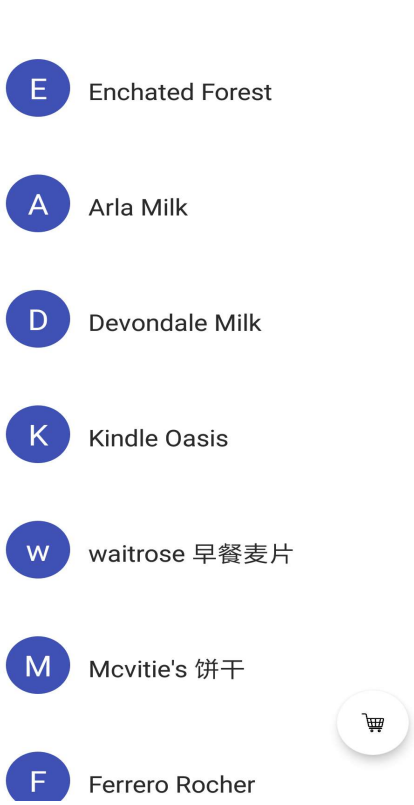
实验三

Intent、Bundle的使用以及RecyclerView、ListView的应用

【实验目的】
1.复习事件处理
2.学习Intent、Bundle在Activity跳转中的应用
3.学习RecyclerView、ListView以及各类适配器的用法

【实验内容】
本次实验模拟实现一个商品表，有两个界面，第一个界面用于呈现商品，如下所示：



下面还有数据，就不截图了，数据在素材中有给出。
点击右下方的悬浮按钮可以切换到购物车:

上面两个列表点击任意一项后，可以看到详细的信息：

实验要求: 布局方面的要求:

1、商品表界面

每一项为一个圆圈和一个名字，圆圈与名字均竖直居中。圆圈中为名字的首字母，首字母要处于圆圈的中心，首字母为白色，名字为黑色，圆圈的颜色自定义即可，建议用深色的颜色，否则白色的首字母可能看不清。

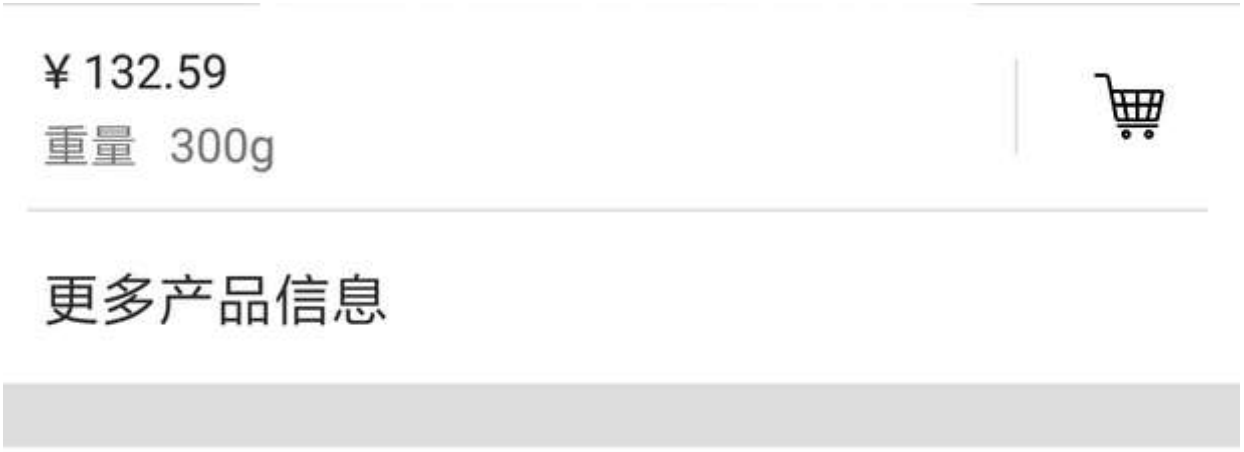
2、购物车列表界面

在商品表界面的基础上增加一个价格,价格为黑色。

3、商品详情界面顶部

顶部占整个界面的1/3。每个商品的图片在商品数据中已给出,图片与这块view等高。返回图标处于这块View的左上角，商品名字处于左下角，星标处于右下角，它们与边距都有一定距离，自己调出合适的距离即可。需要注意的是，返回图标与名字左对齐，名字与星标底边对齐。这一块建议大家使用RelativeLayout实现,以便熟悉RelativeLayout的使用。

4、商品详情界面中部



使用的黑色argb编码值为#D5000000，稍微偏灰色一点的“重量”、“300g”的argb编码值为#8A000000。注意，价格那一栏的下边有一条分割线，argb编码值为#1E000000，右边购物车符号的左边也有一条分割线，argb编码值也是#1E000000，这条分割线要求高度与聊天符号的高度一致，并且竖直居中。字体的大小看着调就可以了。“更多资料”底部的分割线高度自己定，argb编码值与前面的分割线一致。

5、商品详情页面底部

这个没什么说的，内容和样式都很清楚。

6、特别提醒，这次的两个界面顶部都没有标题栏，要用某些方法把它们去掉。

逻辑方面的要求：

1、使用RecyclerView实现商品列表。点击商品列表中的某一个商品会跳转到该商品详情界面，呈现该商品的详细信息；长按商品列表中的第i个商品会删除该商品，并且弹出Toast,提示"移除第i个商品"。

2、点击右下方的FloatingActionButton,从商品列表切换到购物车或从购物车切换到商品列表,并且FloatingActionButton的图片要做相应改变。可通过设置RecyclerView不可见,ListView可见来实现从商品列表切换到购物车。可通过设置RecyclerView可见,ListView不可见来实现从购物车切换到商品列表。

3、使用ListView实现购物车。点击购物车的某一个商品会跳转到商品详情界面，呈现该商品的详细信息；长按购物中的商品会弹出对话框询问是否移除该商品，点击确定则移除该商品，点击取消则对话框消失。

注意对话框中的商品名为被长按的商品。

4、商品详情界面中点击返回图标会返回上一层，点击星标会切换状态，如果原先是空心星星，则会变成实心星星；如果原先是实心星星，则会变成空心星星。点击购物车图标会将该商品添加到购物车中并弹出Toast提示:"商品已添加到购物车"。
注:不要求判断购物车是否已有该商品,即如果已有一件该商品,添加之后则显示两个即可。未退出商品详细界面时,点击多次购物车图标可以只添加一件商品也可以添加多件到购物车中。

【基础知识】

1、ListView的使用

布局上比较简单，在布局文件中写上

```
<ListView
    android:id="@+id/shoppinglist"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

即可，这样就创建了一个空的列表，然后在.java文件中再填充数据，所以id是一定要设的。

在.java文件中获得这个ListView之后，使用Adapter为这个ListView填充数据，常用的Adapter有ArrayAdapter、SimpleAdapter，这两种在下面会详细讲述如何使用。随着ListView中内容的丰富，以上两种Adapter已经很难满足需要，因此现在一般使用自定义的Adapter来填充数据，如何使用自定义的Adapter会在拓展知识中讲。

ArrayAdapter

最简单的Adapter，创建ArrayAdapter时需指定如下三个参数：

Context：这个参数无须多说，它代表了访问整个Android应用的接口。几乎创建所有组件都需要传入Context对象。

textViewResourceId：一个资源ID，该资源ID代表一个TextView，该 TextView组件将作为ArrayAdapter的列表项组件。

数组或List：该数组或List将负责为多个列表项提供数据。

示例：

```
operations = new String[] {"text1", "text2", "text3"};
ArrayAdapter<String> arrayAdapter
        = new ArrayAdapter<>(this, R.layout.operation_list_item, operations);
operationList.setAdapter(arrayAdapter);
```

在创建完ArrayAdapter之后，调用ListView的setAdapter方法即可将数据填充到ListView中。

这里有一点要特别注意的是textViewResourceId是一个layout，在这个layout中只能有一个TextView，其它任何组件都不能有，包括LinearLayout等布局组件，否则会报错。

SimpleAdapter

由于ArrayAdapter只能显示文字，功能实在有限，如果需要多填充一些内容的话指望不上，这时候可以使用SimpleAdapter。

SimpleAdapter相比ArrayAdapter强大很多，创建SimpleAdapter需要5个参数，第一个参数依然是Context，就不多说了，下面介绍余下的4个参数：

第2个参数：该参数应该是一个List<? Extends Map<String, ?>>类型的集合对象，该集合中每个Map<String, ?>对象生成一个列表项。

第3个参数：该参数指定一个界面布局的ID。该界面布局指定每一个列表项的样式。

第4个参数：该参数应该是一个String[]类型的参数，该参数决定提取Map<String, ?>对象中哪些key对应的value来生成列表项。

第5个参数：该参数应该是一个int[]类型的参数，该参数决定填充哪些组件。

示例： 首先构建好数据，这里模拟了一个图书清单，一个map中有两个key，存放书名和价格，然后添加到list中。

```
List<Map<String, Object>> data = new ArrayList<>();
String[] booksName = new String[] {"疯狂Android讲义(第2版)", "疯狂Java讲义(第3版)", "设计模式"};
String[] booksPrice = new String[] {"99.00元", "109.00元", "35.00元"};
for (int i = 0; i < 3; i++) {
    Map<String, Object> temp = new LinkedHashMap<>();
    temp.put("name", booksName[i]);
    temp.put("price", booksPrice[i]);
    data.add(temp);
}
```

然后创建SimpleAdapter：

```
ListView listview = (ListView) findViewById(R.id.listview);
SimpleAdapter simpleAdapter = new SimpleAdapter(this, data, R.layout.item,
        new String[] {"name", "price"}, new int[] {R.id.name, R.id.price});
listview.setAdapter(simpleAdapter);
```

之后还是用ListView的setAdapter方法添加Adapter。

看一下R.layout.item文件：

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

    <TextView
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="15dp"
        android:textSize="20sp"
        android:textColor="#000000"/>

    <TextView
        android:id="@+id/price"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:gravity="end"
        android:padding="15dp"
        android:textSize="20sp"
        android:textColor="#000000"/>

</LinearLayout>
```

可以看到，一个LinearLayout包含两个TextView，一个用于显示书名，一个用于显示价格，这个layout用于规定ListView中每一个列表项的样式。SimpleAdapter中的第四个参数String数组与map的两个key对应，第五个参数int数组与这个layout中两个TextView的id相对应，注意String[]数组与int[]数组中的值要一一对应，在这个示例中，key为name的value填充到id为name的TextView中。效果如下图所示：

2、ListView列表项的单击和长按

方法原型如下：

```
shoppingList.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
        //单击事件处理在这里进行
    }
});

shoppingList.setOnItemLongClickListener((adapterView, view, pos, l) -> {
        //长按事件处理在这里进行
        return false;
});
```

长按有返回值，在理论课的课件中写的很清楚了，这里就不解释了。注意在两个方法的参数中都有int i, long l这两个参数，i指的是这一项在列表中的位置，l指的是这一项的id，在ArrayAdapter和SimpleAdapter中，i和l是相等的，在另一种Adapter——CursorAdapter中，l指的是从数据库中取出的数据在数据库中的id值。

3、ListView数据更新 直观地想，要实现数据更新，只要更新List，重新创建一个SimpleAdapter就可以了，这样会比较麻烦，SimpleAdapter有一个notifyDataSetChanged()方法，当之前创建该SimpleAdapter的List发生改变时，调用该方法就可以刷新列表了。要特别注意的一点是，List不能指向新的内存地址，即不能list = new ArrayList<>();这样是不起作用的，只能调用它的remove()，add()等方法来改变数据集。

示例：

```
data.remove(0);
simpleAdapter.notifyDataSetChanged();
```

错误写法:

```
data = new ArrayList<>();
simpleAdapter.notifyDataSetChanged();
```

4、RecyclerView的使用
在project的build.gradle文件添加依赖:

```
allprojects {
    repositories {
        jcenter()
        maven {
            url "https://maven.google.com"
        }
    }
}
```

在app的build.gradle文件添加依赖:

compile 'com.android.support:cardview-v7:25.4.0'
compile 'com.android.support:recyclerview-v7:25.4.0'

在布局文件加上

```
<android.support.v7.widget.RecyclerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/recycler_view"
    />
```

这样就创建了一个空的RecyclerView.在.java文件中获得这个RecyclerView之后，选择下面其中一种,设置其显示方式:

mRecyclerView.setLayoutManager(new LinearLayoutManager(this));//这里用线性显示 类似于listview

mRecyclerView.setLayoutManager(new GridLayoutManager(this, 2));//这里用线性宫格显示 类似于grid view

mRecyclerView.setLayoutManager(new StaggeredGridLayoutManager(2, OrientationHelper.VERTICAL));//这里用线性宫格显示 类似于瀑布流

使用Adapter为这个RecyclerView填充数据:

mRecyclerView.setAdapter(commonAdapter);

在RecyclerView中必须自定义实现RecyclerView.Adapter并为其提供数据集合，实现时必须遵循ViewHolder设计模式。ViewHolder通常出现在适配器里，为的是listview、RecyclerView滚动的时候快速设置值，而不必每次都重新创建很多对象，从而提升性能。

自定义RecyclerView.ViewHolder

public class ViewHolder extends RecyclerView.ViewHolder

使用一个SparseArray数组存储list_Item的子View：

```
private SparseArray<View> mViews;//存储list_Item的子View
private View mConvertView;//存储list_Item

public ViewHolder(Context context, View itemView, ViewGroup parent)
{
    super(itemView);
    mConvertView = itemView;
    mViews = new SparseArray<View>();
}
```

获取viewHolder实例:

```
public static ViewHolder get(Context context, ViewGroup parent, int layoutId)
{
    View itemView = LayoutInflater.from(context).inflate(layoutId, parent,
            false);
    ViewHolder holder = new ViewHolder(context, itemView, parent);
    return holder;
}
```

viewHolder尚未将子View缓存到SparseArray数组中时,仍然需要通过findViewById()创建View对象,如果已缓存,直接返回：

```
public <T extends View> T getView(int viewId)
{
    View view = mViews.get(viewId);
    if (view == null)
    {
        //创建view
        view = mConvertView.findViewById(viewId);
        //将view存入mViews
        mViews.put(viewId, view);
    }
    return (T) view;
}
```

自定义的RecyclerView.Adapter

public abstract class CommonAdapter extends RecyclerView.Adapter

其构造函数为:

public CommonAdapter(Context context, int layoutId, List datas)

Adapter扮演着两个角色。一是，根据不同ViewType创建与之相应的的Item-Layout，二是，访问数据集合并将数据绑定到正确的View上。这就需要我们重写以下两个函数：

public VH onCreateViewHolder(ViewGroup parent, int viewType) 创建Item视图，并返回相应的ViewHolder

public void onBindViewHolder(VH holder, int position) 绑定数据到正确的Item视图上。

```
@Override
public ViewHolder onCreateViewHolder(final ViewGroup parent, int viewType)
{
    ViewHolder viewHolder = ViewHolder.get(mContext, parent, mLayoutId);
    return viewHolder;
}


@Override
public void onBindViewHolder(final ViewHolder holder, int position)
{
    convert(holder, mDatas.get(position));
}
```

另外我们还需要重写另一个方法，像ListView-Adapter那样，同样地告诉RecyclerView-Adapter列表Items的总数：

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

重写适配器中的convert方法进行数据绑定:

```
commonAdapter=new CommonAdapter<Map<String, Object>>(this, R.layout.goods_list_item, listItems)
{
    @Override
    public void convert(ViewHolder holder, Map<String, Object> s) {
        TextView name = holder.getView(R.id.name);
        name.setText(s.get("name").toString());
        TextView first = holder.getView(R.id.first);
        first.setText(s.get("firstLetter").toString());
    }

};
```

RecyclerView没有OnItemClickListener方法,需要在适配器中实现。实现的方法为:在Adapter中设置一个监听器,当itemView被点击时候,调用该监听器并且将itemView的position作为参数传递出去。

首先添加接口及函数:

```
public interface OnItemClickListener{
    void onClick( int position);
    void onLongClick( int position);
}

public void setOnItemClickListener(OnItemClickListener onItemClickListener ){
    this. mOnItemClickListener=onItemClickListener;
}
```

在onBindViewHolder()中添加:

```
if( mOnItemClickListener!= null){
    holder.itemView.setOnClickListener( new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mOnItemClickListener.onClick(holder.getAdapterPosition());

        }
    });
    holder.itemView.setOnLongClickListener( new View.OnLongClickListener() {
        @Override
        public boolean onLongClick(View v) {
            mOnItemClickListener.onLongClick(holder.getAdapterPosition());
            return false;
        }
    });
}
```

这样就可以使用setOnItemClickListener()方法了.

5、为RecyclerView添加动画

本次实验使用网上的库,如果自己实现了好看的动画,在实验报告中说明，有加分。

添加依赖:

compile 'jp.wasabeef:recyclerview-animators:2.2.7'
compile "com.android.support:support-core-utils:25.4.0"

将

mRecyclerView.setAdapter(commonAdapter);

改成

```
ScaleInAnimationAdapter animationAdapter=new ScaleInAnimationAdapter(commonAdapter);
animationAdapter.setDuration(1000);
mRecyclerView.setAdapter(animationAdapter);
mRecyclerView.setItemAnimator(new OvershootInLeftAnimator());
```

这样就添加了动画效果了。https://github.com/wasabeef/recyclerview-animators上有动画的详细说明,可以自行更换。

6、去掉标题栏

要去掉标题栏有多种做法，这里举一种方法。 Android Studio创建项目时默认的theme是：

```
android:theme="@style/AppTheme">
```

它的定义是:

```
<!-- Base application theme. -->
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
</style>
```

修改parent即可:

```
<style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
```

7、星星图标的切换

星星的切换难点在于如何得知星星此时是空心的还是实心的，这个也有多种做法，这里也只介绍一种。

每个View都可以设置tag，通过tag可以用来判断该View现在的状态。在初始化的时候，将tag设置为0，标记此时为空心星星，如果星星被点击了，并且tag为0，那么就把图片换为实心的星星，然后设置tag为1；如果tag为1，那么就把图片换为空心的星星，然后设置tag为0。建议在java文件中给需要的view设置tag。

8、FLoatingActionButton

添加依赖:

compile 'com.android.support:design:25.4.0'

设置布局文件:

```
<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@mipmap/shoplist"
    app:backgroundTint="@color/white"
    app:backgroundTintMode="src_atop"
    app:rippleColor="@color/white"
    />
```

切换图片的时候使用setImageResource().

【扩展知识】

ListView的自定义Adapter

前面介绍的ArrayAdapter和SimpleAdapter都有一定的局限性，SimpleAdapter较ArrayAdapter要好一些，但还是不够灵活，假如我的某些列表项需要有一些特性，或者我的列表项中的某些控件需要设置监听器，就不够用了。因此，强烈建议大家一开始就习惯自定义Adapter来适配自己的列表，只在某些简单的情况下才使用前面介绍的两种Adapter。

自定义的Adapter需要继承BaseAdapter：

```
public class MyAdapter extends BaseAdapter {
    @Override
    public int getCount() {
        return 0;
    }

    @Override
    public Object getItem(int i) {
        return null;
    }

    @Override
    public long getItemId(int i) {
        return 0;
    }

    @Override
    public View getView(int i, View view, ViewGroup viewGroup) {
        return null;
    }
}
```

上面列出的四个方法是必须重写的四个方法，下面一一介绍这四个方法：

int getCount();获得数据项列表的长度，也就是一共有多少个数据项。

Object getItem(int i);获得某一个数据项。

long getItemId(int i);获得数据项的位置。

View getView(int i, View view, ViewGroup viewGroup);获得数据项的布局样式，最重要的一个方法。

自定义Adapter需要提供给一个数据列表才能填充数据，一般是一个List类型，以我们刚刚图书列表的例子为例，我们可以先给列表项创建一个类Book，然后将List传入Adapter中作为数据提供的列表：

```java
public class Book {

    private String bookName;
    private String bookPrice;

    public Book(String bookName, String bookPrice) {
        this.bookName = bookName;
        this.bookPrice = bookPrice;
    }

    public String getBookName() {
        return bookName;
    }

    public String getBookPrice() {
        return bookPrice;
    }
}
```

```java
public class MyAdapter extends BaseAdapter {

    private List<Book> list;

    public MyAdapter(List<Book> list) {
        this.list = list;
    }

    @Override
    public int getCount() {
        if (list == null) {
            return 0;
        }
        return list.size();
    }

    @Override
    public Object getItem(int i) {
        if (list == null) {
            return null;
        }
        return list.get(i);
    }

    @Override
    public long getItemId(int i) {
        return i;
    }
```

依照刚刚的想法重写完之后，接下来是最重要的重写getView()方法，首先解释一下三个参数的含义：

i指的是当前是在加载第几项的列表项

viewGroup是列表项View的父视图，调整列表项的宽高用的

view指的是一个列表项的视图，我们需要给view一个布局，然后在布局中放置我们需要的内容

```
@Override
public View getView(int i, View view, ViewGroup viewGroup) {
    //  通过inflate的方法加载布局，context这个参数需要使用这个adapter的Activity传入
    view = LayoutInflater.from(context).inflate(R.layout.item, null);
    //  获得布局中显示书名和价格的两个TextView
    TextView bookName = (TextView) view.findViewById(R.id.name);
    TextView bookPrice = (TextView) view.findViewById(R.id.price);
    //  从数据列表中取出对应的对象，然后赋值给它们
    bookName.setText(list.get(i).getBookName());
    bookPrice.setText(list.get(i).getBookPrice());
    //  将这个处理好的view返回
    return view;
}
```

getView()方法的最基本写法：

这种方法没有充分利用view的特点，只是每次从屏幕外滚进来一个新的项就再加载一次布局。其实ListView每次从屏幕外滚进来一项就会有一项滚出屏幕外，这个时候view是有内容的，不过是旧的内容，因此我们只需要改变一下view的内容然后返回它就可以了，不需要再去加载一次布局。

getView()方法的改进版写法：

```
@Override
public View getView(int i, View view, ViewGroup viewGroup) {
    //  当view为空时才加载布局，否则，直接修改内容即可
    if (view == null) {
        //  通过inflate的方法加载布局，context这个参数需要使用这个adapter的Activity传入
        view = LayoutInflater.from(context).inflate(R.layout.item, null);
    }
    //  获得布局中显示书名和价格的两个TextView
    TextView bookName = (TextView) view.findViewById(R.id.name);
    TextView bookPrice = (TextView) view.findViewById(R.id.price);
    //  从数据列表中取出对应的对象，然后赋值给它们
    bookName.setText(list.get(i).getBookName());
    bookPrice.setText(list.get(i).getBookPrice());
    //  将这个处理好的view返回
    return view;
}
```

这样写可以减少一些重复的加载布局的操作，提高效率。但是每次findViewById()也是一件很麻烦的事情，如果控件一多，也会降低ListView的效率。因此，使用setTag的方法和新建一个ViewHolder类来提高这部分的效率。

getView()方法改进版2.0：

```java
@Override
public View getView(int i, View view, ViewGroup viewGroup) {

    //  新声明一个View变量和ViewHolder变量
    View convertView;
    ViewHolder viewHolder;

    //  当view为空时才加载布局，并且创建一个ViewHolder，获得布局中的两个控件
    if (view == null) {
        //  通过inflate的方法加载布局，context这个参数需要使用这个adapter的Activity传入
        convertView = LayoutInflater.from(context).inflate(R.layout.item, null);
        viewHolder = new ViewHolder();
        viewHolder.bookName = (TextView) convertView.findViewById(R.id.name);
        viewHolder.bookPrice = (TextView) convertView.findViewById(R.id.price);
        convertView.setTag(viewHolder);  //  用setTag方法将处理好的viewHolder放入view中
    } else {  //  否则，让convertView等于view，然后从中取出ViewHolder即可
        convertView = view;
        viewHolder = (ViewHolder) convertView.getTag();
    }
    //  从viewHolder中取出对应的对象，然后赋值给它们
    viewHolder.bookName.setText(list.get(i).getBookName());
    viewHolder.bookPrice.setText(list.get(i).getBookPrice());
    //  将这个处理好的view返回
    return convertView;
}

private class ViewHolder {
    public TextView bookName;
    public TextView bookPrice;
}
```

这样写的话ListView的效率就比较高了。

贴一下最终版的自定义Adapter：

```java
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.TextView;

import java.util.List;

public class MyAdapter extends BaseAdapter {

    private Context context;
    private List<Book> list;

    public MyAdapter(Context context, List<Book> list) {
        this.context = context;
        this.list = list;
    }

    @Override
    public int getCount() {
        if (list == null) {
            return 0;
        }
        return list.size();
    }

    @Override
    public Object getItem(int i) {
        if (list == null) {
            return null;
        }
        return list.get(i);
    }

    @Override
    public long getItemId(int i) {
        return i;
    }

```

```
    @Override
    public View getView(int i, View view, ViewGroup viewGroup) {

        //  新声明一个View变量和ViewHolder变量
        View convertView;
        ViewHolder viewHolder;

        //  当view为空时才加载布局，并且创建一个ViewHolder，获得布局中的两个控件
        if (view == null) {
            //  通过inflate的方法加载布局，context这个参数需要使用这个adapter的Activity传入
            convertView = LayoutInflater.from(context).inflate(R.layout.item, null);
            viewHolder = new ViewHolder();
            viewHolder.bookName = (TextView) convertView.findViewById(R.id.name);
            viewHolder.bookPrice = (TextView) convertView.findViewById(R.id.price);
            convertView.setTag(viewHolder);  //  用setTag方法将处理好的viewHolder放入view中
        } else {  //  否则，让convertView等于view，然后从中取出ViewHolder即可
            convertView = view;
            viewHolder = (ViewHolder) convertView.getTag();
        }
        //  从viewHolder中取出对应的对象，然后赋值给它们
        viewHolder.bookName.setText(list.get(i).getBookName());
        viewHolder.bookPrice.setText(list.get(i).getBookPrice());
        //  将这个处理好的view返回
        return convertView;
    }

    private class ViewHolder {
        public TextView bookName;
        public TextView bookPrice;
    }
}
```

【检查内容】 1、有商品列表与购物车的界面中，圆圈是圆的，字母居中，圆圈与名字竖直居中。购物车的界面中还要求圆圈与价格竖直居中。

2、商品详情界面中，顶部占三分之一的实现方法，返回图标，姓名，星标放置在指定位置，对齐。

3、商品详情界面中，界面中部购物车符号旁边的分割线上下和符号等高并竖直居中

4、商品列表用RecyclerView实现,购物车界面使用ListView实现，单个联系人详情界面中底部的四个操作的列表使用ListView实现

5、单击后跳转各项资料显示正确，星星点击后行为正常，返回图标功能正常

6、商品列表长按后能正确删除商品，删除后列表工作正常（点击其它商品显示信息正确，长按其它商品删除操作正确）

7、购物车长按后弹出框的显示内容正常，点击确定能正确删除商品，删除后列表工作正常（点击其它商品显示信息正确，长按其它商品删除操作正确）

8、悬浮按钮的功能正确。

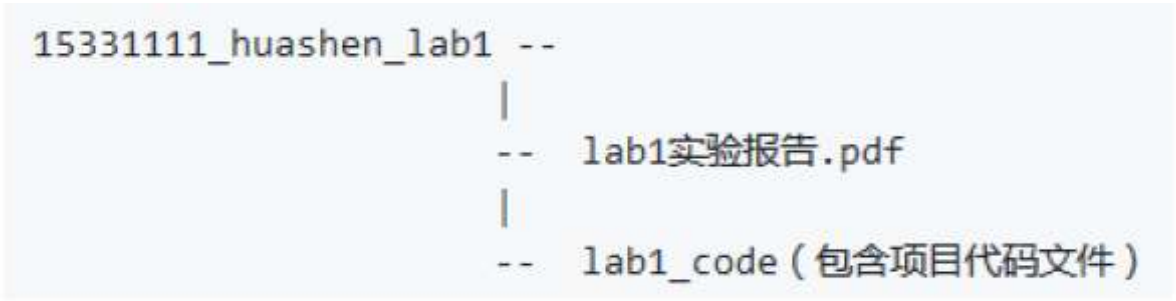
【提交说明】

1、 deadline：下一次实验课前一天晚上12点

2、 提交作业地址：ftp://edin.sysu.edu.cn

3、 文件命名及格式要求：学号*姓名*labX.zip（姓名中文拼音均可）

4、 目录结构：



其中项目代码文件为项目文件夹，*提交之前先 clean