



LIST & TAB VIEW

Today's Overview

1

- Creating Fancier ListView

2

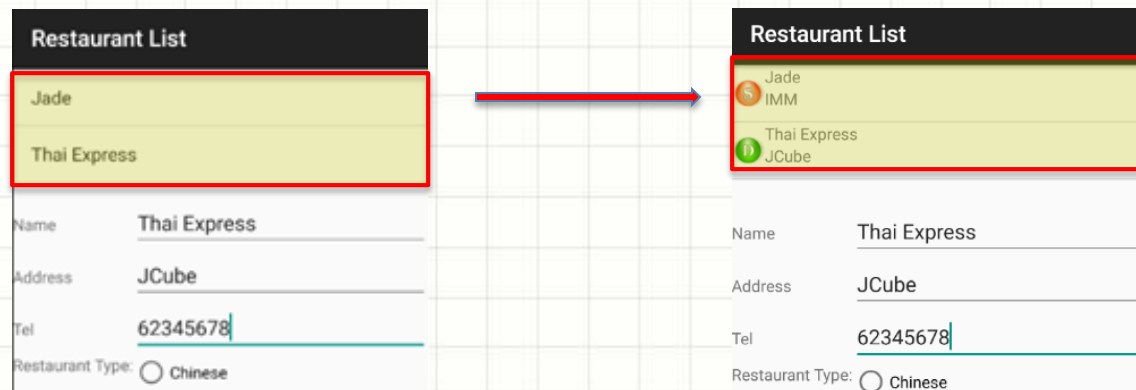
- Creating Tabs

Fancier ListView



Fancier ListView

- Instead of a simple row display in a *ListView*, it can be customized to enhanced the look. As shown below, the new row display comprises restaurant name and address of the restaurant, plus an image icon



Fancier ListView



- Let's check what do we need to modify from the previous exercise to incorporate the new row display?
 - ☐ **Model** - Any change in Data Model?
 - ☐ **View** - Do you need to modify any of the user interface view?
 - ☐ **Controller** - Do you need to tell the Controller to do any thing new?

Fancier ListView



☐ Model → NO

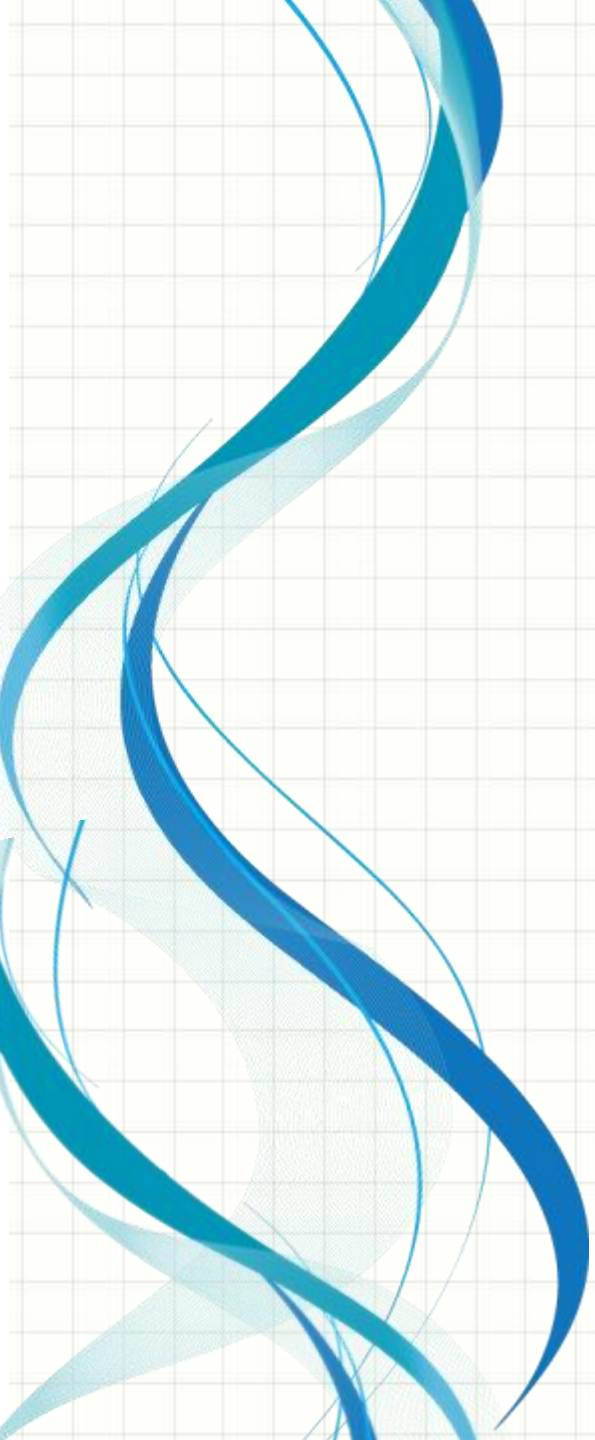
No change in *ArrayList* with Restaurant Data Model

☐ View → YES

New row layout in *ListView* is needed. We will create a *row.xml* layout

☐ Controller → YES

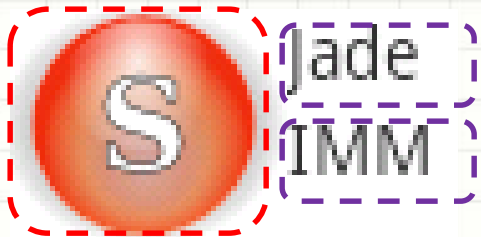
New customized Adapter to inflate the new row display in the *ListView*. We create a subclass *RestaurantAdapter* of *ArrayAdapter* to control the row design and populate row data from the *ArrayList*



View - New Row Layout

Row Layout

- The new row layout *row.xml* will contain
 - one *ImageView* widget for image icon
 - two *TextView* widgets for name and address



```
<LinearLayout android:orientation="horizontal">  
  <ImageView android:id="@+id/icon"/>  
  <LinearLayout android:orientation="vertical">  
    <TextView android:id="@+id/restName"/>  
    <TextView android:id="@+id/restAddr"/>  
  </LinearLayout>  
</LinearLayout>
```

Basic skeleton layout for row.xml

Row Layout

- *row.xml* Listing

```
1      <?xml version="1.0" encoding="utf-8"?>
2      <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3          xmlns:app="http://schemas.android.com/apk/res-auto"
4          android:layout_width="match_parent"
5          android:layout_height="wrap_content"
6          android:orientation="horizontal">
7
8          <ImageView
9              android:layout_width="wrap_content"
10             android:layout_height="wrap_content"
11             app:srcCompat="?android:attr/listChoiceIndicatorSingle"
12             android:id="@+id/icon"
13             android:layout_weight="1" />
14
15         <LinearLayout
16             android:orientation="vertical"
17             android:layout_width="match_parent"
18             android:layout_height="wrap_content"
19             android:layout_weight="1">
20
21             <TextView
22                 android:text="TextView"
23                 android:layout_width="match_parent"
24                 android:layout_height="wrap_content"
25                 android:id="@+id/restName" />
26
27             <TextView
28                 android:text="TextView"
29                 android:layout_width="match_parent"
30                 android:layout_height="wrap_content"
31                 android:id="@+id/restAddr" />
32         </LinearLayout>
33     </LinearLayout>
```



Controller – Customized Adapter

Customized Adapter

- In Practical 1, you have learnt that a *ListView* inflates its rows with the *ArrayList* of Restaurant Data Model through *ArrayAdapter*
- New Customized *Adapter* (a sub-class of *ArrayAdapter* class) is implemented to incorporate the *row.xml* layout with the *ArrayList*

```
List<Restaurant> model = new ArrayList<Restaurant>();
```

```
class RestaurantAdapter extends ArrayAdapter<Restaurant> {  
    RestaurantAdapter() {  
        super(RestaurantList.this, R.layout.row, model);  
    }  
}
```

Customized Adapter

- getView() method is responsible for creating the individual rows of *ListView*
- The method determines the layout and the data assignment of the row

```
public View getView(int position, View convertView, ViewGroup parent) {  
    View row = convertView;  
    RestaurantHolder holder = null;  
  
    if (row == null) {  
        LayoutInflater inflater = getLayoutInflater();  
  
        row = inflater.inflate(R.layout.row, parent, false);  
        holder = new RestaurantHolder(row);  
        row.setTag(holder);  
    } else {  
        holder = (RestaurantHolder) row.getTag();  
    }  
    holder.populateFrom(model.get(position));  
  
    return (row);  
}
```

Customized Adapter

- If user scrolls the list, certain rows will not be visible anymore in *ListView*
- Android recycles rows which are not displayed anymore and allow these rows to be reused
- A performance optimized adapter assigns the new content to the recycled row. Only updating the content of the row avoids loading the *row.xml* XML layout

Customized Adapter

- If *ListView* has a recycled row, it passes these rows to `getView()` method as `convertView` parameter
- `convertView` may be `NULL` if there is no row to recycle

```
public View getView(int position, View convertView, ViewGroup parent) {  
    View row = convertView;  
    RestaurantHolder holder = null;  
  
    if (row == null) {  
        LayoutInflater inflater = getLayoutInflater();  
  
        row = inflater.inflate(R.layout.row, parent, false);  
        holder = new RestaurantHolder(row);  
        row.setTag(holder);  
    } else {  
        holder = (RestaurantHolder) row.getTag();  
    }  
    holder.populateFrom(model.get(position));  
  
    return (row);  
}
```

→ No recycled row

→ Recycled row available

View Holder

- The `findViewById()` method is an “expensive” operation, therefore a “View Holder” (*RestaurantHolder*) is created in *Adapter* to hold references to the widgets (restName, restAddress and icon) in the row layout
- This is faster than the repetitive call of the `findViewById()` method

View Holder

- This “View Holder” is attached to the row via **setTag()** method. If row is recycled, the “View Holder” can be called via **getTag()** method

```
public View getView(int position, View convertView, ViewGroup parent) {  
    View row = convertView;  
    RestaurantHolder holder = null;  
  
    if (row == null) {  
        LayoutInflater inflater = getLayoutInflater();  
  
        row = inflater.inflate(R.layout.row, parent, false);  
        holder = new RestaurantHolder(row);  
        row.setTag(holder);  
    } else {  
        holder = (RestaurantHolder) row.getTag();  
    }  
    holder.populateFrom(model.get(position));  
  
    return (row);  
}
```

View Holder

- *RestaurantHolder* Class Listing

```
static class RestaurantHolder {
    private TextView restName = null;
    private TextView addr = null;
    private ImageView icon = null;

    RestaurantHolder(View row) {
        restName = (TextView) row.findViewById(R.id.restName);
        addr = (TextView) row.findViewById(R.id.restAddr);
        icon = (ImageView) row.findViewById(R.id.icon);
    }

    void populateFrom(Restaurant r) {
        restName.setText(r.getName());
        addr.setText(r.getAddress() + ", " + r.getTelephone());

        if (r.getRestaurantType().equals("Chinese")) {
            icon.setImageResource(R.drawable.ball_red);
        } else if (r.getRestaurantType().equals("Western")) {
            icon.setImageResource(R.drawable.ball_yellow);
        } else {
            icon.setImageResource(R.drawable.ball_green);
        }
    }
}
```

Get different icon if the restaurant's type is different

Binding

- With the customized adapter (*RestaurantAdapter*) ready, it is time to bind the *ListView* to new *ArrayAdapter* -> *RestaurantAdapter* using *setAdapter* method

```
list = (ListView) findViewById(R.id.restaurants);  
adapter = new RestaurantAdapter();  
list.setAdapter(adapter);
```



SPLITTING THE VIEWS

Tabs

- In second part of Practical 2 exercise, the *ListView* and Restaurant Form are split and displayed into two separated tabs (“List” and “Details”)

Restaurant List

Jade
\$ IMM

Thai Express
D JCube

Name Thai Express

Address JCube

Tel 62345678

Restaurant Type: ☐ Chinese

This screenshot shows the initial state of the app. The top section displays a list of restaurants. The bottom section contains a form for viewing details, with fields for Name, Address, Tel, and Restaurant Type. Red brackets on the right side of the form fields indicate that these fields are mapped to the 'DETAILS' tab.

Restaurant List

LIST DETAILS

Jade
\$ IMM

Thai Express
D JCube

Restaurant List

LIST DETAILS

Name Thai Express

Address JCube

Tel 62345678

Restaurant Type: ☐ Chinese

This block contains two screenshots of the app. The top screenshot shows the 'LIST' tab selected, with the 'LIST' label circled in red. The bottom screenshot shows the 'DETAILS' tab selected, with the 'DETAILS' label circled in red. Red dashed arrows connect the form fields from the first screenshot to the corresponding fields in the second screenshot, illustrating the transition between the two views.

Tabs

- Let's check what do we need to modify from the previous exercise to split the views?
 - ☐ **Model** - Any change in Data Model?
 - ☐ **View** - Do you need to modify any of the user interface view?
 - ☐ **Controller** - Do you need to tell the Controller to do any thing new?

Tabs

❑ Model → NO

No change in *ArrayList* with Restaurant Data Model

❑ View → YES

Tab layout is needed. We will modify the *main.xml* layout to use *TabHost*, *TabWidget* and *FrameLayout* to create the new view

❑ Controller → YES

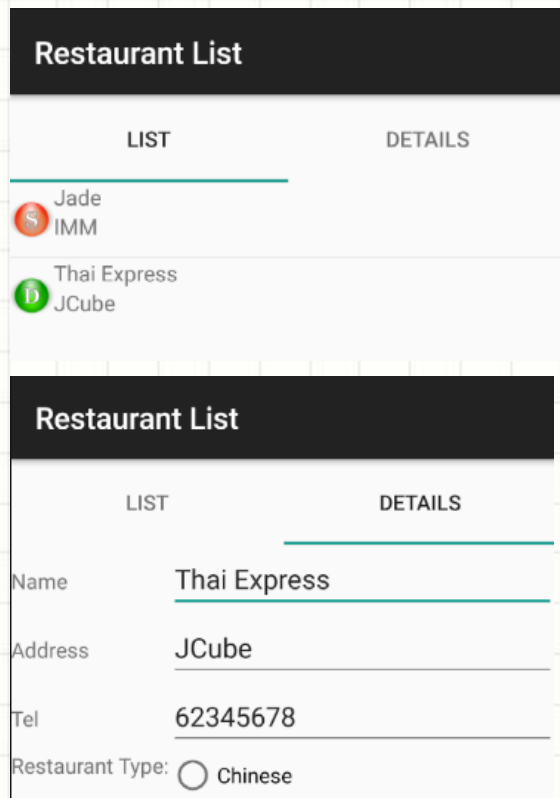
Change the Activity to *TabActivity* and create tabs for “List” and “Details” UI views



UI View - New Main Layout

Tab View

- Update the *main.xml* layout with *TabHost*, *TabWidget* and *FrameLayout* for the new layout



<TabHost>

<LinearLayout android:orientation="vertical">

<TabWidget />

<FrameLayout>

<LinearLayout android:id="@+id/restaurants_tab">

<ListView android:id="@+id/restaurants" />

</LinearLayout>

<LinearLayout android:id="@+id/details_tab">

<TableLayout android:id="@+id/details">

<TableRow>

</TableRow>

.....

</TableLayout>

</LinearLayout>

</FrameLayout>

</LinearLayout>

</TabHost>

Basic Skeleton of main.xml

Tab View

TabHost

- **Container** for a tabbed window view
- This object **holds two children**:
 - a set of **tab labels** that the user clicks to select a specific tab, and
 - a **FrameLayout object** that displays the contents of that page.

Tab View

TabHost

- *TabHost* is used to add labels, add the **callback handler**, and **manage callbacks**. For instance, switching the displayed page
- You might call this object to iterate the list of tabs, or to tweak the layout of the tab list

Tab View

TabWidget

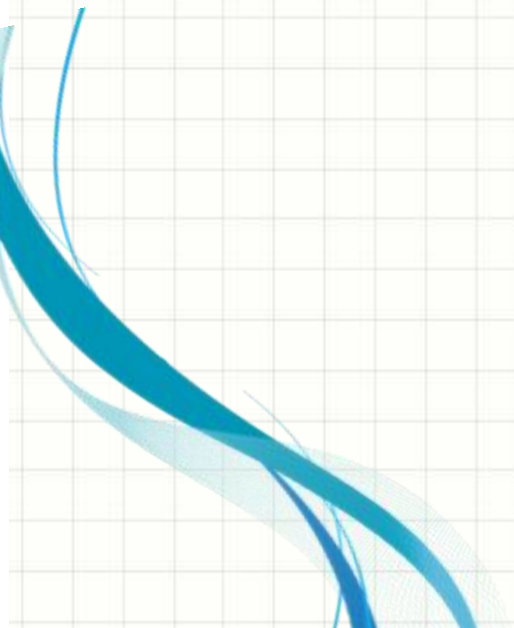
- Displays a list of tab labels representing each page in the `TabHost`'s collection
- When the user selects a tab, *TabWidget* sends a message to the parent container, *TabHost*, to tell it to switch the displayed page. You typically won't use many methods directly on this object.

Main Layout

- *main.xml* Listing (partial)

```
13      <TabHost
14          android:id="@+id/tabHost"
15          android:layout_width="match_parent"
16          android:layout_height="match_parent">
17
18      <LinearLayout
19          android:layout_width="match_parent"
20          android:layout_height="match_parent"
21          android:orientation="vertical">
22
23          <TabWidget
24              android:id="@android:id/tabs"
25              android:layout_width="match_parent"
26              android:layout_height="wrap_content" />
27
28          <FrameLayout
29              android:id="@android:id/tabcontent"
30              android:layout_width="match_parent"
31              android:layout_height="match_parent">
```

Controller – Setting up Tabs View



Controller

- Create the two tabs and load with different UI View content defined in *FrameLayout*

– restaurants (*ListView*) for ‘List’ tab

```
//Tab 1
TabHost.TabSpec spec = host.newTabSpec("List");
spec.setContent(R.id.restaurants_tab);
spec.setIndicator("List");
host.addTab(spec);
```

– restaurantDetails (*TableLayout*) for ‘Details’ tab

```
//Tab 2
spec = host.newTabSpec("Details");
spec.setContent(R.id.details_tab);
spec.setIndicator("Details");
host.addTab(spec);
```

```
<FrameLayout
    android:id="@android:id/tabcontent"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:id="@+id/restaurants_tab"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <ListView
            android:id="@+id/restaurants"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />
    </LinearLayout>

    <LinearLayout
        android:id="@+id/details_tab"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <TableLayout...>

        <Button...>
    </LinearLayout>

</FrameLayout>
```

Controller

Restaurant List

LIST

DETAILS

- Setup the title display (“LIST” and “DETAILS”) of each tab using *setIndicator* method
- Add the new tab to the *TabHost* using *addTab* method

```
spec.setIndicator("List");  
host.addTab(spec);
```

```
spec.setIndicator("Details");  
host.addTab(spec);
```

Controller

TabHost.TabSpec

- A tab has a tab indicator, content, and a tag that is used to keep track of it. This builder helps to choose among these options

Controller

TabHost.TabSpec

- For the tab indicator, your choices are
 - 1) set a label
 - 2) set a label and an icon

Controller

TabHost.TabSpec

- For the tab content, your choices are
 - 1) the **id of a View**
 - 2) a TabHost.TabContentFactory that creates the View content
 - 3) an Intent that launches an Activity

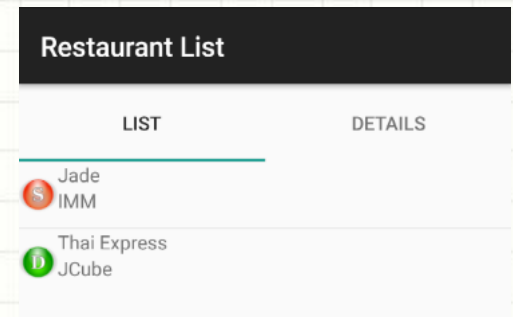
Controller

- After setting up the UI View to link to the Controller, setCurrentTab(int) method can be used to control the UI View to be displayed
 - host.setCurrentTab(0) to show *ListView*
 - host.setCurrentTab(1) to show **Restaurant form**



Controller – Detecting List Item Click

List Click



- Let's check what do we need to modify from the previous exercise to detect list item click?
 - ❑ **Model** - Any change in Data Model?
 - ❑ **View** - Do you need to modify any of the user interface view?
 - ❑ **Controller** - Do you need to tell the Controller to do any thing new?

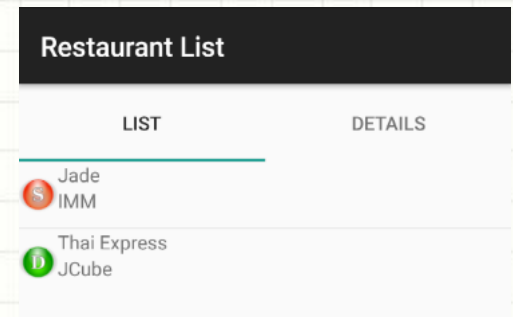
List Click

❑ Model - NO

❑ View - NO

❑ Controller - YES

Same as Button 'click', you need to implement an on **item event listener** to the **ListView**. When a 'click' on the list item, the Controller will activate the listener (*AdapterView* on item click listener) to handle the event



Controller

- Implement the Controller to capture the event generated by *ListView* when a row is selected using *setOnItemClickListener* method and pass the event to *onItemClickListener* (*ArrayView* onItemClick Listener)

```
list = (ListView) findViewById(R.id.restaurants);  
adapter = new RestaurantAdapter();  
list.setAdapter(adapter);
```

```
list.setOnItemClickListener(onItemClickListener);
```

Controller

- When a row is selected, the *ListView* selected event will be captured by the *AdapterView.OnItemClickListener* class and handle by `onItemClick()` method for the necessary action. In this case is to get the restaurant data from *ArrayList* Model to memory (restaurant Model) and transfer to the widgets (name, address, telephone and restaurantType) on the restaurant form

```
private AdapterView.OnItemClickListener onListClick = new AdapterView.OnItemClickListener() {  
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
```

Controller

- Respond to row selected event on *ListView*

```
private AdapterView.OnItemClickListener onListClick = new AdapterView.OnItemClickListener() {
```

```
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
```

```
        Restaurant r = model.get(position);
```

← Get data from ArrayList Model to restaurant Model

```
        restaurantName.setText(r.getName());
```

```
        restaurantAddress.setText(r.getAddress());
```

```
        restaurantTel.setText(r.getTelephone());
```

```
        if (r.getRestaurantType().equals("Chinese")) {
```

```
            restaurantTypes.check(R.id.chinese);
```

```
        } else if (r.getRestaurantType().equals("Western")) {
```

```
            restaurantTypes.check(R.id.western);
```

```
        } else if (r.getRestaurantType().equals("Indian")) {
```

```
            restaurantTypes.check(R.id.indian);
```

```
        } else if (r.getRestaurantType().equals("Indonesia")) {
```

```
            restaurantTypes.check(R.id.indonesian);
```

```
        } else if (r.getRestaurantType().equals("Korean")) {
```

```
            restaurantTypes.check(R.id.korean);
```

```
        } else if (r.getRestaurantType().equals("Japanese")) {
```

```
            restaurantTypes.check(R.id.japanese);
```

```
        } else {
```

```
            restaurantTypes.check(R.id.thai);
```

```
        }
```

```
        host.setCurrentTab(1);
```

```
    }
```

```
};
```

Transfer individual data element to widgets on UI View

← Switch the current UI display to "Details" tab



END