

# Android 应用软件开发

## E 3 Adapter View

学号：SA18225402

姓名：吴文韬

报告撰写时间：2018/09/21

# 一、 主要目标

1. 修改 E2 的 SCOS 工程 mainScreen 布局，使用 GridView 替换已有的按钮导航：点菜、查看订单、登录/注册、系统帮助。形成 launch board 导航样式。

要求：

- 1) GridView 中导航项为：点菜、查看订单、登录/注册、系统帮助
  - 2) 每个导航项都可以点击
  - 3) 每个导航项都有图标和文字说明
  - 4) 当用户点击“登录/注册”按钮时，保持 E2 原有功能实现。
2. 在 SCOS 工程下新建 package 为 es.source.code.model
  3. 在包 es.source.code.model 中新建一个 model 类 User.java, User 包含三个域为：userName, password, oldUser，数据类型分别为：String, String, Boolean, 每个域具有 Setter 和 Getter 方法。
  4. 修改 E2 中的 LoginOrRegister 代码，完成如下功能：
    - 1) 在 LoginOrRegister 界面中添加新按钮“注册”
    - 2) 当用户在 LoginOrRegister 界面点击登录按钮时，按照 E2 验证规则验证用户名输入是否正确，如果正确，则新建 User 对象 loginUser，将用户输入框输入的用户名和密码值通过 Setter 方法赋值到 loginUser 中，并对其 oldUser 域赋值为 True，屏幕跳转至 mainScreen，并向 mainScreen 类传递数据 String 值为“LoginSuccess”，同时将对象 loginUser 传递到 mainScreen 中。如果验证错误，则按照 E2 中要求进行错误提示。
    - 3) 当用户点击返回按钮时，保持 E2 中的功能不变
  5. 修改 mainScreen.java 代码
    - 1) 在 mainScreen.java 中新建一个 User 对象变量 user
    - 2) 判断由 LoginOrRegister 传回的 String 数据，如果返回的数据值为“RegisterSuccess”，则保持 E2 功能不变，并将 user 赋值为 LoginOrRegister 传入的 loginUser

- 3) 如果有 LoginOrRegister 传回的 String 数据值为“RegisterSuccess”，则使用 Toast 提示用户“欢迎您称为 SCOS 新用户”，同时检查“点菜”和“查看订单”导航项是否为隐藏状态，如果为隐藏，则设为显示，并将 user 赋值为 LoginOrRegister 传入的 loginUser
  - 4) 当传入 mainScreen 的 String 值为其他时，将 user 赋值为 NULL
6. 在 SCOS 的 es.source.code.activity 包下新建 Activity 类名为：FoodView，为该 FoodView 定制布局 food\_view.xml, 要求如下：
- 1) 在布局中使用 ViewPager&TabLayout 显示“冷菜”、“热菜”、“海鲜”、“酒水”四个导航目录
  - 2) 支持屏幕左右滑动导航
  - 3) 导航值相应选项时，使用 ListView 或 RecyclerView 加载当前类别的十五列表，并显示到当前屏幕。要求 ListView 或 RecyclerView 每个选项含有：菜名、价格、点菜按钮。当点击点菜按钮时，使用 Toast 提示用户点菜成功，并将按钮显示为退点。当点击选项其他区域时，进入到菜品详细页面
  - 4) 使用 ActionBar 为 FoodView 添加选项菜单，菜单项为“已点菜品”、“查看订单”、“呼叫服务”
7. 在 es.source.code.activity 包下新建 Activity 类名为：FoodOrderView，为该 FoodOrderView 定制布局 food\_order\_view.xml，要求如下：
- 1) 在该页面上使用 ViewPager&TabLayout 显示两个选项卡，分别为“未下单菜”和“已下单菜”
  - 2) 支持屏幕左右滑动导航和导航项点击导航，当用户导航至“已下单菜”，在 FoodOrderView 页面上方使用 ListView 或 RecyclerView 显示订单中的菜品项，每项中显示：菜品、价格、数量、备注。在页面下方显示菜品总数、订单总价及“结账”按钮
  - 3) 当用户导航至“未下单菜”，在 FoodOrderView 页面上方使用 ListView 或 RecyclerView 显示已点但未生成订单菜品，没想显示：菜品、价格、数量、备注、“退点”按钮；在页面下方显示菜品总数、

订单总价及“提交订单”按钮。

8. 在 `es.source.code.activity` 包下新建 Activity 类名为: `FoodDetailed`, 为该 `FoodDetailed` 定制布局 `food_detailed.xml`, 要求如下:
  - 1) 在该布局中显示菜品信息: 菜品图片, 菜名, 菜价, 菜品备注输入框
  - 2) 在该布局中添加一个按钮, 按钮标签根据当前菜品是否已在已点菜品中, 显示不同标签: 如果是已点菜, 则按钮显示“退点”, 否则显示“点菜”
  - 3) `FoodDetailed` 支持左/右滑动切换菜品。当用户在当前屏幕左/右滑动时, 当前屏幕显示不同的菜品详细信息
9. 修改 `FoodOrderView`、`FoodView` 代码, 在其中添加 User 变量 `user`
10. 修改 `MainScreen` 代码, 完成如下功能:
  - 1) 当用户点击“点菜”时, 页面跳转至 `FoodView`, 并使用 `Intent` 将 `user` 作为数据传给 `FoodView` 的 `user` 变量
  - 2) 当用户点击“查看订单”时, 页面跳转至 `FoodOrderView`, 并使用 `Intent` 将 `user` 作为数据传给 `FoodOrderView` 的 `user` 变量
11. 修改 `FoodView` 代码, 当用户点击食物列表项时, 页面跳转至 `FoodDetailed`; 当用户点击 `ActionBar` “已点菜品”时, 页面跳转至 `FoodOrderView`, 并默认显示为“未下单菜”选项页面, 传入 `user` 变量; 当用户点击 `ActionBar` “查看订单”时, 页面跳转至 `FoodOrderView`, 并默认显示为“已下单菜”选项页面, 传入 `user` 变量
12. 修改 `FoodOrderView` 代码, 当用户点击“结账”按钮时, 当 `user` 不为 `null` 时, 判断 `user` 中的 `oldUser` 值; 如果为 `True`, 则使用 `Toast` 提示“您好, 老顾客, 本次你可享受 7 折优惠”; 否则, 不作提示
13. 请根据以上任务, 自主添加所需数据结构和功能代码, 适当扩展功能实现, 完成 SCOS 的基本原型开发, 并调试正确运行
14. 修改 SCOS 工程的 `AndroidManifest.xml` 中 `versionCode` 和 `versionName` 为 2.0, 打包发布 `SCOS2.0.apk`, 并用真机测试运行

## 二、 实现和证明

1. 由于修改了 mainScreen 的布局,需要对 mainScreen 中的方法进行重构,首先设置一个 boolean 值 canBeVisible 决定点菜和查看订单两个按钮是否应该被显示,当该布尔值发生变化时调用函数来决定 GridView 适配器该如何初始化

```
/**
 * 根据canBeVisible来设置mStrings和mImages
 */
private void setMStringsAndMImages(){
    if(!canBeVisible){
        mStrings = Arrays.copyOfRange(Strings, from: 2, to: 4);
        mImages = Arrays.copyOfRange(images, from: 2, to: 4);
        ORDER = -1;
        WATCH_ORDER = -1;
        LOGIN = 0;
        HELP = 1;
    }else{
        mStrings = Arrays.copyOf(Strings,Strings.length);
        mImages = Arrays.copyOf(images,images.length);
        ORDER = 0;
        WATCH_ORDER = 1;
        LOGIN = 2;
        HELP = 3;
    }
    initGridViewAdapter();
}
```

初始化 GridViewAdapter 时使用了一个字符串数组和一个存放图片 id 的 int 数组。在初始化监听器时,为 GridView 设置了一个 OnItemClickListener。在这里设置时要注意,当 GridView 的内部部件为 Button 时,会涉及到一个焦点抢占的问题,当你点击了 GridView 的一个 item 时,此时即便设置了监听器,也不会相应我们预设的程序,这是因为 item 中的 Button 把焦点给抢夺了即,监听器没有获得到 click 事件,所以应当在 gridview 的 item 布局中加上一句 clickable = “false” 这样就可以使监听器正常运行。

```
private void initGridViewListener(){
    //为GridView设置click监听器
    mGridView.setOnItemClickListener((parent, view, position, id) → {
        if(position == LOGIN){
            turnToLogin();
        }
    });
}
```

```
/**
 * 初始化GridView组件
 */
private void initGridViewAdapter(){
    mGridView = findViewById(R.id.main_screen_grid);

    //为GridView组件设置适配器
    mGridView.setAdapter(new BaseAdapter() {
        @Override
        public int getCount() {
            return mStrings.length;
        }

        @Override
        public Object getItem(int position) {
            return mImages[position];
        }

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

        @Override
        public View getView(int position, View convertView, ViewGroup parent) {
            View view = getLayoutInflater().inflate(R.layout.main_screen_grid_item, root: null);
            Button btn = view.findViewById(R.id.main_screen_grid_item_button);
            btn.setText(mStrings[position]);
            Drawable drawable = getResources().getDrawable(mImages[position]);

            //如果不设置bound图片将不会显示
            drawable.setBounds( left: 0, top: 0, right: 100, bottom: 100);
            btn.setCompoundDrawables( left: null, drawable, right: null, bottom: null);
            return view;
        }
    });
}
```

在设置了 gridview 布局之后，界面如下图所示。



2,3. 新建一个 package:es.source.code.model 在包下新建一个 User.java, 代码如下所示。

```
package es.source.code.model;

public class User {

    private String userName;
    private String password;
    private boolean oldUser;

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public boolean isOldUser() {
        return oldUser;
    }

    public void setOldUser(boolean oldUser) {
        this.oldUser = oldUser;
    }
}
```

4. 修改之前的代码, 在之前的判断用户名和密码正确后添加代码新建 User 对象, 并且使用 Intent 传回到 MainScreen 中。注册按钮则使用检测登录按钮是否可用的方法来判断是否允许注册, 判断成功后操作同之前。

```
//当注册按钮被按下时
if(login.isEnabled()) {
    User loginUser = new User();
    loginUser.setUserName(user_name.getText().toString());
    loginUser.setPassword(password.getText().toString());
    loginUser.setOldUser(false);
    Bundle bundle = new Bundle();
    bundle.putSerializable("user", loginUser);
    bundle.putString("data", "RegisterSuccess");
    mIntent.putExtras(bundle);
    setResult(Activity.RESULT_OK, mIntent);
    Toast.makeText(context: LoginOrRegister.this, text: "注册成功", Toast.LENGTH_SHORT).show();
    finish();
}
```

5. 首先判断通过 Intent 传回来的 bundle 对象是否为空，如果不为空则去除 bundle 中的 String 对象，根据条件判断，如果是 LoginSuccess，将 user 赋值为 bundle 中的对象。如果是 RegisterSuccess，则提示欢迎称为 SCOS 新用户，同时把 canBeVisible 标志设置为 true 之后初始化各个按钮。如果得到的不是按要求的字符串则把 user 赋值为空。

```
Bundle bundle = data != null ? data.getExtras() : null;
if(bundle != null){
    String message = bundle.getString( key: "data");
    if(message.equals("RegisterSuccess")){
        Toast.makeText( context: MainScreen.this, text: "欢迎您成为SCOS新用户", Toast.LENGTH_SHORT).show();
        user = (User)bundle.getSerializable( key: "user");
        canBeVisible = true;
    }else if(message.equals("LoginSuccess")){
        user = (User)bundle.getSerializable( key: "user");
        canBeVisible = true;
    }else{
        user = null;
        canBeVisible = false;
    }
    setMStringsAndMImages();
}
```

6. 在 FoodView 中，定义一个适配器类继承 PagerAdapter, 在使用 setAdapter 方法为 ViewPager 设置一个适配器。

```
/**
 * 自定义的ViewPager适配器
 */
private class MyAdapter extends PagerAdapter{

    @Override
    public int getCount() {
        return mViewList.size();
    }

    @Override
    public boolean isViewFromObject(@NonNull View view, @NonNull Object o) {
        return view == o;
    }

    @NonNull
    @Override
    public Object instantiateItem(@NonNull ViewGroup container, int position) {
        container.addView(mViewList.get(position));
        return mViewList.get(position);
    }

    @Override
    public void destroyItem(@NonNull ViewGroup container, int position, @NonNull Object object) {
        container.removeView(mViewList.get(position));
    }
}
```

```
/**
 * 初始化ViewPager
 */
private void initTabViewPager(){
    mViewPager.setAdapter(new MyAdapter());

    mViewPager.addOnPageChangeListener((SimpleOnPageChangeListener) onPageSelected(position) -> {
        mTabLayout.getTabAt(position).select();
    });
}
```



在此次定义中一个有四个 PageView, 分别是“冷菜”, “热菜”, “海鲜”, “酒水”, 在 PageView 中使用 ListView 填充, 同样也使用 Adapter 对 ListView 初始化, 注意要在使用适配器初始化时, 对填充的 item 里的 button 要设置 OnClickListener 来实现 button 和 item 的不同点击事件, 在使用 setOnItemClickListener 来对 item 实现监听。设置完以上之后还要再 listView 布局中添加  
android:touchscreenBlocksFocus="true", 用来防止 item 中的 button 控件对焦点的抢占。对 ListView 的初始化如下图所示。

```
/**
 * 初始化热菜的列表
 */
private void initHotFoodList(){
    hotFoodListView = hotFoodView.findViewById(R.id.hot_food_list_view);

    hotFoodListView.setOnItemClickListener((parent, view, position, id) -> {
        String message = "点击第"+position+"项";
        Toast.makeText( context: FoodView.this,message,Toast.LENGTH_SHORT).show();
    });

    hotFoodListView.setAdapter(new BaseAdapter() {
        @Override
        public int getCount() {
            return 10;
        }

        @Override
        public Object getItem(int position) {
            String str = "烤肉";
            return str;
        }

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

        @Override
        public View getView(int position, View convertView, ViewGroup parent) {
            if(convertView == null){
                convertView = getLayoutInflater().inflate(R.layout.order_item, parent, attachToRoot: false);
            }

            ImageView imageView = convertView.findViewById(R.id.order_item_image);
            imageView.setImageResource(R.drawable.logo);

            TextView textView_name = convertView.findViewById(R.id.order_item_name);
            textView_name.setText("烤肉");

            TextView textView_price = convertView.findViewById(R.id.order_item_price);
            textView_price.setText("100元");

            final Button button = convertView.findViewById(R.id.order_item_button);
            button.setText(ORDER);
            initViewPageItemButton(button);

            return convertView;
        }
    });
}
```

完成上述步骤以后再对当前的布局实现 ActionBar, 首先需要在 AndroidManifest.xml 中的 foodView 的 activity 那一部分加上主题为  
"@style/Theme.AppCompat.Light.NoActionBar"防止现有主题对

ActionBar 的干扰，再在 FoodView 的布局最上方添加 ActionBar 的 xml 代码，同时主类中重载 onCreateOptionsMenu 和 onOptionsItemSelected 方法，其中 R.menu.food\_view\_menu 为 ActionBar 中包含的属性（已点菜品，查看订单，呼叫服务）。目前没有对这些属性实现相关功能，使用 Toast 来实现反馈。

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.food_view_menu, menu);
    return super.onCreateOptionsMenu(menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.food_view_already_ordered: Toast.makeText(context, FoodView.this, text: "已点菜单", Toast.LENGTH_SHORT).show(); return true;
        case R.id.food_view_watch_ordered: Toast.makeText(context, FoodView.this, text: "查看菜单", Toast.LENGTH_SHORT).show(); return true;
        case R.id.food_view_call_for_service: Toast.makeText(context, FoodView.this, text: "呼叫服务", Toast.LENGTH_SHORT).show(); return true;
        default: return super.onOptionsItemSelected(item);
    }
}
```

最后结果如下图所示：



7. 新建 Activity: FoodOrderView 后，按照第六步的方法也为该界面添加 ViewPager 和 TabLayout 组合，在 TabLayout 实现的接口

OnTabSelectedListener 中的方法 onTabSelected 中添加当界面变换时对结账和提交订单按钮的改变。

```
@Override
public void onTabSelected(TabLayout.Tab tab) {
    mViewPager.setCurrentItem(tab.getPosition());
    if(tab.getPosition() == NOTORDER){
        mButton.setText("提交订单");
        mButton.setBackgroundColor(Color.parseColor( colorString: "#3498DB"));
    }else if(tab.getPosition() == ORDER){
        mButton.setText("结账");
        mButton.setBackgroundColor(Color.parseColor( colorString: "#2ECC71"));
    }
}
```



8. 新建 Activity: FoodDetailed, 按照需求, 应当将布局设置为 PageViewer 布局, 并且在上一个布局中即 FoodView 中实现一个 Intent 传入相关数据, 例如: 当前点击的时那个布局 (热菜、凉菜、海鲜还是饮料)、当前点击的时第几个 item、当前布局中的点菜按钮点击情况, 将这些信息统统处理后使用 Intent 传到目标 Activity 中, 在实现恢复布局。

```

public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
    Intent intent = new Intent( packageContext: FoodView.this,FoodDetailed.class);
    String str = "";
    String tag = "";
    if(parent == hotFoodListView){
        str = "hotFood";
        tag = tagHotFood.toString();
    }
    else if(parent == coldFoodListView){
        str = "coldFood";
        tag = tagColdFood.toString();
    }
    else if(parent == seeFoodListView){
        str = "seeFood";
        tag = tagSeeFood.toString();
    }
    else if(parent == drinkListView){
        str = "drink";
        tag = tagDrink.toString();
    }
    intent.putExtra( name: "data",str);
    intent.putExtra( name: "pos",position);
    intent.putExtra( name: "tag",tag);
    startActivity(intent);
}

```

其中 str 表示当前时点击了什么视图，tag 表示各个 item 的点菜按钮点击情况，pos 表示当前点击的时第几个 item，将这些信息通过 Intent 传递。在冲另一个 Activity 中接受。

```

mIntent = getIntent();
String str = mIntent.getStringExtra( name: "data");
int pos = mIntent.getIntExtra( name: "pos", defaultValue: 0);
String tag = mIntent.getStringExtra( name: "tag");

```

接受之后根据这些数据初始化 ViewPager 的适配器。

```

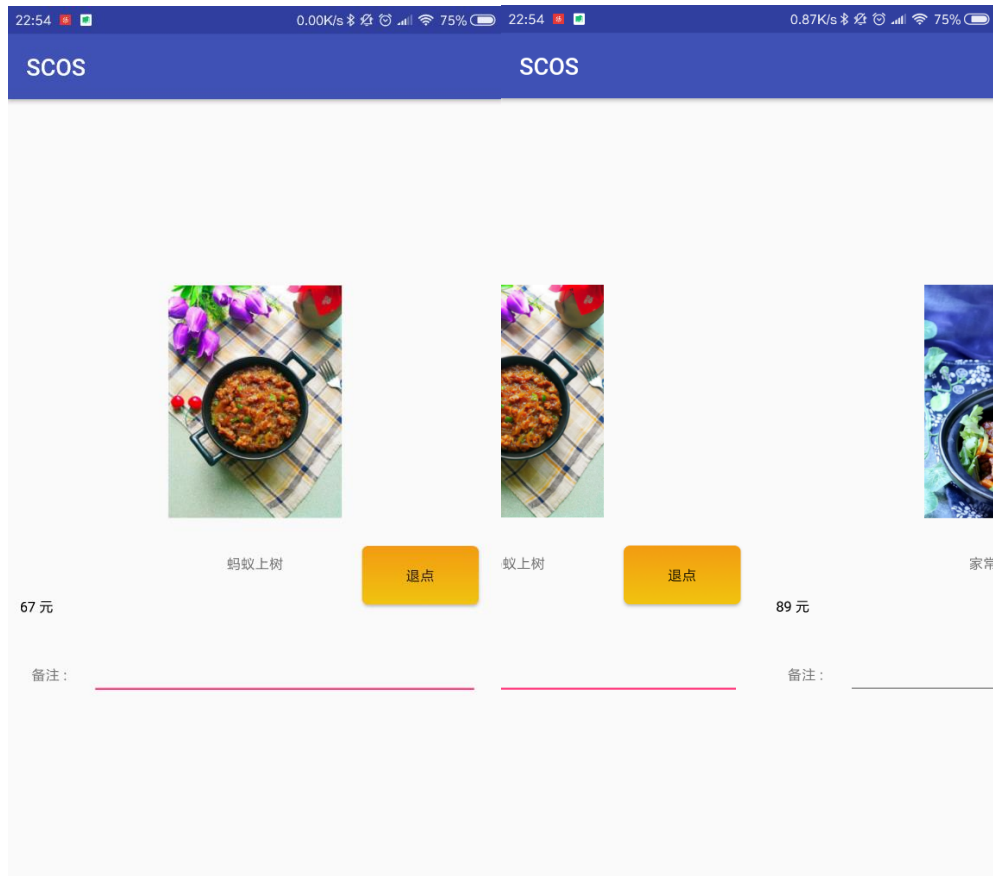
private void initViewList(String[] names,int[] prices,int[] images,String tag){
    for(int i = 0 ;i<names.length;i++){
        final View mView = getLayoutInflater().inflate(R.layout.food_detail_view, root: null);
        final Button button= mView.findViewById(R.id.food_detail_button);
        ImageView imageView = mView.findViewById(R.id.food_detail_image);
        TextView textView_name = mView.findViewById(R.id.food_detail_name);
        TextView textView_price = mView.findViewById(R.id.food_detail_price);
        imageView.setImageResource(images[i]);
        textView_name.setText(names[i]);
        textView_price.setText(""+prices[i]+" 元");
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if(button.getText().toString().equals("退点")){
                    button.setText("点菜");
                    button.setTextColor(Color.BLACK);
                }else if(button.getText().toString().equals("点菜")){
                    button.setText("退点");
                    button.setTextColor(Color.RED);
                }
            }
        });
        if(tag.charAt(i) == '1'){
            button.setText("退点");
        }else if(tag.charAt(i) == '0'){
            button.setText("点菜");
        }
        mViewList.add(mView);
    }
}

```

其中 names 为菜品名，prices 为价格，images 为图片 id，tag 为按钮的点菜情况。前三个信息被我同一放在另一个类 FoodItem.java 中。

```
public class FoodItems {  
    public static String[] hot_food_name = {"香辣牛肉西兰花", "蚂蚁上树", "家常干锅排骨", "菠萝咕咕肉", "家常口水鸡", "地三鲜", "干贝边炖豆腐", "鱼香肉丝", "水煮肉片", "铁锅芋头鸡"};  
    public static int[] hot_food_image = {R.drawable.hot_food_1,R.drawable.hot_food_2,R.drawable.hot_food_3,R.drawable.hot_food_4,R.drawable.hot_food_5,  
        R.drawable.hot_food_6,R.drawable.hot_food_7,R.drawable.hot_food_8,R.drawable.hot_food_9,R.drawable.hot_food_10};  
    public static int[] hot_food_price = {35,67,89,58,66,52,32,55,76,78};  
  
    public static String[] cold_food_name = {"蜜汁苦瓜", "皮蛋生菜卷", "墨鱼汁凉粉", "凉拌鲈鱼", "龙须菜拌花生", "香辣手撕鸡", "红油金针菇"};  
    public static int[] cold_food_image = {R.drawable.cold_food_1,R.drawable.cold_food_2,R.drawable.cold_food_3,R.drawable.cold_food_4,R.drawable.cold_food_5,  
        R.drawable.cold_food_6,R.drawable.cold_food_7};  
    public static int[] cold_food_price = {15,23,17,47,25,36,14};  
  
    public static String[] see_food_name = {"香辣肉蟹煲", "美味蒜香烤生蚝", "啤酒烧鱿鱼", "椒盐皮皮虾", "海带芹菜", "辣花甲"};  
    public static int[] see_food_image = {R.drawable.see_food_1,R.drawable.see_food_2,R.drawable.see_food_3,R.drawable.see_food_4,R.drawable.see_food_5,R.drawable.see_food_6};  
    public static int[] see_food_price = {99,73,102,90,87,40};  
  
    public static String[] drink_name = {"可乐", "啤酒", "雪碧", "椰汁", "果粒橙"};  
    public static int[] drink_image = {R.drawable.drink_1,R.drawable.drink_2,R.drawable.drink_3,R.drawable.drink_4,R.drawable.drink_5};  
    public static int[] drink_price = {4,3,4,8,5};  
}
```

初始化完毕后，在使用 mViewPager.setCurrentItem(pos); 语句来设置当前显示的 View.



9, 10. 修改相应代码，修该如下

```
private void turnToFoodView(){  
    Intent intent_toFoodView = new Intent( packageContext: MainScreen.this, FoodView.class);  
    intent_toFoodView.putExtra( name: "user",user);  
    startActivity(intent_toFoodView);  
}
```

添加 putExtra(), 把 user 放入 intent 传递过去。

```

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.food_view_already_ordered: Toast.makeText( context: FoodView.this, text: "已点菜单", Toast.LENGTH_SHORT).show(); return true;
        case R.id.food_view_watch_ordered: {
            Intent intent = new Intent( packageContext: FoodView.this, FoodOrderView.class);
            intent.putExtra( name: "user", user);
            startActivity(intent);
        } return true;
        case R.id.food_view_call_for_service: Toast.makeText( context: FoodView.this, text: "呼叫服务", Toast.LENGTH_SHORT).show(); return true;
        default: return super.onOptionsItemSelected(item);
    }
}

```

当点击 ActionBar 中的条目时，响应相应的操作。

11. 之前在设置 setOnItemClickListener 的时候就已经实现了点击食物列表项时可以进入 FoodDetailed 界面。在点击 ActionBar 的内容时，只需要在重载的 onOptionsItemSelected 方法中实现即可。

```

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.food_view_already_ordered: Toast.makeText( context: FoodView.this, text: "已点菜单", Toast.LENGTH_SHORT).show(); return true;
        case R.id.food_view_watch_ordered: {
            Intent intent = new Intent( packageContext: FoodView.this, FoodOrderView.class);
            intent.putExtra( name: "user", user);
            startActivity(intent);
        } return true;
        case R.id.food_view_call_for_service: Toast.makeText( context: FoodView.this, text: "呼叫服务", Toast.LENGTH_SHORT).show(); return true;
        default: return super.onOptionsItemSelected(item);
    }
}

```

其中 user 为传入的 User 对象，pos 为传入的默认要显示的界面，0 代表未下单菜，1 代表已下单菜。

12. 在 FoodOesweView 中对 Button 按钮进行设置监听，具体为当 button 中的内容为结账时就使用 Toast 进行提示。

```


@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.food_view_already_ordered: Toast.makeText( context: FoodView.this, text: "已点菜单", Toast.LENGTH_SHORT).show(); return true;
        case R.id.food_view_watch_ordered: {
            Intent intent = new Intent( packageContext: FoodView.this, FoodOrderView.class);
            intent.putExtra( name: "user", user);
            startActivity(intent);
        } return true;
        case R.id.food_view_call_for_service: Toast.makeText( context: FoodView.this, text: "呼叫服务", Toast.LENGTH_SHORT).show(); return true;
        default: return super.onOptionsItemSelected(item);
    }
}

```

13, 14. 增加了后续代码，使得未提交订单和已经提交订单能够和之前点菜的菜品项目匹配。

13:570.00K/s100%

SCOS




香辣牛肉西兰花

35

数量

1 份



蚂蚁上树

67

数量

1 份

菜品总数2

菜品总价102

提交订单

菜品总数0

SCOS: 提交订单成功

提交订单

未下单菜

已下单菜

13:570.05K/s100%

SCOS

菜品总数0

SCOS: 提交订单成功

提交订单

未下单菜

已下单菜

13:570.04K/s100%

SCOS



香辣牛肉西兰花

35

数量

1 份



蚂蚁上树

67

数量

1 份

菜品总数2

菜品总价102

结账

未下单菜

已下单菜

### 三、 结论

在本次作业中熟悉了各种布局的使用例如 GridView, ViewPager 和 Tablayout 的联合使用, 以及为各个视图添加适配器的方法还有 ActionBar 的使用等等, 以及在各个 Activity 之间传递 user 对象的方法, 首先使得 user 实现 Serializable 方法这样就可以使用 intent 来传递了。



## 四、 参考文献

1. [Android 基本控件之 GridView](#)
2. [Flat UI](#)
3. [Add Action Buttons](#)
4. [美食杰](#)