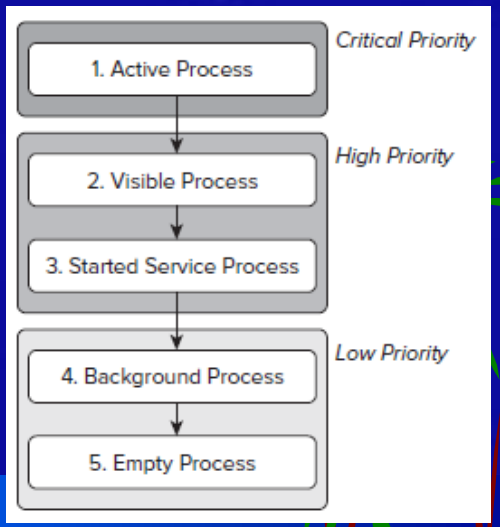
Android

Applications
Activities
Interface

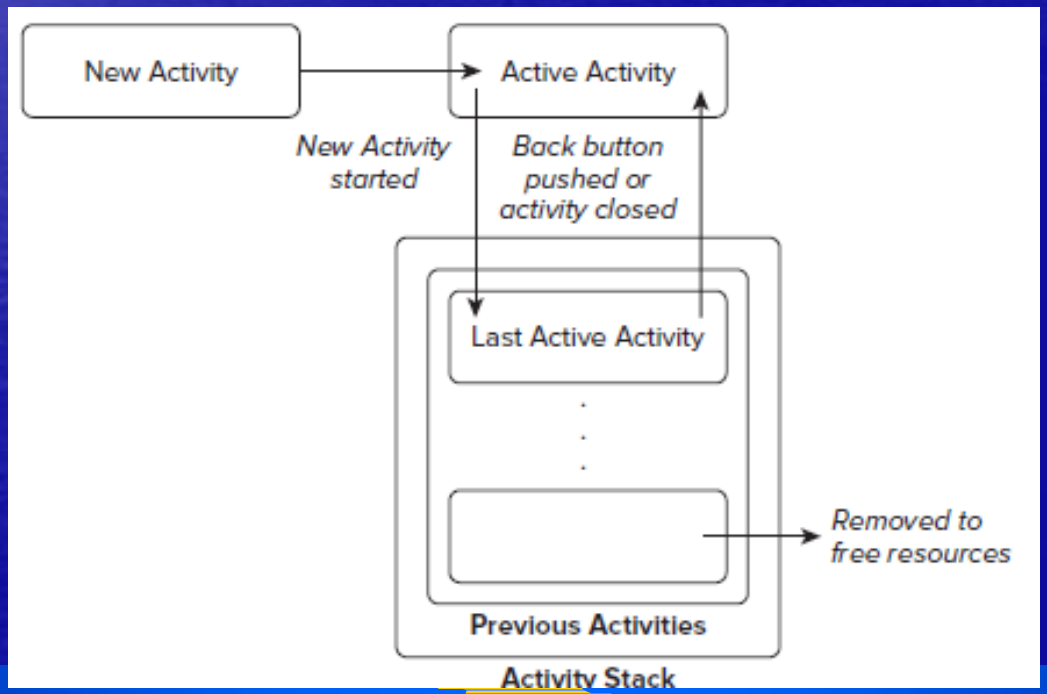
Applications

- **Processes**
  - Usually each application, including all its components, runs in a single process; the Application object from the framework is always in memory (singleton object)
- **Priorities**
  - Processes are organized in priority levels depending on the state of its components
  - When there is some lack of resources processes are automatically terminated
    - **Active** – In interaction, or components in execution invoked from processes in interaction
    - **Visible** – Visible under the interface from the process in interaction
    - **Background** – Without interaction

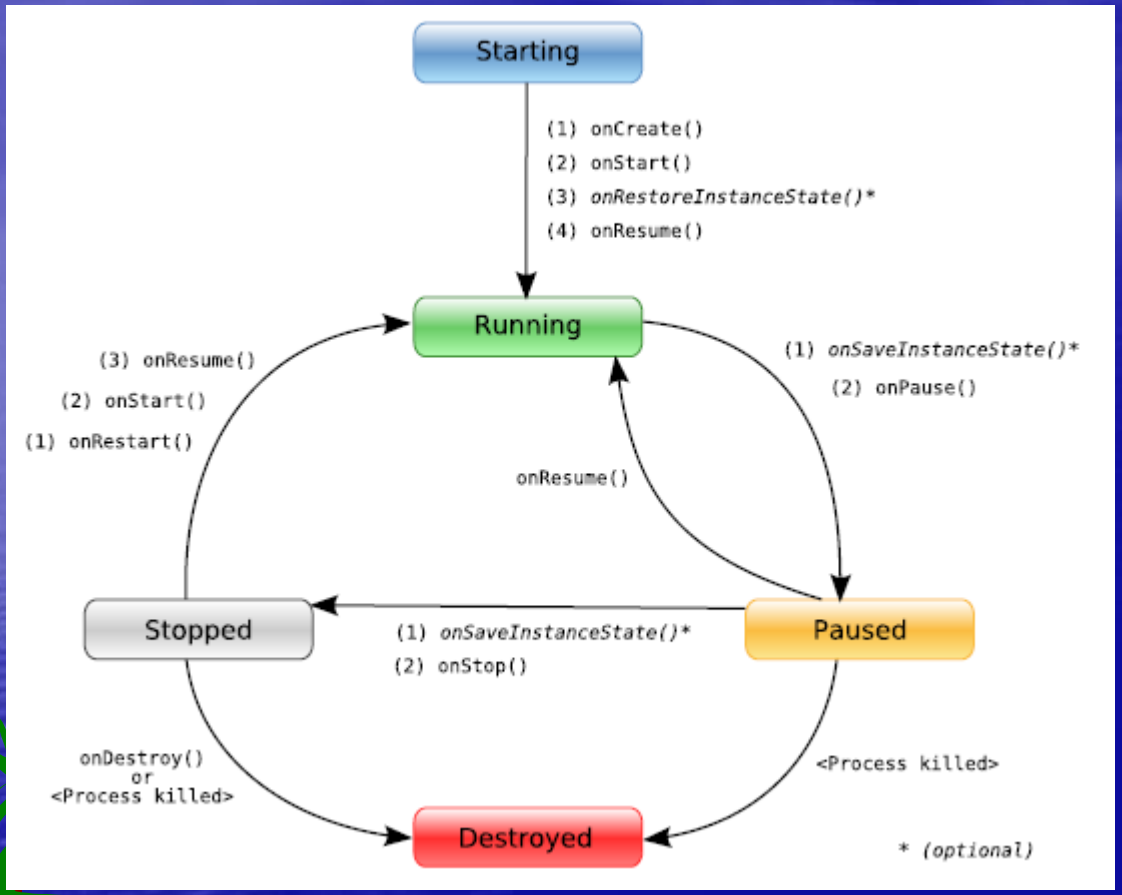


Activities

- **Activities**
  - Usually show an user interface (using ViewGroups and Views) and inherit from *android.app.Activity*
  - Execute a specific task in the application
  - Android maintains a *stack* of activities



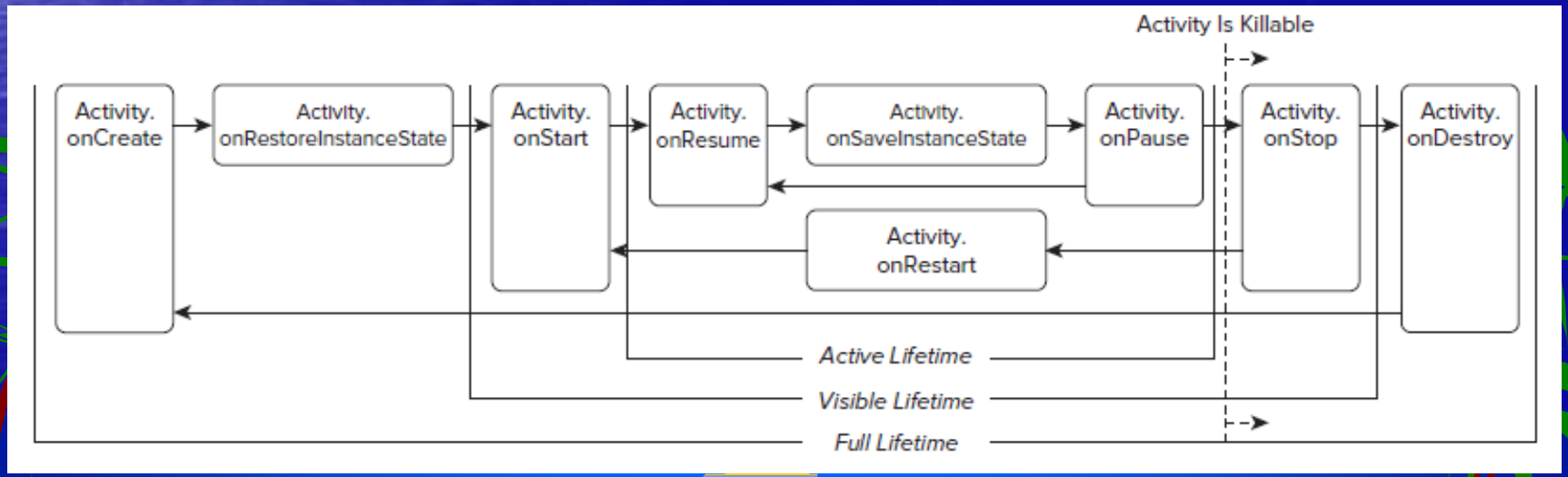
Activity states



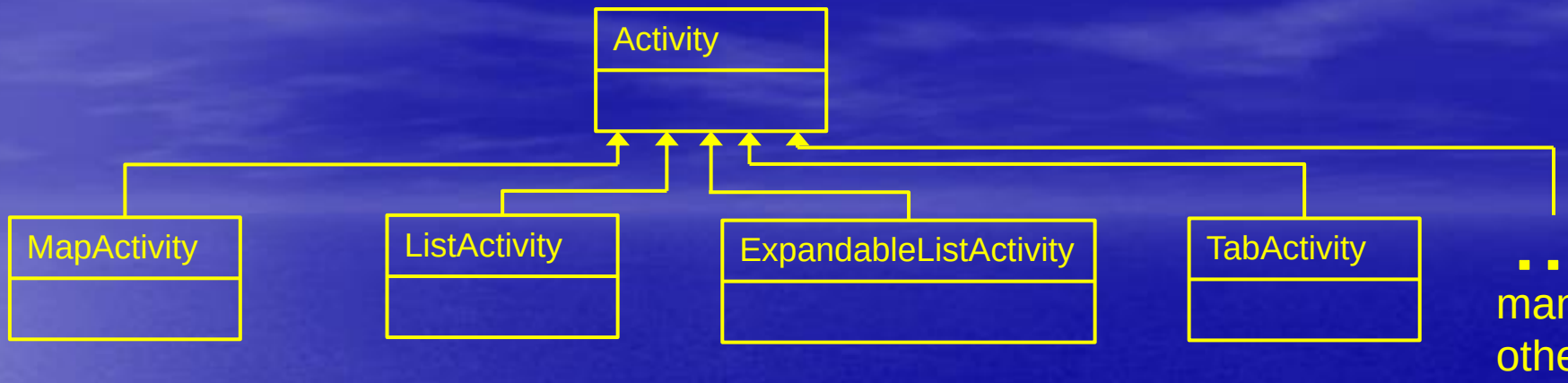
States are completely managed by the Android system

States and lifecycle

- Activity states
  - Active (running) – In interaction
  - Paused – Visible
  - Stopped – Non visible
  - Inactive (destroyed) – Without an instance object
- Android calls lifecycle methods when there is a state transition



Specialized activities



Activity starting

When an activity starts

- The Android system calls life-cycle callbacks starting with the onCreate() method
- We must override, in our derived Activity class, the life-cycle methods that we want to customize
- We must always call the parent method
- We must create the user interface, usually from a resource of type *layout* (XML), in the onCreate() method override, using setContentView()
- We can also represent an interface View (an element in the layout) in code, with an object, from its id, using findViewById()
- We should associate the listeners to interaction events in onCreate()

```
@Override
  public void onCreate(Bundle savedInstanceState)  {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);
     Button bt = findViewById(R.id.some_button);
     bt.setOnClickListener(this);
  }
```

Activity Navigation

Activity navigation uses two criteria:

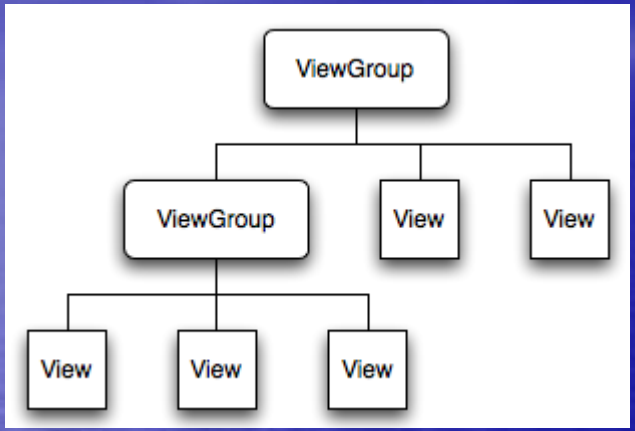
- based on the past activated activities and the activity stack:
  - . go back with the back button
- based on a defined hierarchy:
  - . declare an activity parent in the manifest file for each activity (except the one activated by the Launcher)
  - . use the Up button in the ActionBar

Resources

- Usually XML specifications of the application elements (user interface, strings, animations, etc)
  - Also icons, images, etc, in binary format
  - Organized in a hierarchy of folders in the Eclipse project with root in res/
    - drawable/ - bitmaps, graphics, shapes, etc
    - anim/ - XML specifying animations between 2 configurations (tweens)
    - color/ - colors (#AARRGGBB) for many elements of the interface according to state (pressed, focused, selected, …)
    - layout/ - screen organization
    - menu/ - options and context menus specifications
    - values/ - value collections with a name: strings, arrays, colors, styles, dimensions, …
    - xml/ - other XML files read with getXml()

Layouts

- Layouts are Views + ViewGroups organizations (in a hierarchy), defining an interface screen



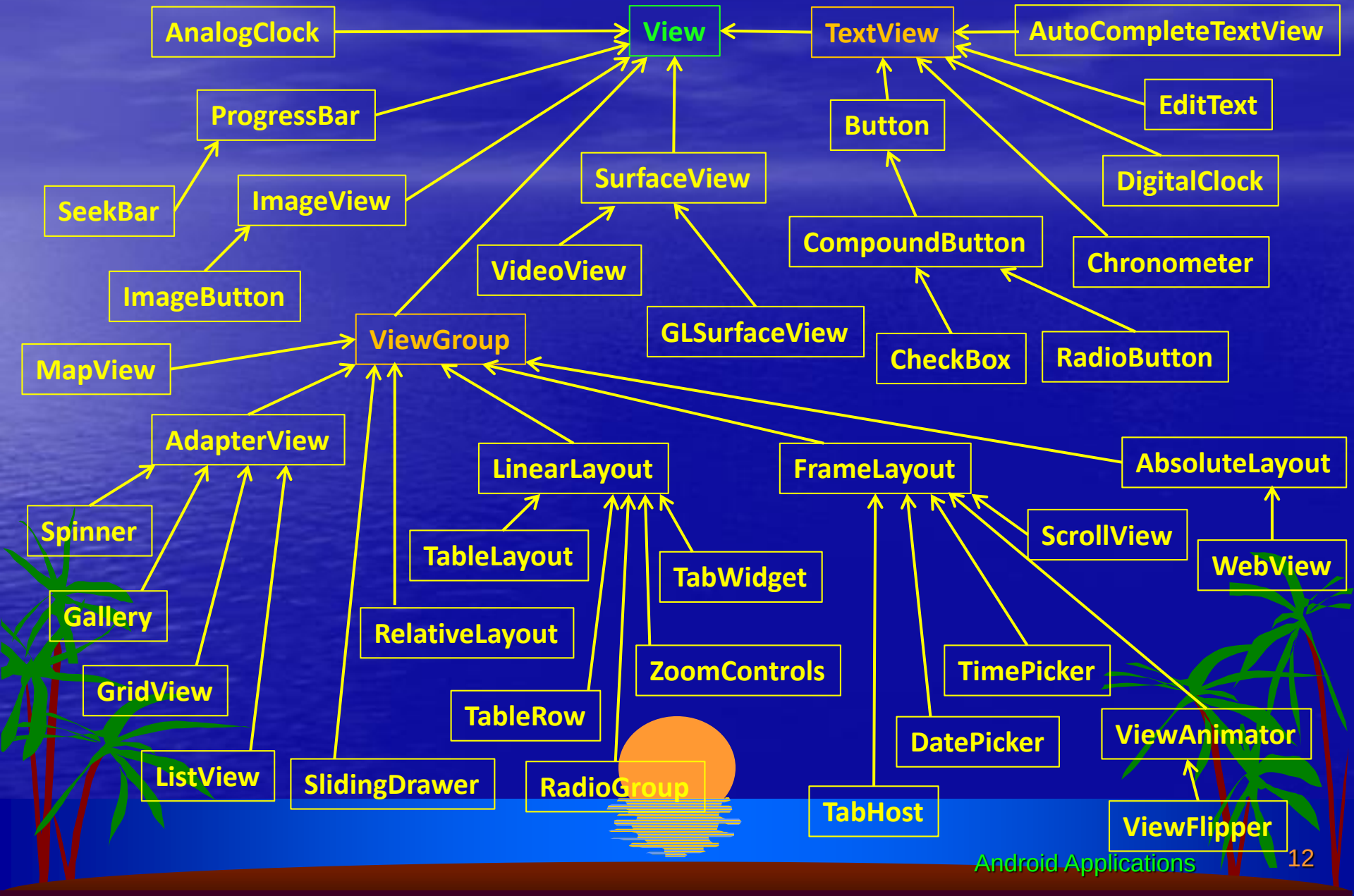

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
          android:layout_width="match_parent"
          android:layout_height="match_parent"
          android:orientation="vertical" >
   <TextView android:id="@+id/text"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="Hello, I am a TextView" />
   <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```

Many elements appearing in the XML files have an associated unique id (an integer).
The tools can generate these ids automatically.
The sintax @+id/name_id generates a new unique id and associates it to the name 'name_id'. These ids are defined in the generated file R.java and grouped in classes.

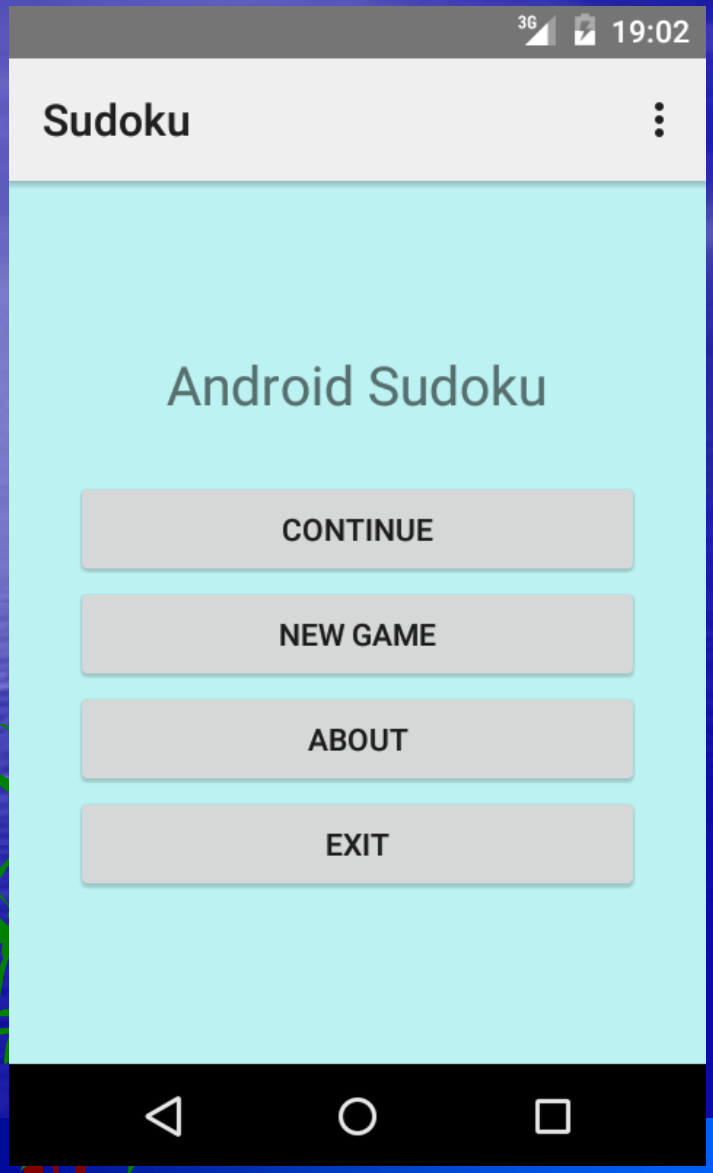
ViewGroups

- ViewGroups
  - Specify how to organize their contained Views
  - Some ViewGroups
    - LinearLayout – Organizes its descendants in a single row or column. It's the most commonly used layout.
    - RelativeLayout – Organizes each descendant relatively to the previous or to the common ancestor
    - TableLayout – Organizes its descendants in rows and columns with a syntax close to an HTML table
    - FrameLayout – All descendants are positioned relatively to left top of the ancestor. Used in Views with selectable panels
  - Descendant positioning, size, margins and other properties are specified with XML attributes. Two important values used in sizes are:
    - match_parent: has a size matching the ancestor size
    - wrap_content: has a size matching its own content

View and ViewGroup hierarchy

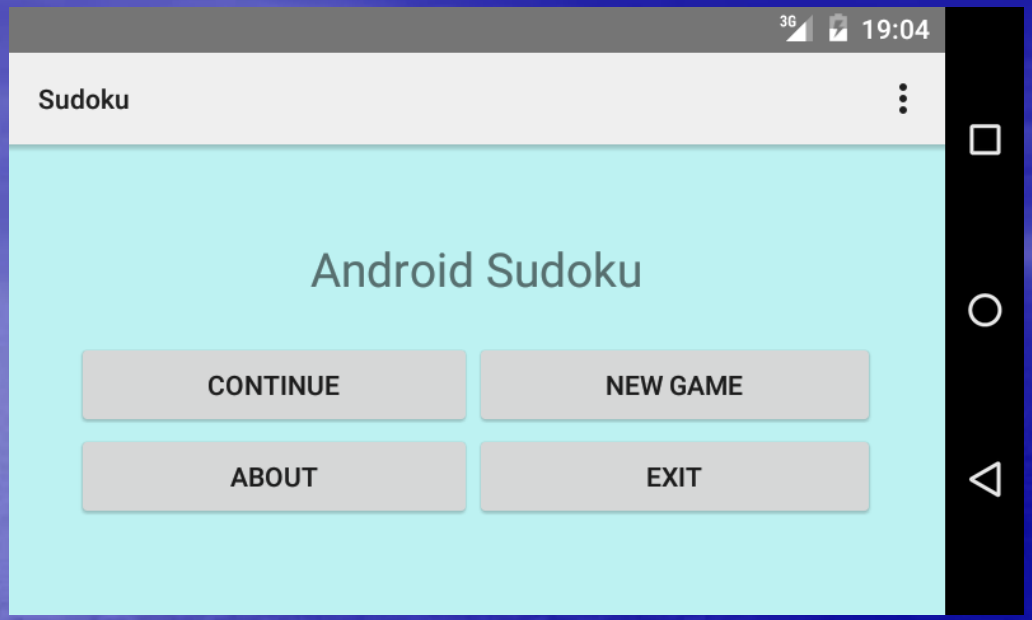


Example




```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:background="@color/background"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    android:padding="30dip"
    android:orientation="horizontal" >
    <LinearLayout
        android:orientation="vertical"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:layout_gravity="center" >
        <TextView
            android:text="@string/main_title"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:layout_gravity="center"
            android:layout_marginBottom="25dip"
            android:textSize="24.5sp" />
        <Button
            android:id="@+id/continue_button"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/continue_label" />
            .....
</LinearLayout>
```

In landscape




```
<TableLayout
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:layout_gravity="center"
    android:stretchColumns="*" >
    <TableRow>
        <Button
            android:id="@+id/continue_button"
            android:text="@string/continue_label" />
        <Button
            android:id="@+id/new_button"
            android:text="@string/new_game_label" />
    </TableRow>
    <TableRow>
        <Button
            android:id="@+id/about_button"
            android:text="@string/about_label" />
        <Button
            android:id="@+id/exit_button"
            android:text="@string/exit_label" />
    </TableRow>
</TableLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:padding="15dip“
    ...
        <LinearLayout
            ...
            android:paddingLeft="20dip"
            android:paddingRight="20dip" >
            <TextView
                ...
                android:layout_marginBottom="20dip"
                android:textSize="24.5sp" />
```

res\layout
main.xml
res\layout-land
main.xml

Styles and themes

A style is a resource that defines a set of attributes applicable to views, activities or even applications

Example: Definition of bigred style (that can be applied to, for instance, a button)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="bigred">
    <item name="android:textSize">30sp</item>
    <item name="android:textColor">#FFFF0000</item>
  </style>
</resources>
```

Application to a Button:

```
<Button
   android:id="@+id/button"
   android:text="some"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   style="@style/bigred"
/>
```

Styles can be inherited and there are a number of them already defined by Android (called themes)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="activated" parent="android:Theme.Holo">
    <item name="android:background">?android:attr/activatedBackgroundIndicator</item>
  </style>
</resources>
```

using the value of an item defined in the parent

Styles applied to an activity or an application are known as themes (but are defined the same way)

Android defines a lot of themes that we can apply to our activities using the notation @android:style/...

Examples: @android:style/Theme.Dialog @android:style/Theme.Light ...
@android:style/Widget.ProgressBar.Horizontal

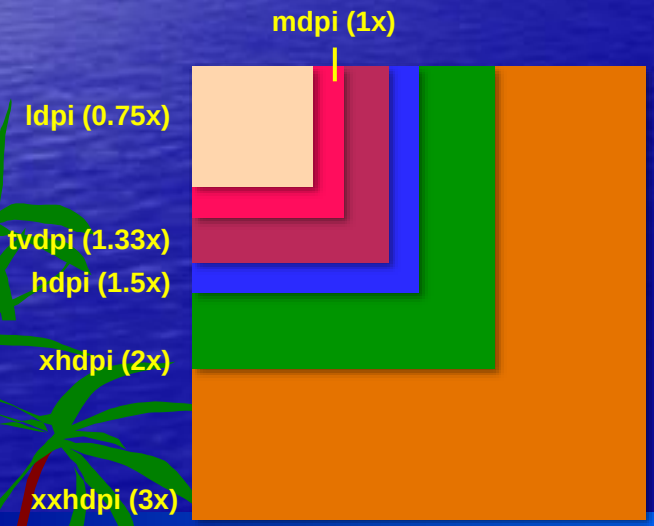
Alternative resources

- Specified in suffixed folders inside res/
  - Language and region: en, fr, pt, …, en-rUS, fr-rCA, …
  - Screen size: small, normal, large, xlarge, w<N>dp, h…, sw…
  - Screen aspect ratio: long, notlong
  - Pixel density: ldpi, mdpi, tvdpi, hdpi, xhdpi, xxhdpi
  - Screen orientation: land, port
  - Text entry method: nokeys, qwerty, 12key
  - Navigation: nonav, dpad, trackball, wheel
  - Hour: night, notnight
  - Version: v3, v4, v5, v6, v7, v8, …, v15, …, v29

Icons are very important in Android applications.
They should be adapted to the screen pixel density.
Sizes:
Launcher and Menu: 36 px (ldpi), 48 px (mdpi), 72 px (hdpi), 96 px (xhdpi)
ActionBar, Tabs, Dialogs, List Views: 24 px (ldpi), 32 px (mdpi), 48 px (hdpi), 64 px (xhdpi)

Android icons

- Icons are important square PNG images
  - Launch icons (Launcher)
  - Menus, Status, Notifications, ActionBar, …
- For better results they should be provided in multiple resolutions as drawable resources
  - mdpi is the base (48x48 launch; 32x32 other)



See the Android Design Guide and Style

Events and listeners

User interaction

Events

main thread

(looper)

message queue

get message

Android system (life-cycle states)

call calback (life-cycle)

or

call event listener (handler)

Programmer's code

Activity

callback

callback

Register Listener for some View

view.setSomeListener(listener)

SomeListener

<<interface>>

Listener

handler

IoC – Inversion of Control pattern

Events generated by the interface

- Many Views generate events
  - Events are described in interfaces (Java)
  - We need to install *listeners* to these events, by implementing the interfaces that describe them
    - In the class that contains the View (usually an Activity or Dialog)
    - As anonymous classes
    - More rarely in other classes
  - The implemented methods are called when the event occurs (the system triggers the event) Examples:
    - onClick()
    - onLongClick()
    - onFocusChange()
    - onKey()
    - onTouch()
    - onCreateContextMenu()

Defining a listener

```
public class ExampleActivity extends Activity implements OnClickListener {
    protected void onCreate(Bundle savedValues) {
        ...
        Button button = (Button)findViewById(R.id.corky);
        button.setOnClickListener(this);
    }

    // Implement the OnClickListener callback
    public void onClick(View v) {
            // do something when the button is clicked
    }
    ...
}
```

Listener defined in an Activity

```
...
 button.setOnClickListener((View v)->btClick(v));
}

void btClick(View v) {
    // do something when the button is clicked
}
...
```

```
protected void onCreate(Bundle savedValues) {
    ...
    // Capture our button from layout
    Button button = (Button)findViewById(R.id.corky);
    // Register the onClick listener with the implementation
    button.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            // do something when the button is clicked
        }
    });
    ...
}
```

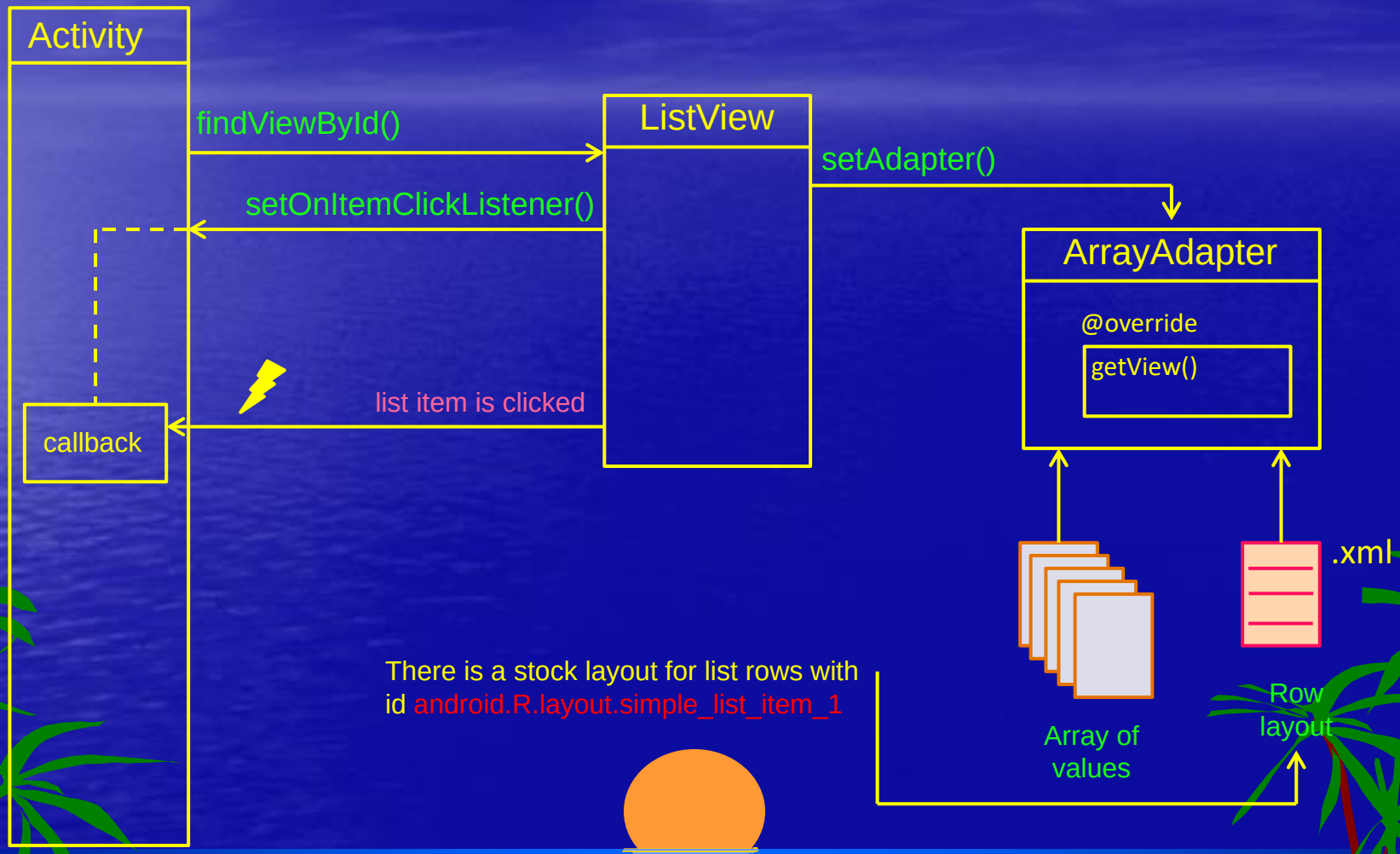
Anonymous *listener*

using a lambda (requires Java 1.8)

Note: Some Views have a property onClick. You can specify a method name in this property and define the method in the corresponding Activity. This method will be the onClick listener for that View. (This is no longer considered a good practice, because it can create confusion and deeper dependencies between layouts and code)

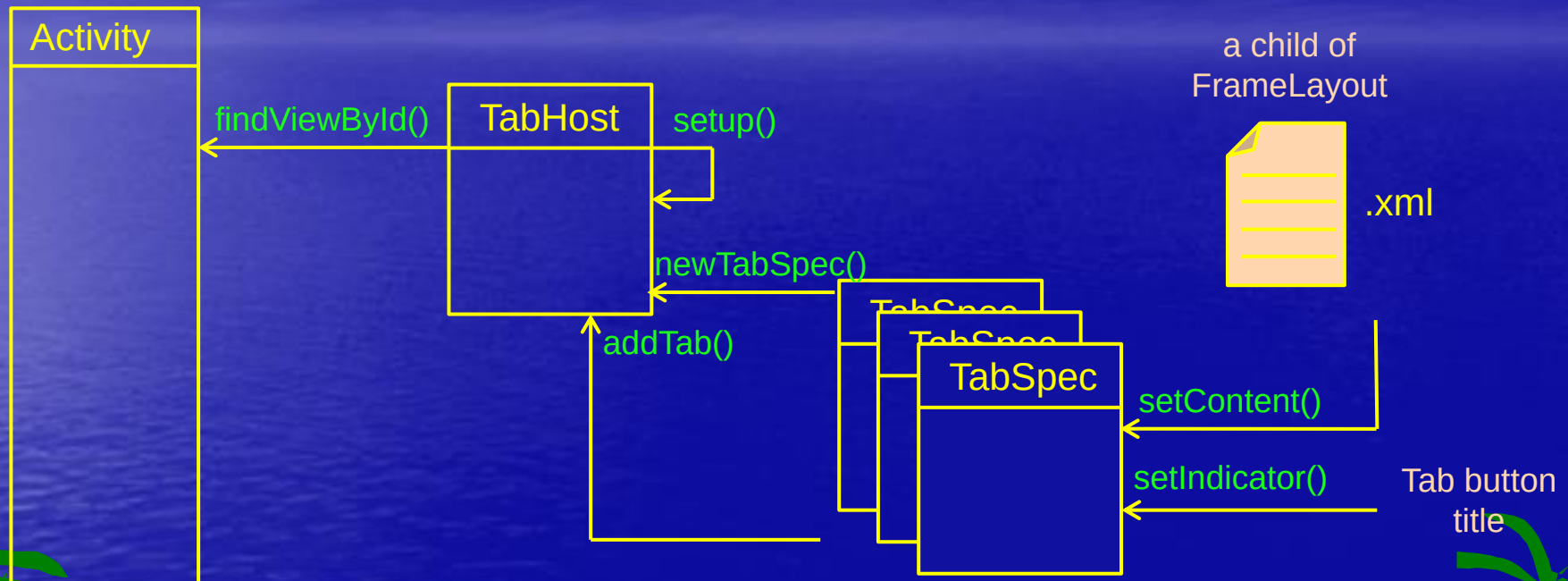
Selection containers

- **ListView, Spinner, GridView, Gallery**

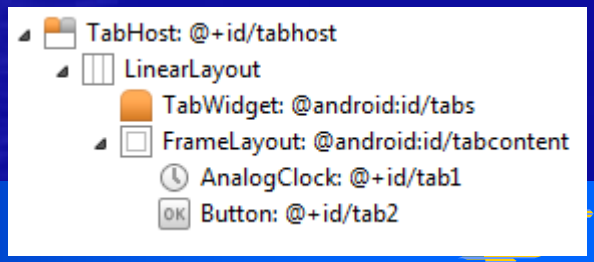


Tabs in Android

- A combination of
  - TabHost, TabWidget (tab buttons) and FrameLayout



Typical layout



These stock id's must be used (assumed by the TabHost control)

Menus

- There are two types of menus in Android
  - Option menus – associated to Activities
    - Launched by the menu key (or on the Action Bar after v. 3)
    - It calls onCreateOptionsMenu() (only the 1st time) that should build the menu from resources
    - After the selection of an item onOptionsItemSelected() is called
  - Context menus – associated to Views
    - Launched by a 'long tap' over a View
    - The View should be registered as having a context menu through registerForContextMenu()
    - If the View is registered onCreateContextMenu() and onContextItemSelected() are called in an identical way to option menus

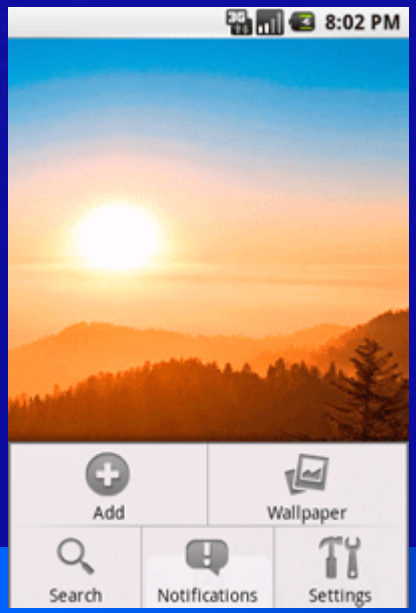
Menu definition

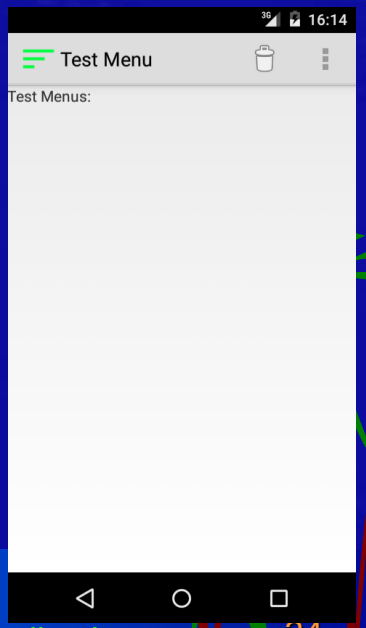
- Menus are defined as a resource in folder **menu/**
  - Can contain groups and submenus (only 1 level)

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:id="@[+][package:]id/resource_name"
        android:title="string"
        android:icon="@[package:]drawable/drawable_resource_name"
        android:alphabeticShortcut="string"
        android:numericShortcut="string"
        android:checkable=["true" | "false"]
        android:visible=["visible" | "invisible" | "gone"]
        android:enabled=["enabled" | "disabled"]
        android:showAsAction=[“never” | “always”] | [“ifRoom”] />
   <group android:id="@[+][package:]id/resource name"
         android:checkableBehavior=["none" | "all" | "single"]
         android:visible=["visible" | "invisible" | "gone"]
         android:enabled=["enabled" | "disabled"] >
      <item />
   </group>
   <item >
      <menu>
         <item />
      </menu>
   </item>
</menu>
```

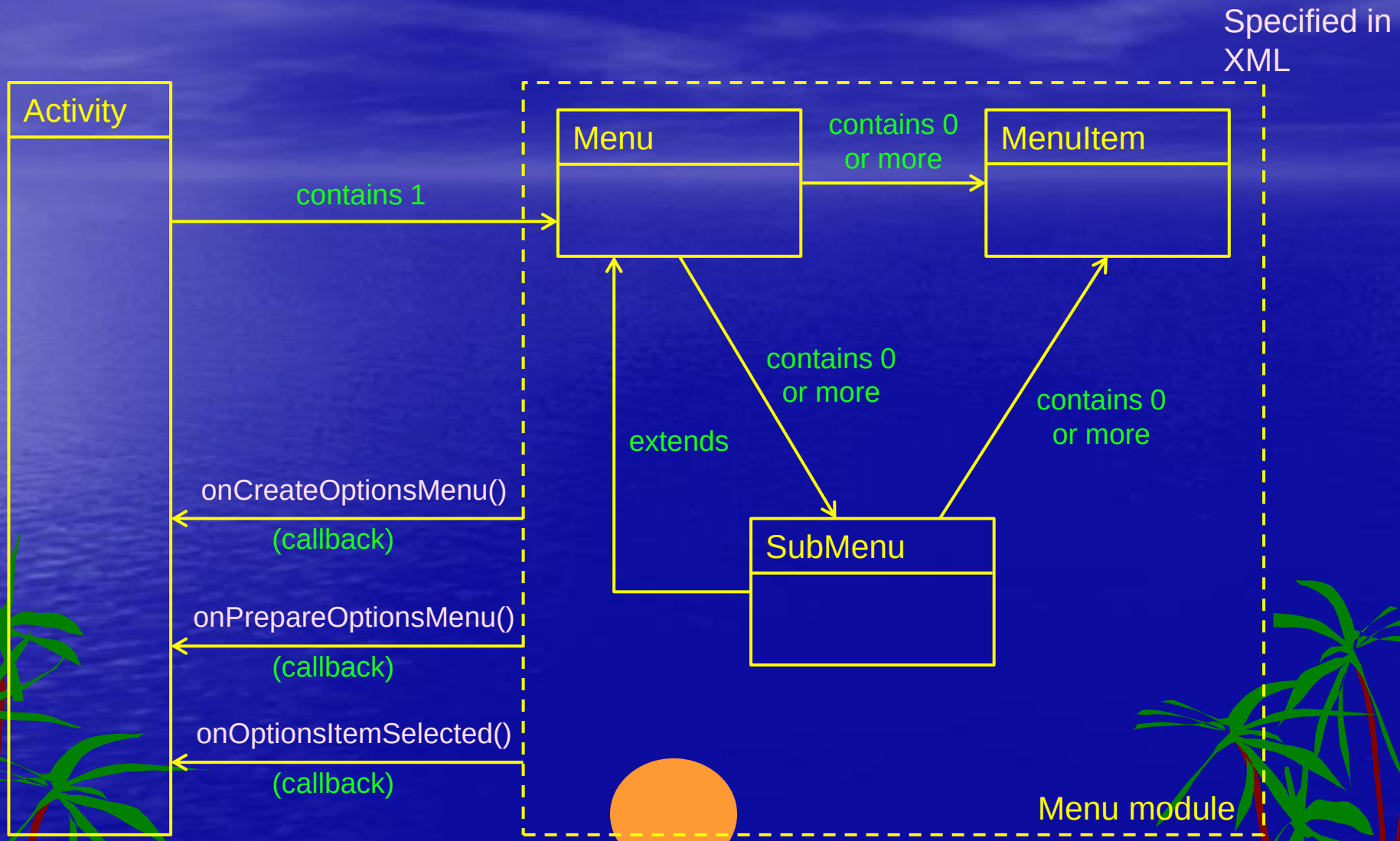
Menu creation in onCreate…Menu():

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

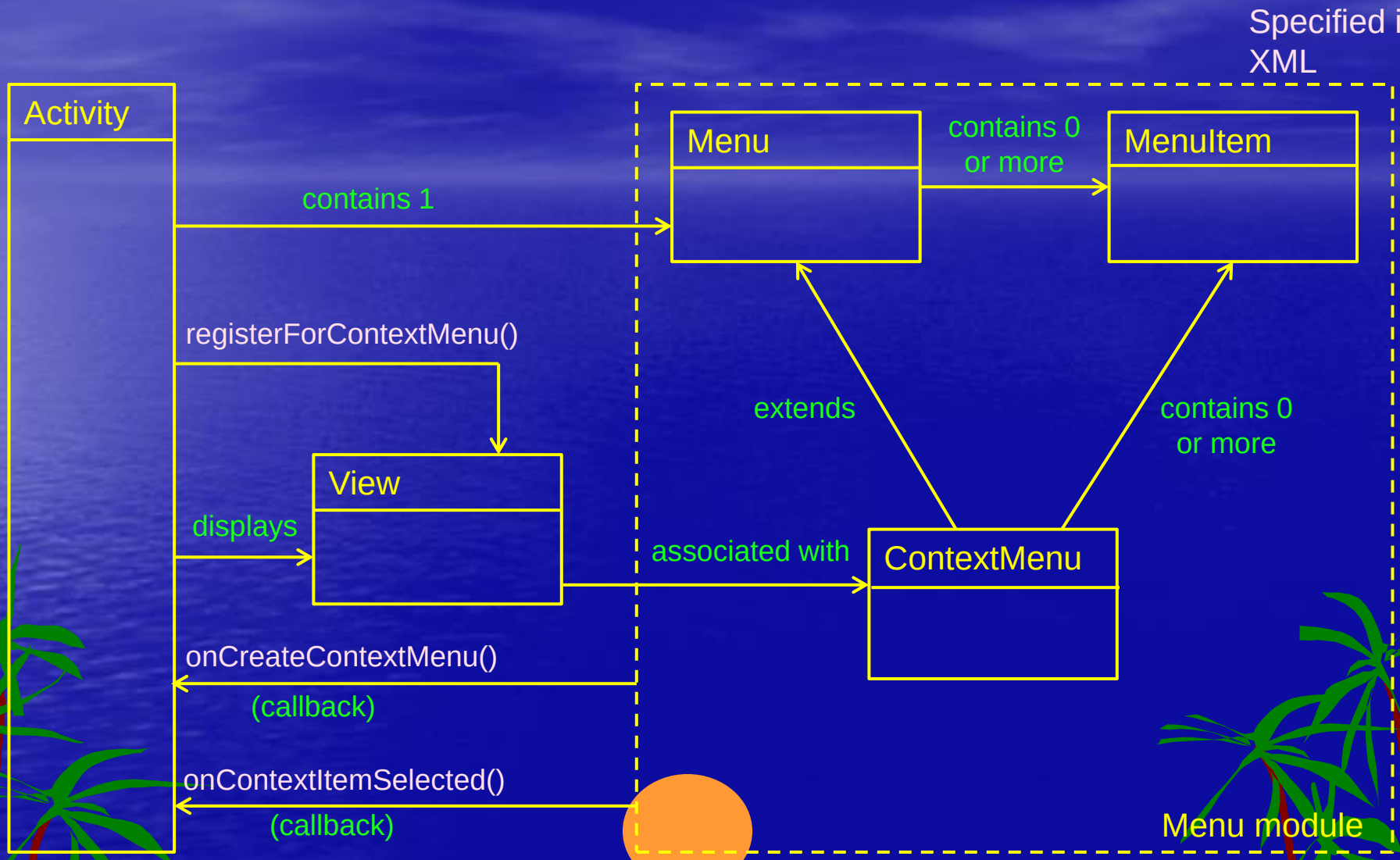




Options menu system



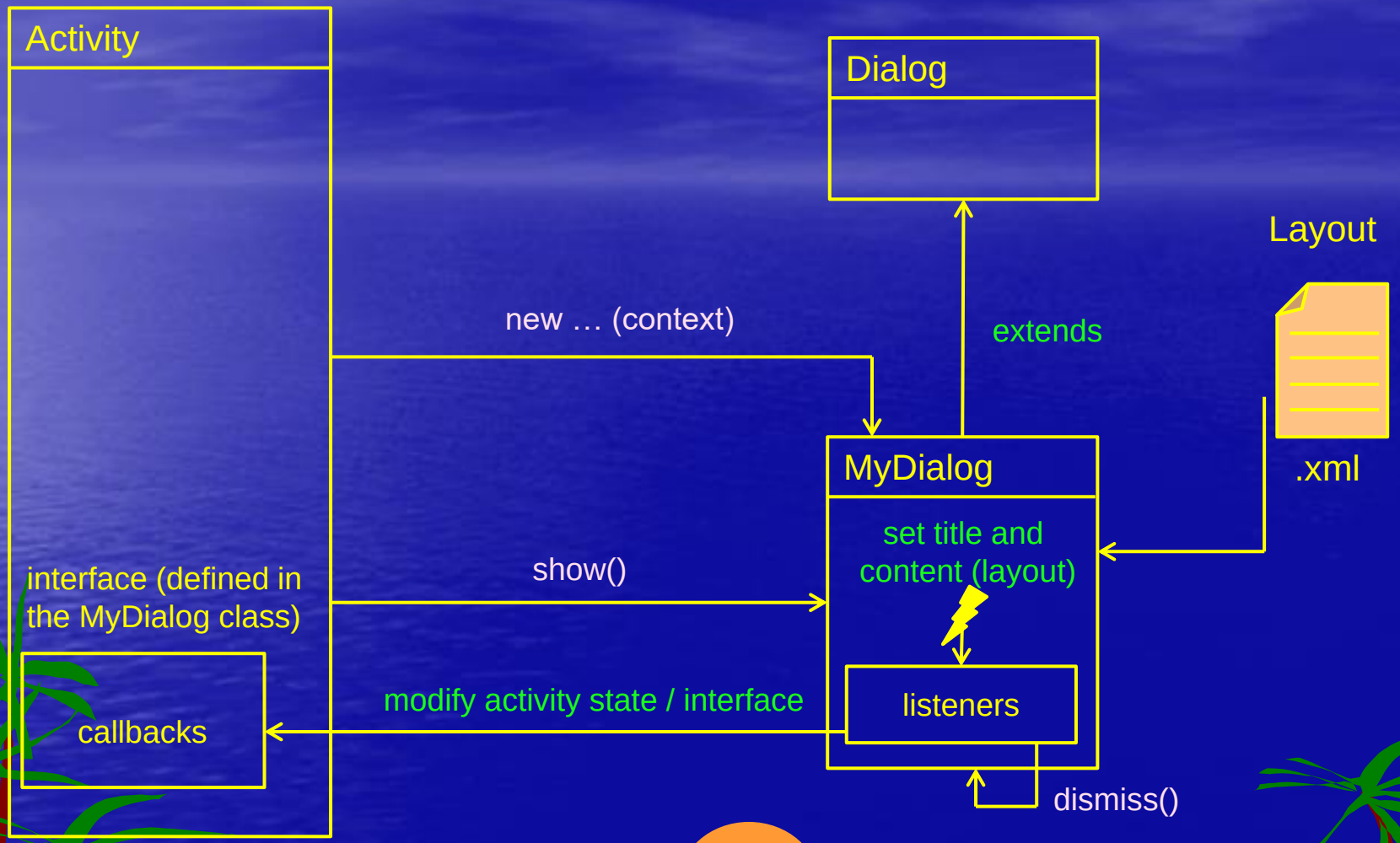
Context menu system



Dialog boxes

- Generic
  - Implemented in classes inheriting from android.app.Dialog
  - In the constructor we can define a title (setTitle()) and save the parent activity
  - An Activity displays a dialog box through the method show() from the Dialog class
  - Override onCreate() in the Dialog class to define the content through a layout (setContentView())
  - Terminates calling dismiss() from the Dialog class
- Pre-defined
  - AlertDialog
  - ProgressDialog
  - DatePickerDialog
  - TimePickerDialog

A simple dialog usage scenario



Example

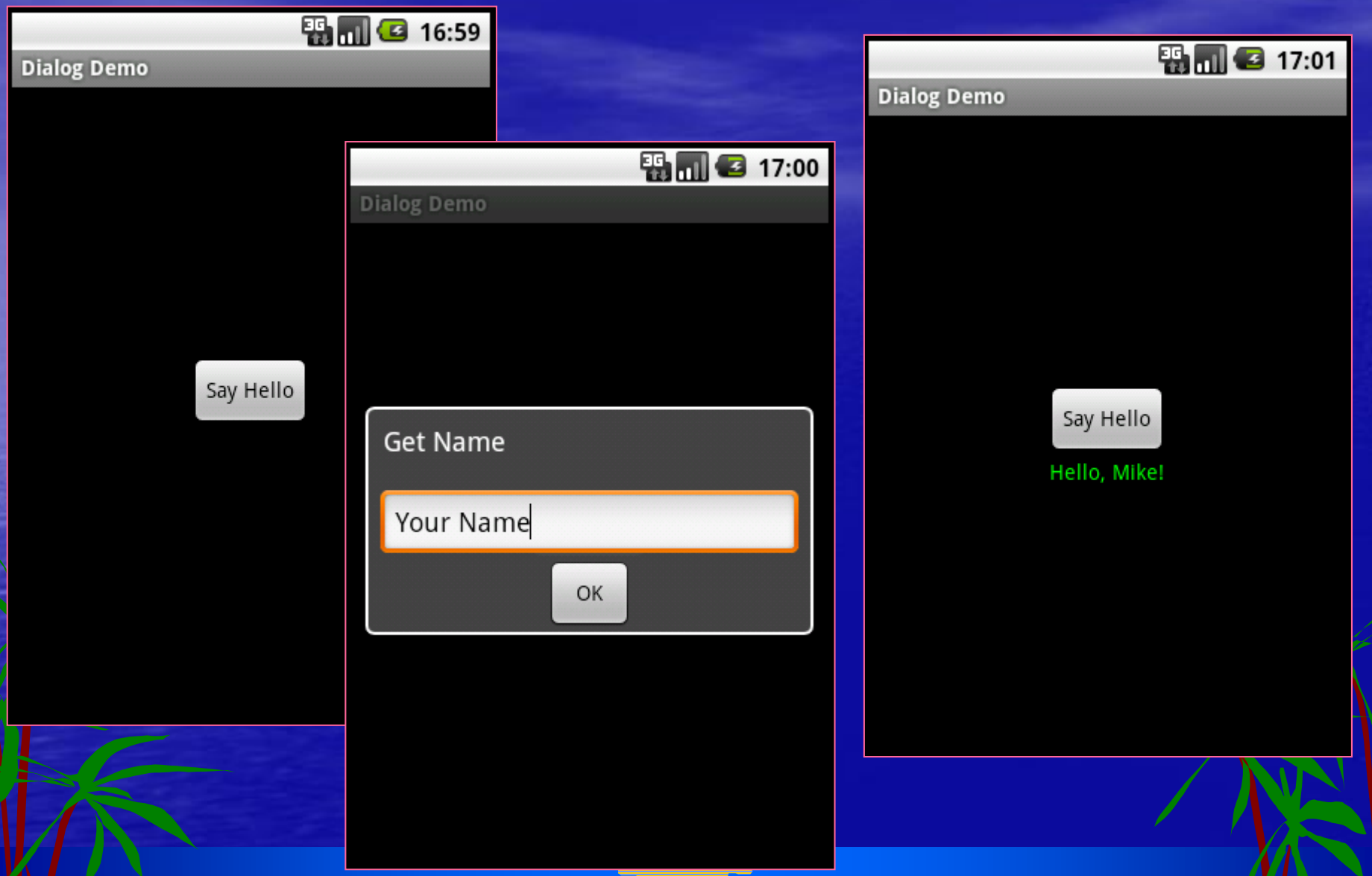
Activity

```
public class DialogDemo extends Activity implements View.OnClickListener {
  private TextView tv;
  private GetNameDialog dialog;
  ...

   public void onClick(View v) {
     switch (v.getId()) {
       case R.id.button:
          dialog = new GetNameDialog(this, getString(R.string.initial_name));
          dialog.show();
          break;
       case R.id.getnameokbutton:
          tv.setText("Hello, " + dialog.edt.getText().toString() + "!");
          dialog.dismiss();
          break;
     }
   }
}
```

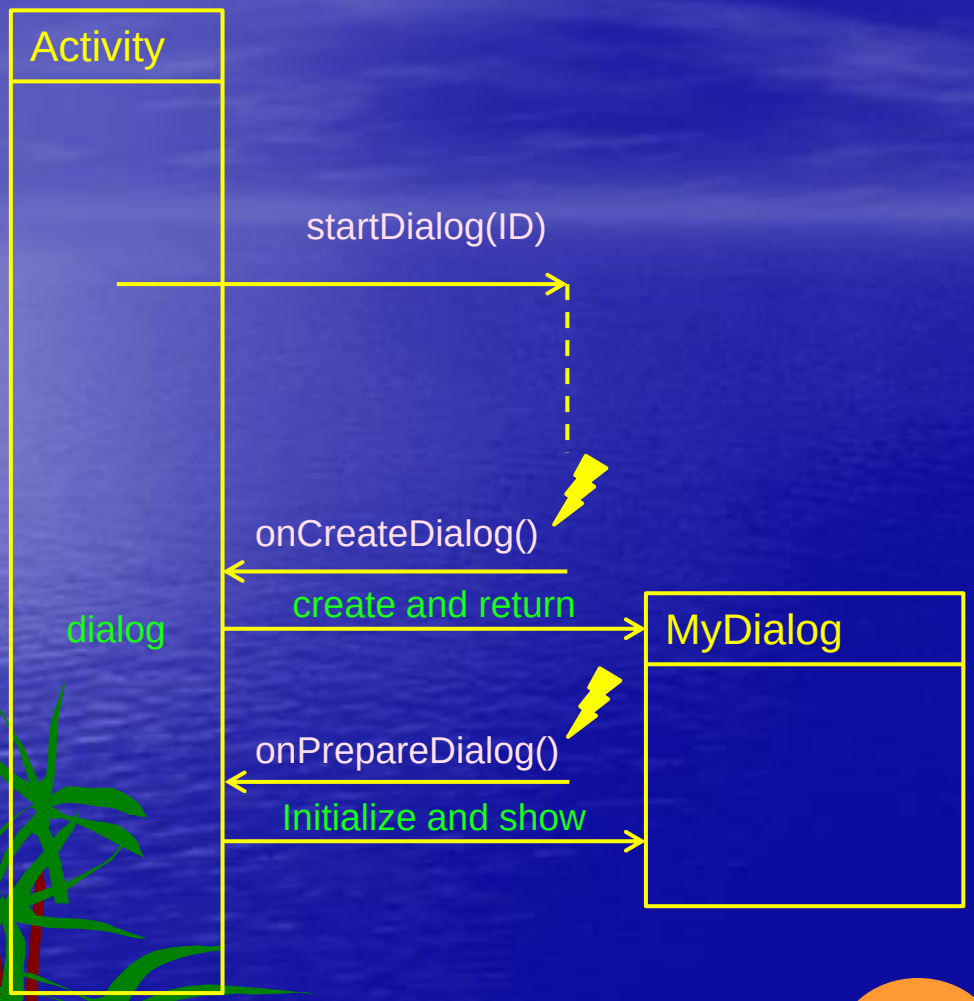
Dialog

```
public class GetNameDialog extends Dialog {
  EditText edt;

  public GetNameDialog(Context context, String init) {
     super(context);
     setTitle(R.string.dialog_title);
     setContentView(R.layout.dialog);
     getWindow().setLayout(LayoutParams.MATCH_PARENT, LayoutParams.WRAP_CONTENTS);
     edt = (EditText) findViewById(R.id.yourname);
     edt.setText(init);
     Button okbut = (Button) findViewById(R.id.getnameokbutton);
     okbut.setOnClickListener((View.OnClickListener) context);
  }
}
```

Example in action

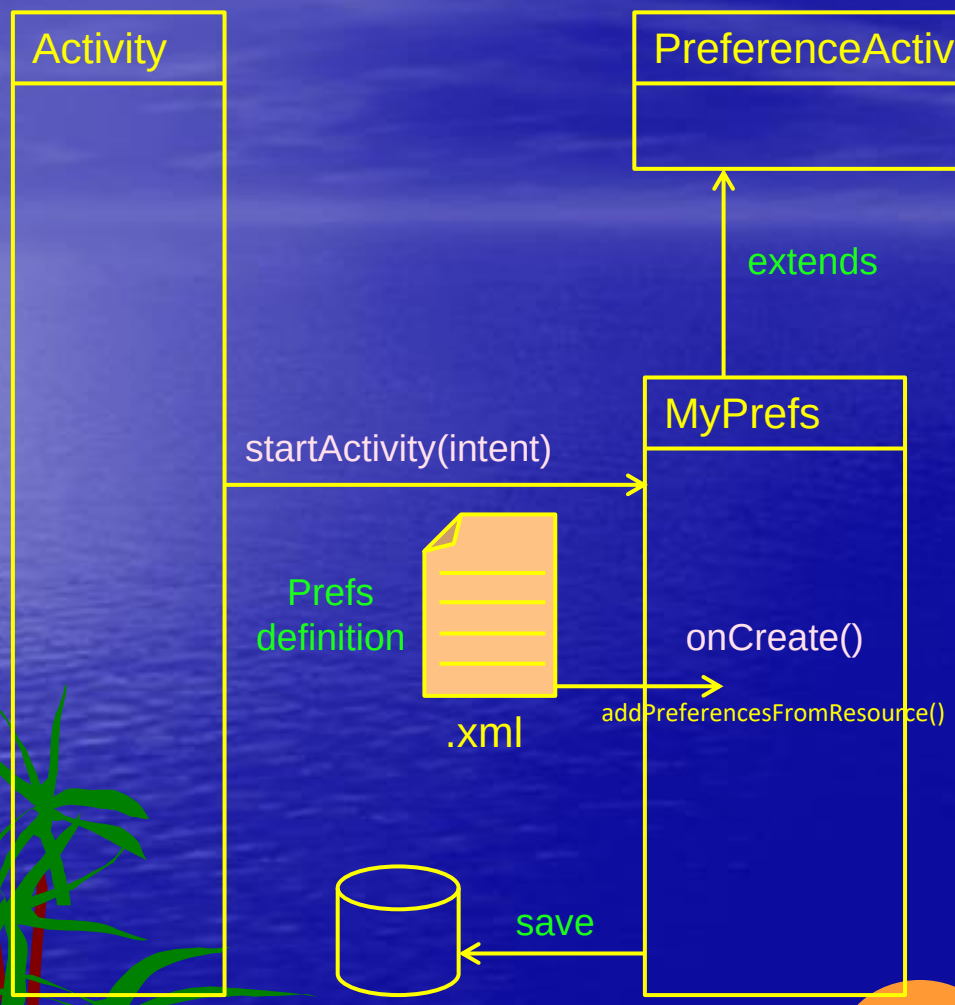


Dialog reuse pattern




```
...

@Override
public Dialog onCreateDialog(int id) {
   switch(id) {
     case (GETNAME_DIALOG):
          dialog = new GetNameDialog(this);
          return dialog;
   }
   return null;
}

@Override
public void onPrepareDialog(int id, Dialog dialog) {
   switch(id) {
     case (GETNAME_DIALOG):
        GetNameDialog gn = (GetNameDialog) dialog;
        gn.edt.setText(getString(R.string.initial_name));
        break;
   }
}
```

Application preferences

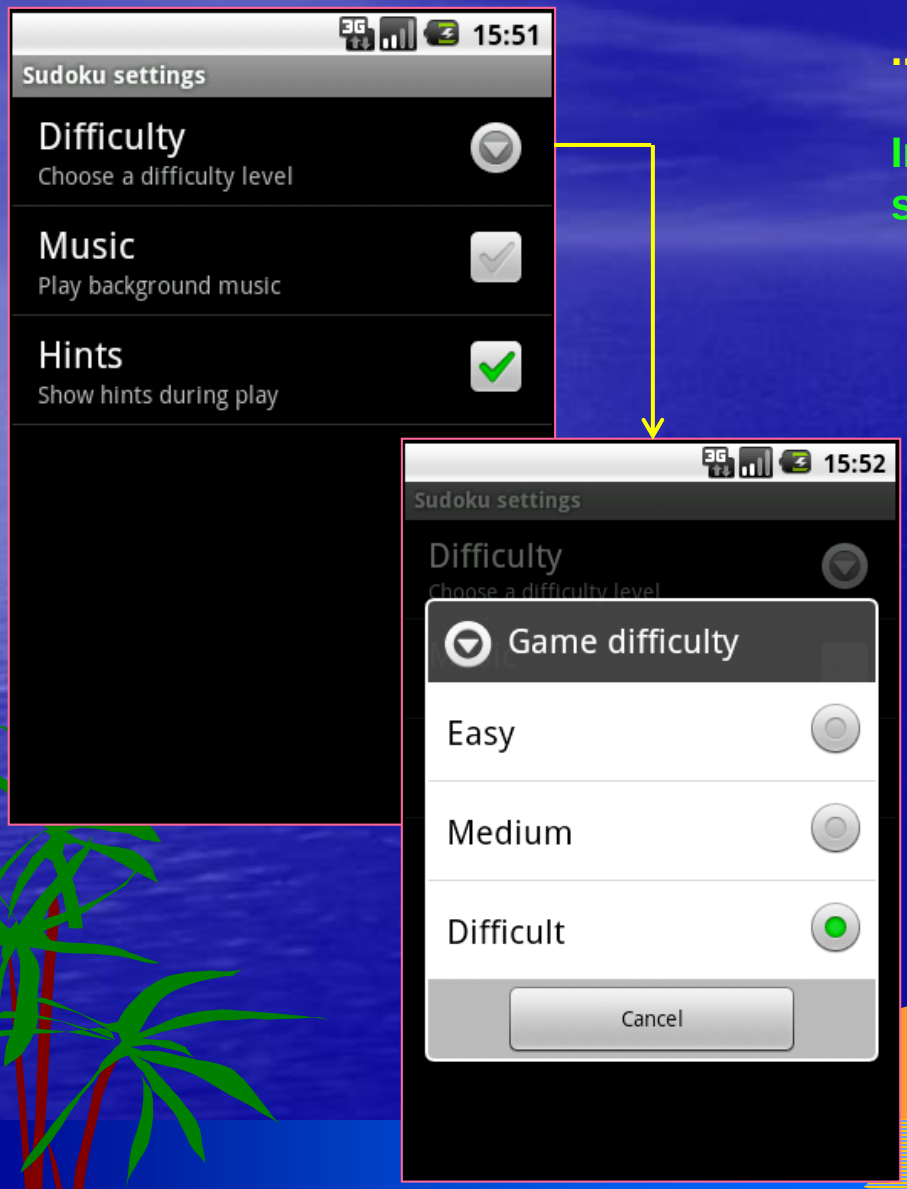


Several types of preferences:

- ListPreferences - mutually exclusive
- CheckBoxPreferences - boolean
- EditTextPreferences – string
- IntentPreferences – launch an activity
- RingTonePreference - choose a ringtone

```
<PreferenceScreen  xmlns:android="....."> 
  <ListPreference 
      android:key="difficulty" 
      android:title="@string/diff_title" 
      android:summary="@string/diff_summary" 
      android:entries="@array/diff_options" 
      android:entryValues="@array/diff_vals" 
      android:dialogTitle="@string/dialog_title" 
      android:defaultValue="1" />
  <CheckBoxPreference 
      android:key="music" 
      android:title="@string/music_title" 
      android:summary="@string/music_summary" 
      android:defaultValue="true" />
  <CheckBoxPreference 
      android:key="hints" 
      android:title="@string/hints_title" 
      android:summary="@string/hints_summary" 
      android:defaultValue="true" />
</PreferenceScreen>
```

Preferences example



... Starting the preferences activity ...

```
Intent intent = new Intent(this, MyPrefs.class);
startActivity(intent);
```

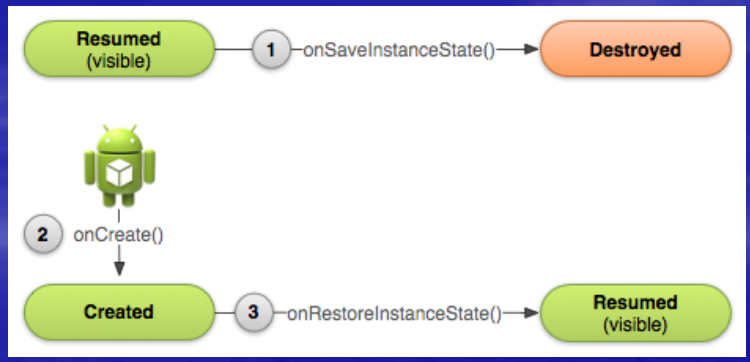
... Getting the preferences values ...

MyPrefs class:

```
...

public static boolean getMusic(Context context) {
   String key = "music";
   boolean default = true;
   SharedPreferences sprefs =
      PreferenceManager.
         getDefaultSharedPreferences(context);
   return sprefs.getBoolean(key, default);
}
...
```

State preservation

Standard activities in an application save some of their internal state whenever the activity is in its way to the destroyed state. If the activity is active again it will restore its state before the next time it is displayed.
This facility can be extended for non-standard activities (i.e. graphical) or other type of state.



Activities call onSaveInstanceState(Bundle) and onRestoreInstanceState(Bundle) when they need to save and restore state (it is saved on memory). These methods can be overridden in the derived class.

onSaveInstanceState(Bundle)
overriden

add custom state

save the Bundle and add view state (call base method)

reverse for restoring

```
@Override
protected void onSaveInstanceState(Bundle state) {
   state.putFloat("Score", mCurrentScore);
   state.putInt("Level", mCurrentLevel);
   super.onSaveInstanceState(state);
}

@Override
protected void onRestoreInstanceState(Bundle state) {
   super.onRestoreInstanceState(state);
   mCurrentScore = state.getFloat("Score");
   mCurrentLevel = state.getInt("Level");
}
```

Device rotation

When certain situations occur that cause a change in device configuration (rotating the device, extending or hiding a logical keyboard, docking or undocking, changing the locale or language) Android destroys and re-creates the running and paused activities, the next time they are viewed.

This could be necessary to load new resources for the user interface, more adapted to the new configuration.

Device rotation is a very common situation and every user expects that all applications support this change, eventually adapting the interface to portrait and landscape.

But, usually, is not enough to provide different layouts adapted to the device orientation. When the activity is destroyed, all internal variables loses their values, and the activity default mechanism of saving state based on the onSaveInstanceState() and onRestoreInstanceState() methods only saves and restores the some of the internal contents of Views (but not all).

Overriding these methods we can save/restore more values on the same Bundle. This Bundle is saved in memory as long as the activity remains in the activity stack (if the process is removed from memory, or the user pushes the back button, or if the activity calls finish() the Bundle is lost).

Bundles can store simple types, their arrays and serializable objects (they are hash tables).

Another similar way (obsolete) that saves/restores any kind of object is by overriding the activity method onRetainNonConfigurationInstance() and calling getLastNonConfigurationInstance() inside onCreate().

Features and permissions

Features

- Declared in the manifest when an application needs a certain hardware characteristic that can be unavailable
  - The features are defined in the Android class **PackageManager**

```
<uses-feature android:name="android.hardware.bluetooth_le"  android:required="true" />
<uses-feature android:name="android.hardware.camera" />
```

Permissions

- To use certain API functionalities the application needs permission from the user
  - Declared in the manifest and asked for during installation
  - Defined in the class Manifest.permission

```
<uses-permission android:name="android.permission.INTERNET“ />
```

  - After marshmallow (6.0) ‘dangerous’ permissions should be obtained in runtime