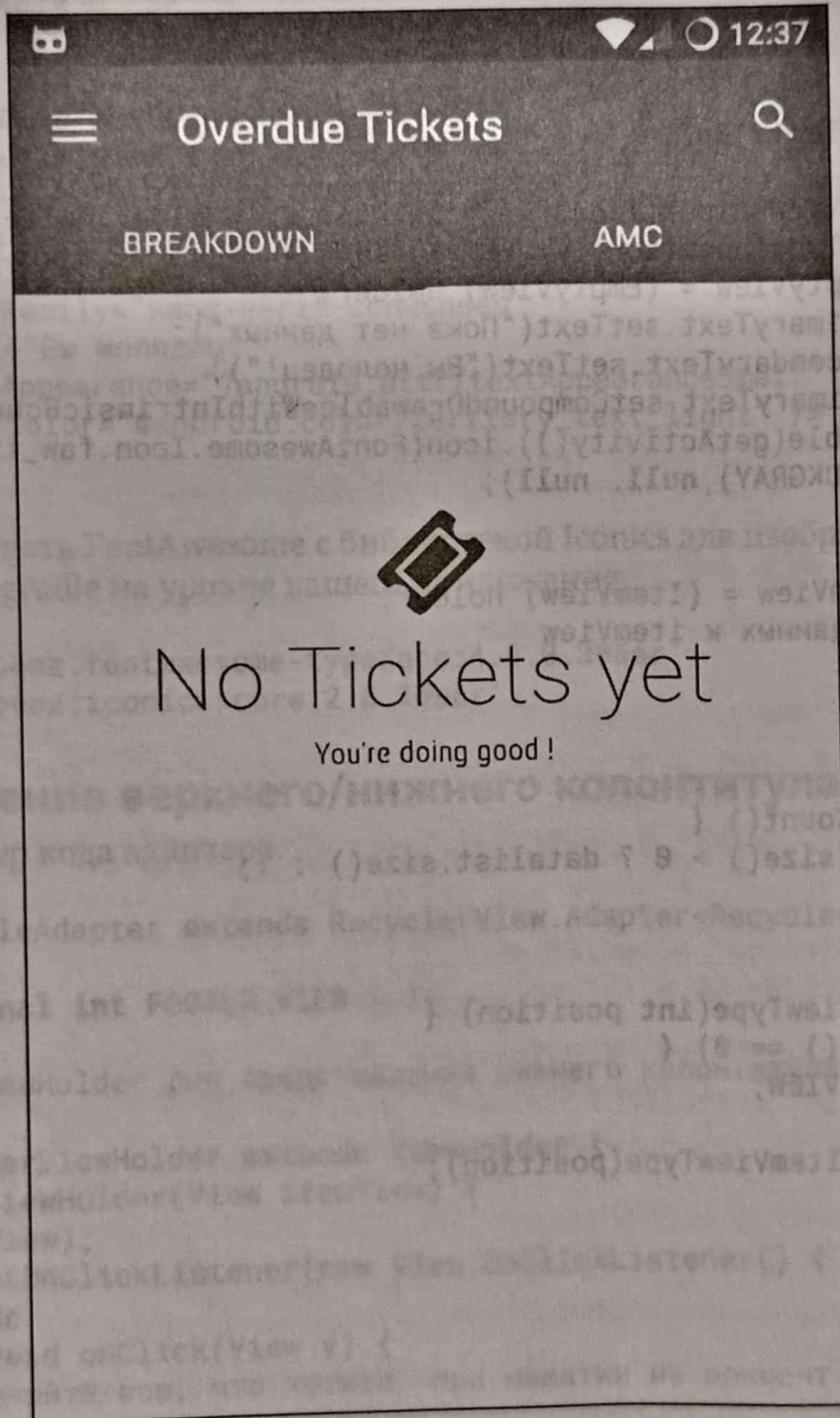
```

    // без пролистывания заголовка
    return 0;
}
// пролистывание по умолчанию для всех остальных элементов
return super.getSwipeDirs(recyclerView, viewHolder);
}
}

```

16.9. Как показывать представление по умолчанию до загрузки элементов или когда данные недоступны

Скриншот:



Класс адаптера

```

private class MyAdapter extends RecyclerView.Adapter<RecyclerView.ViewHolder> {
    final int EMPTY_VIEW = 77777;
    List<CustomData> dataList = new ArrayList<>();
}

```

```

MyAdapter() {
    super();
}

@Override
public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {

    LayoutInflator layoutInflater = LayoutInflator.from(parent.getContext());
    if (viewType == EMPTY_VIEW) {
        return new EmptyView(layoutInflater.inflate(R.layout.nothing_yet, parent,
            false));
    } else {
        return new ItemView(layoutInflater.inflate(R.layout.my_item, parent, false));
    }
}

@SuppressLint("SetTextI18n")
@Override
public void onBindViewHolder(final RecyclerView.ViewHolder holder, int position) {
    if (getItemViewType(position) == EMPTY_VIEW) {
        EmptyView emptyView = (EmptyView) holder;
        emptyView.primaryText.setText("Пока нет данных");
        emptyView.secondaryText.setText("Вы молодец!");
        emptyView.primaryText.setCompoundDrawablesWithIntrinsicBounds(null, new
            IconicsDrawable(getActivity()).icon(FontAwesome.Icon.faw_ticket).sizeDp(48),
            null, null);
    } else {
        ItemView itemView = (ItemView) holder;
        // Привязка данных к itemView
    }
}

@Override
public int getItemCount() {
    return datalist.size() > 0 ? datalist.size() : 1;
}

@Override
public int getItemViewType(int position) {
    if (datalist.size() == 0) {
        return EMPTY_VIEW;
    }
    return super.getItemViewType(position);
}
}

```

nothing_yet.xml

```

<?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:layout_gravity="center"
    android:orientation="vertical"
    android:paddingBottom="100dp"
    android:paddingTop="100dp">

```

```

<TextView
    android:id="@+id/nothingPrimary"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:drawableTint="@android:color/secondary_text_light"
    android:drawableTop="@drawable/ic_folder_open_black_24dp"
    android:enabled="false"
    android:fontFamily="sans-serif-light"
    android:text="Пока нет данных"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:textColor="@android:color/secondary_text_light"
    android:textSize="40sp"
    tools:targetApi="m" />

<TextView
    android:id="@+id/nothingSecondary"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:enabled="false"
    android:fontFamily="sans-serif-condensed"
    android:text="Вы молодец!"
    android:textAppearance="?android:attr/textAppearanceSmall"
    android:textColor="@android:color/tertiary_text_light" />
</LinearLayout>

```

Можно использовать FontAwesome с библиотекой Iconics для изображений. Добавьте эти строки в файл build.gradle на уровне вашего приложения:

```
compile 'com.mikepenz:fontawesome-typeface:4.6.0.3@aar'
compile 'com.mikepenz:iconics-core:2.8.1@aar'
```

16.10. Добавление верхнего/нижнего колонтитула в RecyclerView

Приведем пример кода адаптера.

```

public class SampleAdapter extends RecyclerView.Adapter<RecyclerView.ViewHolder> {
    private static final int FOOTER_VIEW = 1;

    // Определение ViewHolder для представления нижнего колонтитула (Footer)

    public class FooterViewHolder extends ViewHolder {
        public FooterViewHolder(View itemView) {
            super(itemView);
            itemView.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    // Делайте все, что хотите, при нажатии на элемент
                }
            });
        }
    }

    // Теперь определим объект просмотра для элемента списка Normal
    public class NormalViewHolder extends ViewHolder {
        public NormalViewHolder(View itemView) {
            super(itemView);
        }
    }
}

```

```

    itemView.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // Делайте все, что хотите, при нажатии на обычные элементы
        }
    });
}

// И теперь в onCreateViewHolder нужно передать правильное представление
// при заполнении элемента списка
@Override
public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    View v;

    if (viewType == FOOTER_VIEW) {
        v = LayoutInflater.from(parent.getContext()).inflate(R.layout.list_item_footer, parent, false);

        FooterViewHolder vh = new FooterViewHolder(v);
        return vh;
    }

    v = LayoutInflater.from(parent.getContext()).inflate(R.layout.list_item_normal, parent, false);

    NormalViewHolder vh = new NormalViewHolder(v);
    return vh;
}

// Теперь привязываем держатели представлений в onBindViewHolder
@Override
public void onBindViewHolder(RecyclerView.ViewHolder holder, int position) {
    try {
        if (holder instanceof NormalViewHolder) {
            NormalViewHolder vh = (NormalViewHolder) holder;

            vh.bindView(position);
        } else if (holder instanceof FooterViewHolder) {
            FooterViewHolder vh = (FooterViewHolder) holder;
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

// Теперь самая важная часть. Вы должны вернуть точное количество
// элементов вашего списка
// В примере только один нижний колонтитул (footer). Поэтому вернем data.size() + 1
// Если у вас несколько верхних и нижних колонтитулов, то вы должны
// вернуть общее количество,
// например, headers.size() + data.size() + footers.size()

@Override
public int getItemCount() {
}

```

```

if (data == null) {
    return 0;
}

if (data.size() == 0) {
    // Возвращаем 1, чтобы ничего не показывать
    return 1;
}

// Добавить дополнительный вид для отображения нижнего колонтитула
return data.size() + 1;
}

// Теперь определите свой собственный getItemViewType.
@Override
public int getItemViewType(int position) {
    if (position == data.size()) {
        // Здесь мы добавим нижний колонтитул.
        return FOOTER_VIEW;
    }

    return super.getItemViewType(position);
}

// На этом добавление колонтитула и его действие на onClick завершено.
// Теперь установим по умолчанию ViewHolder для NormalViewHolder

public class ViewHolder extends RecyclerView.ViewHolder {
    // Определяем здесь элементы ряда
    public ViewHolder(View itemView) {
        super(itemView);
        // Найти представление по идентификатору и инициализировать здесь
    }

    public void bindView(int position) {
        // метод bindView() для реализации действий
    }
}
}

```

О реализации RecyclerView с верхним и нижним колонтитулами можно прочитать здесь: <https://plus.google.com/+WillBlaschko/posts/3MFmgPbQuWx>.

Альтернативный метод

Хотя приведенный выше код будет работать, можно использовать и подход с применением RecyclerView с помощью NestedScrollView. Добавить макет для заголовка можно следующим образом:

```

<android.support.v4.widget.NestedScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <include
            layout="@layout/drawer_view_header">

```

```

        android:id="@+id/navigation_header"/>

    <android.support.v7.widget.RecyclerView
        android:layout_below="@+id/navigation_header"
        android:id="@+id/followers_list"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>

</RelativeLayout>
</android.support.v4.widget.NestedScrollView>

```

Также можно использовать LinearLayout с вертикальным выравниванием в вашем NestedScrollView.

Примечание: это будет работать только с RecyclerView версии 23.2.0.

```
compile 'com.android.support:recyclerview-v7:23.2.0'
```

16.11. Бесконечная прокрутка в RecyclerView

Здесь приводится фрагмент кода для реализации бесконечной прокрутки в RecyclerView.

Шаг 1. Сначала создайте в адаптере RecyclerView один абстрактный метод, как показано ниже:

```

public abstract class ViewAllCategoryAdapter extends RecyclerView.Adapter<RecyclerView.ViewHolder> {
    public abstract void load();
}

```

Шаг 2. Теперь переопределите onBindViewHolder и метод getItemCount() класса ViewAllCategoryAdapter и вызовите метод Load(), как показано ниже:

```

@Override
public void onBindViewHolder(RecyclerView.ViewHolder holder, final int position) {
    if ((position >= getItemCount() - 1)) {
        load();
    }
}

@Override
public int getItemCount() {
    return YOURLIST.size();
}

```

Шаг 3. Теперь все логические операции бэкенда завершены, и пришло время их выполнить. Это просто: переопределите метод load, в котором создается объект вашего адаптера.

Этот метод автоматически вызывается, когда пользователь достигает конца списка.

```

adapter = new ViewAllCategoryAdapter(CONTEXT, YOURLIST) {
    @Override
    public void load() {
        /* ваш код здесь */
        /* Этот метод будет автоматически вызываться, когда пользователь дойдет
        до конца списка. */
    }
}

```

```

};

recycleCategory.setAdapter(adapter);

```

Теперь метод `load()` автоматически вызывается при прокрутке пользователем до конца списка.

16.12. Добавление разделительных линий в элементы RecyclerView

Просто добавьте эти строки в инициализацию:

```

RecyclerView mRecyclerView = (RecyclerView) view.findViewById(recyclerView);
mRecyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));
mRecyclerView.addItemDecoration(new DividerItemDecoration(getActivity(),
DividerItemDecoration.VERTICAL));

```

Добавьте адаптер и вызовите `.notifyDataSetChanged()`, как обычно.

Это не встроенная функция `RecyclerView`, а добавленная в библиотеки поддержки. Поэтому не забудьте включить это в файл `build.gradle` на уровне вашего приложения:

```
compile "com.android.support:appcompat-v7:25.3.1" compile "com.android.
support:recyclerview-v7:25.3.1"
```

В один `RecyclerView` может быть добавлено несколько декоров `ItemDecorations`.

Изменение цвета разделителя:

Задать цвет для элемента `Decoration` довольно просто.

1-й шаг: создание файла `divider.xml`, который находится в папке `drawable`.

```

<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="line">
    <size
        android:width="1px"
        android:height="1px"/>
    <solid android:color="@color/divider_color"/>
</shape>

```

2-й шаг: установка `drawable`.

```

// Получить объект drawable
Drawable mDivider = ContextCompat.getDrawable(m_jContext, R.drawable.divider);
// Создаем декор DividerItemDecoration с горизонтальной ориентацией
DividerItemDecoration hItemDecoration = new DividerItemDecoration(m_jContext,
DividerItemDecoration.HORIZONTAL);
// Установите на него drawable
hItemDecoration.setDrawable(mDivider);

```

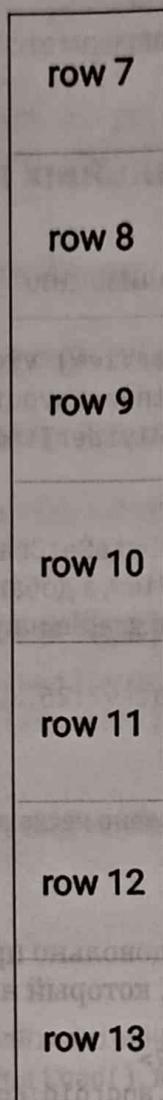
large colum n 0	colum n 1	colum n 2	large colum n 3	colum n 4	col n
-----------------------	--------------	--------------	-----------------------	--------------	----------

```

// Создать декор DividerItemDecoration с вертикальной ориентацией
DividerItemDecoration vItemDecoration = new DividerItemDecoration(m_jContext,
DividerItemDecoration.VERTICAL);

```

```
// Установите на него drawable
vItemDecoration.setDrawable(mDivider);
```



Глава 17. Украшения RecyclerView

Параметр	Подробности
decoration	украшение элемента для добавления в RecyclerView
index	индекс в списке украшений для данного RecyclerView. Это порядок, в котором вызываются getItemOffset и onDraw. Последующие вызовы могут быть по приоритету выше предыдущих.

17.1. Добавление разделителя в RecyclerView

Прежде всего необходимо создать класс, расширяющий RecyclerView.ItemDecoration:

```
public class SimpleBlueDivider extends RecyclerView.ItemDecoration {
    private Drawable mDivider;

    public SimpleBlueDivider(Context context) {
        mDivider = context.getResources().getDrawable(R.drawable.divider_blue);
    }

    @Override
    public void onDraw(Canvas c, RecyclerView parent, RecyclerView.State state) {
        int childCount = parent.getChildCount();
        for (int i = 0; i < childCount; i++) {
            View child = parent.getChildAt(i);
            if (child.getBottom() > 0) {
                int left = child.getRight() + getLeftMargin(parent);
                int right = child.getRight() + getRightMargin(parent);
                int top = child.getBottom() + getTopMargin(parent);
                int bottom = child.getBottom() + getBottomMargin(parent);
                mDivider.setBounds(left, top, right, bottom);
                mDivider.draw(c);
            }
        }
    }

    @Override
    public void getItemOffsets(Rect outRect, View view, RecyclerView parent, RecyclerView.State state) {
        outRect.set(0, 0, 0, 0);
    }
}
```

```
@Override  
public void onDrawOver(Canvas c, RecyclerView parent, RecyclerView.State state) {  
    //divider padding дать отступ, какой хотите, или отключить  
    int left = parent.getPaddingLeft() + 80;  
    int right = parent.getWidth() - parent.getPaddingRight() - 30;  
  
    int childCount = parent.getChildCount();  
    for (int i = 0; i < childCount; i++) {  
        View child = parent.getChildAt(i);  
  
        RecyclerView.LayoutParams params = (RecyclerView.LayoutParams) child.  
            getLayoutParams();  
  
        int top = child.getBottom() + params.bottomMargin;  
        int bottom = top + mDivider.getIntrinsicHeight();  
  
        mDivider.setBounds(left, top, right, bottom);  
        mDivider.draw(c);  
    }  
}
```

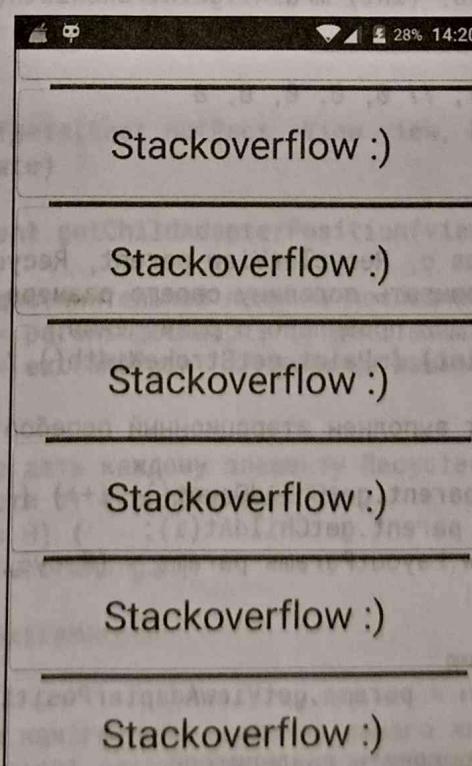
Добавьте файл divider_blue.xml в папку drawable:

```
<?xml version="1.0" encoding="utf-8"?>  
<shape xmlns:android="http://schemas.android.com/apk/res/android"  
    android:shape="rectangle">  
    <size android:width="1dp" android:height="4dp" />  
    <solid android:color="#AA123456" />  
</shape>
```

Затем используйте его как:

```
recyclerView.addItemDecoration(new SimpleBlueDivider(context));
```

Результат будет иметь вид:



Это изображение – просто пример работы разделителей, если вы хотите следовать спецификациям материального оформления Material Design при добавлении разделителей, пожалуйста, посмотрите этот ресурс: <https://material.google.com/components/dividers.html>; спасибо @Brenden Kromhout за предоставленную ссылку.

17.2. Отрисовывание сепаратора

При применении следующего кода внизу каждого вида, кроме последнего, будет проведена линия, служащая разделителем между элементами.

```
public class SeparatorDecoration extends RecyclerView.ItemDecoration {  
  
    private final Paint mPaint;  
    private final int mAlpha;  
  
    public SeparatorDecoration(@ColorInt int color, float width) {  
        mPaint = new Paint();  
        mPaint.setColor(color);  
        mPaint.setStrokeWidth(width);  
        mAlpha = mPaint.getAlpha();  
    }  
  
    @Override  
    public void getItemOffsets(Rect outRect, View view, RecyclerView parent,  
        RecyclerView.State state) {  
        final RecyclerView.LayoutParams params = (RecyclerView.LayoutParams) view.  
            getLayoutParams();  
  
        // мы получаем позицию в списке  
        final int position = params.getViewAdapterPosition();  
  
        // добавить место для разделителя в нижнюю часть каждого вида, кроме  
        // последнего  
        if (position < state.getItemCount()) {  
            outRect.set(0, 0, 0, (int) mPaint.getStrokeWidth()); // слева, сверху,  
            // справа, снизу  
        } else {  
            outRect.setEmpty(); // 0, 0, 0, 0  
        }  
    }  
  
    @Override  
    public void onDraw(Canvas c, RecyclerView parent, RecyclerView.State state) {  
        // линия будет отрисовывать половину своего размера сверху и снизу,  
        // отсюда и смещение для правильного размещения  
        final int offset = (int) (mPaint.getStrokeWidth() / 2);  
  
        // в результате будет выполнен итерационный перебор всех видимых  
        // представлений  
        for (int i = 0; i < parent.getChildCount(); i++) {  
            final View view = parent.getChildAt(i);  
            final RecyclerView.LayoutParams params = (RecyclerView.LayoutParams) view.  
                getLayoutParams();  
  
            // получить позицию  
            final int position = params.getViewAdapterPosition();  
  
            // и, наконец, отрисовать разделитель
```

```
if (position < state.getItemCount()) {
    // применение альфа-функции для поддержки анимации
    mPaint.setAlpha((int) (view.getAlpha() * mAlpha));

    float positionY = view.getBottom() + offset + view.getTranslationY();
    // выполнить отрисовку
    c.drawLine(view.getLeft() + view.getTranslationX(), positionY,
              view.getRight() + view.getTranslationX(), positionY,
              mPaint);
}
}
```

17.3. Как добавить разделители с помощью DividerItemDecoration

Декор DividerItemDecoration относится к RecyclerView. Это ItemDecoration, который может использоваться в качестве разделителя между элементами.

```
DividerItemDecoration mDividerItemDecoration = new DividerItemDecoration(context,  
mLayoutManager.getOrientation());  
recyclerView.addItemDecoration(mDividerItemDecoration);
```

Он поддерживает два варианта ориентации, используя `DividerItemDecoration.VERTICAL` и `DividerItemDecoration.HORIZONTAL`.

17.4. Отступы для каждого элемента с помощью ItemDecoration

Вы можете использовать `RecyclerView.ItemDecoration` для установки дополнительных полей вокруг каждого элемента в `RecyclerView`. В некоторых случаях это позволяет очистить как реализацию адаптера, так и XML-представления элементов.

```
public class MyItemDecoration  
    extends RecyclerView.ItemDecoration {  
  
    private final int extraMargin;  
  
    @Override  
    public void getItemOffsets(Rect outRect, View view, RecyclerView parent,  
        RecyclerView.State state) {  
  
        int position = parent.getChildAdapterPosition(view);  
  
        // Легко наложить дополнительное поле на последний элемент...  
        if (position + 1 == parent.getAdapter().getItemCount()) {  
            outRect.bottom = extraMargin; // единица измерения - px  
        }  
  
        // ...или вы можете дать каждому элементу RecyclerView разные  
        // поля в зависимости от его положения...  
        if (position % 2 == 0) {  
            outRect.right = extraMargin;  
        } else {  
            outRect.left = extraMargin;  
        }  
  
        // ...или на основе какого-либо свойства самого элемента  
        MyListItem item = parent.getAdapter().getItem(position);
```

```

    if (item.isFirstItemInSection()) {
        outRect.top = extraMargin;
    }
}

public MyItemDecoration(Context context) {
    extraMargin = context.getResources()
        .getDimensionPixelOffset(R.dimen.extra_margin);
}

```

Чтобы включить украшение (decoration), просто добавьте его в RecyclerView:

```
// в onCreate()
RecyclerView rv = (RecyclerView) findViewById(R.id myList);
rv.addItemDecoration(new MyItemDecoration(context));
```

17.5. ItemOffsetDecoration для GridLayoutManager в RecyclerView

Следующий пример поможет задать равное пространство для элемента в GridLayout.

ItemOffsetDecoration.java

```

public class ItemOffsetDecoration extends RecyclerView.ItemDecoration {

    private int mItemOffset;

    private int spanCount = 2;

    public ItemOffsetDecoration(int itemOffset) {
        mItemOffset = itemOffset;
    }

    public ItemOffsetDecoration(@NonNull Context context, @DimenRes int
        itemOffsetId) {
        this(context.getResources().getDimensionPixelSize(itemOffsetId));
    }

    @Override
    public void getItemOffsets(Rect outRect, View view, RecyclerView parent,
        RecyclerView.State state) {
        super.getItemOffsets(outRect, view, parent, state);

        int position = parent.getChildLayoutPosition(view);

        GridLayoutManager manager = (GridLayoutManager) parent.getLayoutManager();

        if (position < manager.getSpanCount())
            outRect.top = mItemOffset;

        if (position % 2 != 0) {
            outRect.right = mItemOffset;
        }

        outRect.left = mItemOffset;
        outRect.bottom = mItemOffset;
    }
}

```

Вы можете вызвать ItemDecoration следующим образом:

```
recyclerView = (RecyclerView) view.findViewById(R.id.recycler_view);
GridLayoutManager lLayout = new GridLayoutManager(getActivity(), 2);
ItemOffsetDecoration itemDecoration = new ItemOffsetDecoration(mActivity, R.dimen.item_offset); recyclerView.addItemDecoration(itemDecoration);
recyclerView.setLayoutManager(lLayout);
```

и пример смещения элемента:

```
<dimen name="item_offset">5dp</dimen>
```

Глава 18. Слушатели onClickListeners в RecyclerView

18.1. Пример на Kotlin и RxJava

Первый пример переделан на Kotlin и использует RxJava для более чистого взаимодействия:

```
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.support.v7.widget.RecyclerView
import rx.subjects.PublishSubject

public class SampleAdapter(private val items: Array<String>) : RecyclerView.Adapter<SampleAdapter.ViewHolder>() {

    // для получения другого поведения меняйте субъекты из rx.subjects
    // BehaviorSubject, например, позволяет получать последнее событие по подписке
    // PublishSubject отправляет события только после подписки, что, с другой стороны,
    // желательно для кликов
    public val itemClickStream: PublishSubject<View> = PublishSubject.create()

    override fun getItemCount(): Int {
        return items.size
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder? {
        val v = LayoutInflater.from(parent.getContext()).inflate(R.layout.text_row_item, parent, false);
        return ViewHolder(view)
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        holder.bind(items[position])
    }

    public inner class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {
        private val textView: TextView by lazy { view.findViewById(R.id.textView) as TextView }

        init {
```

```
        view.setOnClickListener { v -> itemClickStream.onNext(v) }
    }

    fun bind(text: String) {
        textView.text = text
    }
}
```

В этом случае использование достаточно простое. Можно подписаться на отдельный поток, используя средства RxJava.

```
val adapter = SampleAdapter(arrayOf("Hello", "World"))
adapter.itemClickStream.subscribe { v ->
    if (v.id == R.id.textView) {
        // сделать что-нибудь
    }
}
```

18.2. Слушатель кликов Recycler View

```
public class RecyclerTouchListener implements RecyclerView.OnItemTouchListener {
    private GestureDetector gestureDetector;
    private RecyclerTouchListener.ClickListener clickListener;

    public RecyclerTouchListener(Context context, final RecyclerView recyclerView,
        final RecyclerTouchListener.ClickListener clickListener) {
        this.clickListener = clickListener;
        gestureDetector = new GestureDetector(context, new GestureDetector.SimpleOnGestureListener() {
            @Override
            public boolean onSingleTapUp(MotionEvent e) {
                return true;
            }
            @Override
            public void onLongPress(MotionEvent e) {
                View child = recyclerView.findChildViewUnder(e.getX(), e.getY());
                if (child != null & clickListener != null) {
                    clickListener.onLongClick(child, recyclerView.getChildPosition(child));
                }
            }
        });
    }

    @Override
    public boolean onInterceptTouchEvent(RecyclerView rv, MotionEvent e) {
        View child = rv.findChildViewUnder(e.getX(), e.getY());
        if (child != null & clickListener != null & gestureDetector.
            onTouchEvent(e)) {
            clickListener.onClick(child, rv.getChildPosition(child));
        }
        return false;
    }

    @Override
    public void onTouchEvent(RecyclerView rv, MotionEvent e) {
```

```

    }

    @Override
    public void onRequestDisallowInterceptTouchEvent(boolean disallowIntercept) {
    }

    public interface ClickListener {
        void onLongClick(View child, int childPosition);
        void onClick(View child, int childPosition);
    }
}

B MainActivity:
RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recyclerView);
recyclerView.addOnItemTouchListener(new
RecyclerTouchListener(getApplicationContext(),recyclerView, new RecyclerTouchListener.
ClickListener() {
    @Override
    public void onLongClick(View child, int childPosition) {
    }

    @Override
    public void onClick(View child, int childPosition) {
    }
})));

```

18.3. Другой способ реализации слушателя клика элемента

Другой способ реализации заключается в использовании интерфейса с несколькими методами, число которых равно числу кликабельных представлений, и использовании определенных слушателей (прослушивателей) клика, как показано ниже. Этот способ является более гибким, поскольку можно назначить слушателей кликов на разные представления и легко управлять логикой клика отдельно для каждого из них.

```

public class CustomAdapter extends RecyclerView.Adapter<CustomAdapter.CustomHolder> {

    private ArrayList<Object> mObjects;
    private ClickInterface mClickInterface;

    public interface ClickInterface {
        void clickEventOne(Object obj);
        void clickEventTwo(Object obj1, Object obj2);
    }

    public void setClickInterface(ClickInterface clickInterface) {
        mClickInterface = clickInterface;
    }

    public CustomAdapter(){
        mList = new ArrayList<>();
    }

    public void addItems(ArrayList<Object> objects) {
        mObjects.clear();
    }
}

```

```
mObjects.addAll(objects);
notifyDataSetChanged();
}

@Override
public CustomHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    View v = LayoutInflater.from(parent.getContext())
        .inflate(R.layout.list_item, parent, false);
    return new CustomHolder(v);
}

@Override
public void onBindViewHolder(CustomHolder holder, int position) {
    //сделать все четные позиции некликабельными
    holder.firstClickListener.setClickable(position%2==0);
    holder.firstClickListener.setPosition(position);
    holder.secondClickListener.setPosition(position);
}

private class FirstClickListener implements View.OnClickListener {
    private int mPosition;
    private boolean mClickable;
    void setPosition(int position) {
        mPosition = position;
    }

    void setClickable(boolean clickable) {
        mPosition = position;
    }

    @Override
    public void onClick(View v) {
        if(mClickable) {
            mClickInterface.clickEventOne(mObjects.get(mPosition));
        }
    }
}

private class SecondClickListener implements View.OnClickListener {
    private int mPosition;

    void setPosition(int position) {
        mPosition = position;
    }

    @Override
    public void onClick(View v) {
        mClickInterface.clickEventTwo(mObjects.get(mPosition), v);
    }
}

@Override
public int getItemCount() {
    return mObjects.size();
}

protected class CustomHolder extends RecyclerView.ViewHolder {
    FirstClickListener firstClickListener;
```

```
SecondClickListener secondClickListener;
View v1, v2;

public DialogHolder(View itemView) {
    super(itemView);
    v1 = itemView.findViewById(R.id.v1);
    v2 = itemView.findViewById(R.id.v2);
    firstClickListener = new FirstClickListener();
    secondClickListener = new SecondClickListener();

    v1.setOnClickListener(firstClickListener);
    v2.setOnClickListener(secondClickListener);
}

}
```

Когда у вас есть экземпляр адаптера, вы можете установить свой слушатель кликов, который будет прослушивать щелчки на каждом из представлений:

```
customAdapter.setOnClickListener(new CustomAdapter.ClickInterface {
    @Override
    public void clickEventOne(Object obj) {
        // Здесь ваша реализация
    }
    @Override
    public void clickEventTwo(Object obj1, Object obj2) {
        // Здесь ваша реализация
    }
});
```

18.4. Новый пример

```
public class SampleAdapter extends RecyclerView.Adapter<SampleAdapter.ViewHolder> {  
    private String[] mDataSet;  
    private OnRVItemClickListener mListener;  
  
    /**  
     * Укажите ссылку на тип используемых представлений (пользовательский ViewHolder)  
     */  
    public static class ViewHolder extends RecyclerView.ViewHolder {  
        private final TextView textView;  
  
        public ViewHolder(View v) {  
            super(v);  
            // Определяем слушателя щелчка для ViewHolder View.  
            v.setOnClickListener(new View.OnClickListener() {  
                @Override  
                public void onClick(View v) { // обрабатываем события клика здесь  
                    Log.d(TAG, "Element " + getPosition() + " clicked.");  
                    mListener.onRVItemClicked(getPosition(), v); // устанавливаем  
                    // обратный вызов  
                }  
            });  
            textView = (TextView) v.findViewById(R.id.textView);  
        }  
  
        public TextView getTextView() {  
            return textView;  
        }  
    }  
}
```

```

        return textView;
    }

    /**
     * Инициализация набора данных адаптера.
     *
     * @param dataSet String[], содержащий данные для заполнения представлений,
     * используемых RecyclerView.
     */
    public SampleAdapter(String[] dataSet) {
        mDataSet = dataSet;
    }

    // Создание новых представлений (вызывается менеджером компоновки)
    @Override
    public ViewHolder onCreateViewHolder(ViewGroup viewGroup, int viewType) {
        // Создание нового представления.
        View v = LayoutInflater.from(viewGroup.getContext())
            .inflate(R.layout.text_row_item, viewGroup, false);

        return new ViewHolder(v);
    }

    // Замена содержимого представления (вызывается менеджером компоновки)
    @Override
    public void onBindViewHolder(ViewHolder viewHolder, final int position) {
        // Получить элемент из набора данных в данной позиции и заменить содержимое
        // представления этим элементом
        viewHolder.getTextView().setText(mDataSet[position]);
    }

    // Возврат размера набора данных (вызывается менеджером компоновки)
    @Override
    public int getItemCount() {
        return mDataSet.length;
    }

    public void setOnRVClickListener(OnRVIItemClickListener) {
        mListener = OnRVIItemClickListener;
    }

    public interface OnRVIItemClickListener {
        void onRVIItemClicked(int position, View v);
    }
}

```

18.5. Простой пример OnLongClick и OnClick

Прежде всего реализуйте держатель вида:

```
implements View.OnClickListener, View.OnLongClickListener
```

Затем зарегистрируйте слушателей следующим образом:

```
itemView.setOnClickListener(this);
itemView.setOnLongClickListener(this);
```

Далее переопределите слушателей следующим образом:

```

@Override
public void onClick(View v) {
    onclicklistner.onItemClick(getAdapterPosition(), v);
}

@Override
public boolean onLongClick(View v) {
    onclicklistner.onItemLongClick(getAdapterPosition(), v);
    return true;
}

```

И, наконец, добавьте следующий код:

```

public void setOnItemClickListener(OnClickListener onclicklistner) {
    SampleAdapter.onclicklistner = onclicklistner;
}

public void setHeader(View v) {
    this.headerView = v;
}

public interface OnClickListener {
    void onItemClick(int position, View v);
    void onItemLongClick(int position, View v);
}

```

Демонстрация адаптера:

```

package adaptor;
import android.annotation.SuppressLint;
import android.content.Context;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

import com.wings.example.recyclerview.MainActivity;
import com.wings.example.recyclerview.R;

import java.util.ArrayList;

public class SampleAdapter extends RecyclerView.Adapter<RecyclerView.ViewHolder> {
    Context context;
    private ArrayList<String> arrayList;
    private static OnClickListener onclicklistner;
    private static final int VIEW_HEADER = 0;
    private static final int VIEW_NORMAL = 1;
    private View headerView;

    public SampleAdapter(Context context) {
        this.context = context;
        arrayList = MainActivity.arrayList;
    }

    public class HeaderViewHolder extends RecyclerView.ViewHolder {
        public HeaderViewHolder(View itemView) {
            super(itemView);
        }
    }
}

```

```

public class ItemViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener, View.OnLongClickListener {
    TextView txt_pos;
    SampleAdapter sampleAdapter;

    public ItemViewHolder(View itemView, SampleAdapter sampleAdapter) {
        super(itemView);

        itemView.setOnClickListener(this); itemView.setOnLongClickListener(this);

        txt_pos = (TextView) itemView.findViewById(R.id.txt_pos);
        this.sampleAdapter = sampleAdapter;

        itemView.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        onclicklistner.onItemClick(getAdapterPosition(), v);
    }

    @Override
    public boolean onLongClick(View v) {
        onclicklistner.onItemLongClick(getAdapterPosition(), v);
        return true;
    }
}

public void setOnItemClickListener(OnClickListener onclicklistner) {
    SampleAdapter.onclicklistner = onclicklistner;
}

public void setHeader(View v) {
    this.headerView = v;
}

public interface OnClickListener {
    void onItemClick(int position, View v);
    void onItemLongClick(int position, View v);
}

@Override
public int getItemCount() {
    return arrayList.size();
}

@Override
public int getItemViewType(int position) {
    return position == 0 ? VIEW_HEADER : VIEW_NORMAL;
}

@SuppressWarnings("InflateParams")
@Override
public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup viewGroup, int viewType) {
    if (viewType == VIEW_HEADER) {
        return new HeaderViewHolder(headerView);
    } else {
}

```

```

        View view = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.
            custom_recycler_row_sample_item, viewGroup, false);
        return new ItemViewHolder(view, this);
    }
}

@Override
public void onBindViewHolder(RecyclerView.ViewHolder viewHolder, int position) {
    if (viewHolder.getItemViewType() == VIEW_HEADER) {
        return;
    } else {
        ItemViewHolder itemViewHolder = (ItemViewHolder) viewHolder;
        itemViewHolder.txt_pos.setText(arrayList.get(position-1));
    }
}
}

```

Приведенный выше код примера может быть вызван при помощи:

```

sampleAdapter.setOnItemClickListener(new SampleAdapter.OnClickListner() {
    @Override
    public void onItemClick(int position, View v) {
        position = position+1; //При добавлении заголовка
        Log.e(TAG + "ON ITEM CLICK", position + "");
        Snackbar.make(v, "On item click "+position, Snackbar.LENGTH_LONG).show();
    }

    @Override
    public void onItemLongClick(int position, View v) {
        position = position+1; //При добавлении заголовка
        Log.e(TAG + "ON ITEM LONG CLICK", position + "");
        Snackbar.make(v, "On item longclick "+position, Snackbar.LENGTH_LONG).show();
    }
});

```

18.6. Слушатели клика элементов

Для реализации слушателя клика по элементу и/или слушателя длительного клика по элементу можно создать интерфейс в адаптере:

```

public class CustomAdapter extends RecyclerView.Adapter<CustomAdapter.ViewHolder> {

    public interface OnItemClickListener {
        void onItemSelected(int position, View view, CustomObject object);
    }

    public interface OnItemLongClickListener {
        boolean onItemSelected(int position, View view, CustomObject object);
    }

    public final class ViewHolder extends RecyclerView.ViewHolder {
        public ViewHolder(View itemView) {
            super(itemView);
            final int position = getAdapterPosition();

            itemView.setOnClickListener(new View.OnClickListener() {

```

```

        @Override
        public void onClick(View view) {
            if(mOnItemClickListener != null) {
                mOnItemClickListener.onItemSelected(position, view, mDataSet.
get(position));
            }
        }
    });

itemView.setOnLongClickListener(new View.OnLongClickListener() {
    @Override
    public boolean onLongClick(View view) {
        if(mOnItemLongClickListener != null) {
            return mOnItemLongClickListener.onItemSelected(position, view,
mDataSet.get(position));
        }
    }
});

private List<CustomObject> mDataSet;
private OnItemClickListener mOnItemClickListener;
private OnItemLongClickListener mOnItemLongClickListener;

public CustomAdapter(List<CustomObject> dataSet) {
    mDataSet = dataSet;
}

@Override
public CustomAdapter.ViewHolder onCreateViewHolder(ViewGroup parent, int
viewType) {
    View view = LayoutInflater.from(parent.getContext())
        .inflate(R.layout.view_item_custom, parent, false);
    return new ViewHolder(view);
}

@Override
public void onBindViewHolder(CustomAdapter.ViewHolder holder, int position) {
    // Связывание представлений
}

@Override
public int getItemCount() {
    return mDataSet.size();
}

public void setOnItemClickListener(OnItemClickListener listener) {
    mOnItemClickListener = listener;
}

public void setOnItemLongClickListener(OnItemLongClickListener listener) {
    mOnItemLongClickListener = listener;
}
}

```

Тогда после создания экземпляра адаптера можно установить слушателей кликов:

```
customAdapter.setOnItemClickListener(new CustomAdapter.OnItemClickListener {
    @Override
    public void onItemClick(int position, View view, CustomObject object) {
        // Здесь ваша реализация
    }
});

customAdapter.setOnItemLongClickListener(new CustomAdapter.OnItemLongClickListener {
    @Override
    public boolean onItemSelected(int position, View view, CustomObject object) {
        // Здесь ваша реализация
        return true;
    }
});
```

Глава 19. RecyclerView и LayoutManagers

19.1. Добавление представления заголовка в RecyclerView с помощью менеджера GridLayout

Чтобы добавить заголовок в RecyclerView с помощью GridLayout, сначала адаптеру необходимо указать, что первой позицией является представление заголовка, а не стандартная ячейка, используемая для содержимого. Затем менеджеру компоновки необходимо указать, что первая позиция должна иметь колонку (span), равную количеству колонок всего списка.

Возьмем обычный класс RecyclerView.Adapter и настроим его следующим образом:

```
public class HeaderAdapter extends RecyclerView.Adapter<RecyclerView.ViewHolder> {

    private static final int ITEM_VIEW_TYPE_HEADER = 0;
    private static final int ITEM_VIEW_TYPE_ITEM = 1;

    private List<YourModel> mModelList;

    public HeaderAdapter (List<YourModel> modelList) {
        mModelList = modelList;
    }

    public boolean isHeader(int position) {
        return position == 0;
    }

    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        LayoutInflater inflater = LayoutInflater.from(parent.getContext());

        if (viewType == ITEM_VIEW_TYPE_HEADER) {
            View headerView = inflater.inflate(R.layout.header, parent, false);
            return new HeaderHolder(headerView);
        }

        View cellView = inflater.inflate(R.layout.gridcell, parent, false);
```

```

        return new ModelHolder(cellView);
    }

@Override
public int getItemViewType(int position) {
    return isHeader(position) ? ITEM_VIEW_TYPE_HEADER : ITEM_VIEW_TYPE_ITEM;
}

@Override
public void onBindViewHolder(RecyclerView.ViewHolder h, int position) {
    if (isHeader(position)) {
        return;
    }

    final YourModel model = mModelList.get(position - 1); // Вычтите 1 для заголовка

    ModelHolder holder = (ModelHolder) h;
    // заполните держатель данными из вашей модели, как обычно
}

@Override
public int getItemCount() {
    return _categories.size() + 1; // добавить единицу для заголовка
}
}

```

Затем в активности/фрагменте:

```

final HeaderAdapter adapter = new HeaderAdapter (mModelList);
final GridLayoutManager manager = new GridLayoutManager();
manager.setSpanSizeLookup(new GridLayoutManager.SpanSizeLookup() {
    @Override
    public int getSpanSize(int position) {
        return adapter.isHeader(position) ? manager.getSpanCount() : 1;
    }
});
mRecyclerView.setLayoutManager(manager);
mRecyclerView.setAdapter(adapter);

```

Аналогичным образом можно добавить нижний колонтитул в дополнение или вместо верхнего колонтитула. Источник: <http://blog.sqisland.com/2014/12/recyclerview-grid-with-header.html>.

19.2. GridLayoutManager с динамическим подсчетом диапазонов

При создании RecyclerView с менеджером компоновки GridLayout необходимо указать в конструкторе количество колонок (span count). Это довольно неудобно и не учитывает большие размеры экрана и его ориентацию. Один из подходов заключается в создании нескольких макетов для различных размеров экрана. Другой, более динамичный подход, показан ниже.

Сначала создадим пользовательский класс RecyclerView следующим образом:

```

public class AutofitRecyclerView extends RecyclerView {
    private GridLayoutManager manager;
    private int columnWidth = -1;

    public AutofitRecyclerView(Context context) {
        super(context);
    }
}

```

```

    init(context, null);
}

public AutofitRecyclerView(Context context, AttributeSet attrs) {
    super(context, attrs);
    init(context, attrs);
}

public AutofitRecyclerView(Context context, AttributeSet attrs, int defStyle) {
    super(context, attrs, defStyle);
    init(context, attrs);
}

private void init(Context context, AttributeSet attrs) {
    if (attrs != null) {
        int[] attrsArray = {
            android.R.attr.columnWidth
        };
        TypedArray array = context.obtainStyledAttributes(attrs, attrsArray);
        columnWidth = array.getDimensionPixelSize(0, -1);
        array.recycle();
    }
    manager = new GridLayoutManager(getContext(), 1);
    setLayoutManager(manager);
}

@Override
protected void onMeasure(int widthSpec, int heightSpec) {
    super.onMeasure(widthSpec, heightSpec);
    if (columnWidth > 0) {
        int spanCount = Math.max(1, getMeasuredWidth() / columnWidth);
        manager.setSpanCount(spanCount);
    }
}
}

```

Этот класс определяет, сколько колонок может поместиться в RecyclerView. Для его использования необходимо поместить его в файл layout.xml следующим образом:

```

<?xml version="1.0" encoding="utf-8"?>
<com.path.to.your.class.autofitRecyclerView.AutofitRecyclerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/auto_fit_recycler_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnWidth="200dp"
    android:clipToPadding="false"
/>

```

Обратите внимание, что мы используем атрибут columnWidth. Он понадобится RecyclerView, чтобы определить, сколько колонок поместится в свободное пространство.

В своей активности или фрагменте вы просто получаете ссылку на RecyclerView и устанавливаете для него адаптер (и любые украшения или анимации элементов, которые вы хотите добавить). **НЕ УСТАНАВЛИВАЙТЕ МЕНЕДЖЕР КОМПОНОВКИ (LAYOUT MANAGER)!**

```

RecyclerView recyclerView = (RecyclerView) findViewById(R.id.auto_fit_recycler_view);
recyclerView.setAdapter(new MyAdapter());

```

где MyAdapter – класс вашего адаптера.

Теперь у вас есть RecyclerView, которое будет подстраивать spanCount (то есть столбцы) под размер экрана. В качестве последнего дополнения вы можете захотеть выровнять столбцы в окне RecyclerView по центру (по умолчанию они выровнены по layout_start). Это можно сделать, немного изменив класс AutofitRecyclerView. Начните с создания внутреннего класса в окне RecyclerView. Это будет класс, расширяющийся от GridLayoutManager. Он добавит достаточное количество подкладок слева и справа, чтобы отцентрировать строки:

```
public class AutofitRecyclerView extends RecyclerView {
    // и т. д. см. выше

    private class CenteredGridLayoutManager extends GridLayoutManager {
        public CenteredGridLayoutManager(Context context, AttributeSet attrs, int defStyleAttr, int defStyleRes) {
            super(context, attrs, defStyleAttr, defStyleRes);
        }

        public CenteredGridLayoutManager(Context context, int spanCount) {
            super(context, spanCount);
        }

        public CenteredGridLayoutManager(Context context, int spanCount, int orientation, boolean reverseLayout) {
            super(context, spanCount, orientation, reverseLayout);
        }

        @Override
        public int getPaddingLeft() {
            final int totalItemWidth = columnWidth * getSpanCount();
            if (totalItemWidth >= AutofitRecyclerView.this.getMeasuredWidth()) {
                return super.getPaddingLeft(); // ничего не делать
            } else {
                return Math.round((AutofitRecyclerView.this.getMeasuredWidth() / (1f + getSpanCount())) - (totalItemWidth / (1f + getSpanCount())));
            }
        }

        @Override
        public int getPaddingRight() {
            return getPaddingLeft();
        }
    }
}
```

Затем при установке LayoutManager в AutofitRecyclerView используйте следующим образом CenteredGridLayoutManager:

```
private void init(Context context, AttributeSet attrs) {
    if (attrs != null) {
        int[] attrsArray = {
            android.R.attr.columnWidth
        };
        TypedArray array = context.obtainStyledAttributes(attrs, attrsArray);
        columnWidth = array.getDimensionPixelSize(0, -1);
        array.recycle();
    }
}
```

```

    manager = new CenteredGridLayoutManager(getContext(), 1);
    setLayoutManager(manager);
}

```

И все! У вас есть динамический spancount, выровненный по центру gridlayoutmanager на основе RecyclerView. Источники для главы: <http://blog.sqisland.com/2014/12/recyclerview-autofit-grid.html> и <http://stackoverflow.com/questions/29117916/android-centering-item-in-recyclerview/30118826>.

19.3. Простой список с LinearLayoutManager

Этот пример добавляет список мест с изображением и названием, используя в качестве набора данных ArrayList пользовательских Place-объектов.

Макет активности

Макет активности, или фрагмента, или места, где используется RecyclerView, должен содержать только RecyclerView. Не требуется ни ScrollView, ни специального макета.

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.v7.widget.RecyclerView
        android:id="@+id/my_recycler_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</RelativeLayout>

```

Определение модели данных

В качестве модели можно использовать любой класс или примитивный тип данных, например int, String, float[] или CustomObject. Наш RecyclerView будет ссылаться на список (List) этих объектов или примитивов.

Когда элемент списка относится к различным типам данных, таким как текст, числа, изображения (как в данном примере), часто бывает целесообразно использовать пользовательский объект.

```

public class Place {
    // эти поля будут отображаться в элементе списка
    private Bitmap image;
    private String name;

    // типовой конструктор
    public Place(Bitmap image, String name) {
        this.image = image;
        this.name = name;
    }

    // геттеры
    public Bitmap getImage() {
        return image;
    }

    public String getName() {
        return name;
    }
}

```

Расположение элементов списка

Необходимо указать xml-файл компоновки, который будет использоваться для каждого элемента списка. В данном примере для изображения используется ImageView, а для названия – TextView. LinearLayout позиционирует ImageView слева, а TextView справа от изображения.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_vertical"
    android:orientation="horizontal"
    android:padding="8dp">

    <ImageView
        android:id="@+id/image"
        android:layout_width="36dp"
        android:layout_height="36dp"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp" />

    <TextView
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

Создайте адаптер RecyclerView и держатель вида ViewHolder

Далее необходимо наследовать RecyclerView.Adapter и RecyclerView.ViewHolder. Обычная структура классов выглядит следующим образом:

```
public class PlaceListAdapter extends RecyclerView.Adapter<PlaceListAdapter.
ViewHolder> {
    // ...

    public class ViewHolder extends RecyclerView.ViewHolder {
        // ...
    }
}
```

Сначала мы реализуем ViewHolder. Он наследует только конструктор по умолчанию и сохраняет необходимые представления в некоторых полях:

```
public class ViewHolder extends RecyclerView.ViewHolder {
    private ImageView imageView;
    private TextView nameView;

    public ViewHolder(View itemView) {
        super(itemView);

        imageView = (ImageView) itemView.findViewById(R.id.image);
        nameView = (TextView) itemView.findViewById(R.id.name);
    }
}
```

Конструктор адаптера задает используемый набор данных:

```
public class PlaceListAdapter extends RecyclerView.Adapter<PlaceListAdapter.
ViewHolder> {
```

```

private List<Place> mPlaces;

public PlaceListAdapter(List<Place> contacts) {
    mPlaces = contacts;
}

// ...
}

```

Чтобы использовать наш пользовательский макет элемента списка, мы переопределяем метод `onCreateViewHolder(...)`.

В данном примере файл макета называется `place_list_item.xml`.

```

public class PlaceListAdapter extends RecyclerView.Adapter<PlaceListAdapter.
ViewHolder> {
    // ...

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext()).inflate(
            R.layout.place_list_item,
            parent,
            false
        );
        return new ViewHolder(view);
    }
    // ...
}

```

В `onBindViewHolder(...)` мы фактически устанавливаем содержимое представления. Мы получаем используемую модель, находя ее в списке `List` в заданной позиции, а затем устанавливаем изображение и имя на представлениях `ViewHolder`.

```

public class PlaceListAdapter extends RecyclerView.Adapter<PlaceListAdapter.
ViewHolder> {
    // ...

    @Override
    public void onBindViewHolder(PlaceListAdapter.ViewHolder viewHolder, int
    position) {
        Place place = mPlaces.get(position);

        viewHolder.nameView.setText(place.getName());
        viewHolder.imageView.setImageBitmap(place.getImage());
    }
    // ...
}

```

Нам также необходимо реализовать функцию `getItemCount()`, которая просто возвращает размер списка.

```

public class PlaceListAdapter extends RecyclerView.Adapter<PlaceListAdapter.
ViewHolder> {
    // ...

    @Override
    public int getItemCount() {

```

```

        return mPlaces.size();
    }

    // ...
}

```

Генерация случайных данных

Для данного примера мы сгенерируем несколько случайных Places.

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    // ...

    List<Place> places = randomPlaces(5);
    // ...
}

private List<Place> randomPlaces(int amount) {
    List<Place> places = new ArrayList<>();
    for (int i = 0; i < amount; i++) {
        places.add(new Place(
            BitmapFactory.decodeResource(getResources(), Math.random() > 0.5 ?
                R.drawable.ic_account_grey600_36dp :
                R.drawable.ic_android_grey600_36dp),
            "Place #" + (int) (Math.random() * 1000)
        ));
    }
    return places;
}

```

Подключение RecyclerView к PlaceListAdapter и набору данных

Подключить RecyclerView к адаптеру очень просто. Для получения макета списка необходимо установить LinearLayoutManager в качестве менеджера макетов.

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    // ...

    RecyclerView recyclerView = (RecyclerView) findViewById(R.id.my_recycler_view);
    recyclerView.setAdapter(new PlaceListAdapter(places));
    recyclerView.setLayoutManager(new LinearLayoutManager(this));
}

```

Готово!

19.4. StaggeredGridLayoutManager

1. Создайте RecyclerView в xml-файле макета:

```

<android.support.v7.widget.RecyclerView
    android:id="@+id/recycleView"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />

```

2. Создайте класс Model для хранения данных:

```

public class PintrestItem {

```