Android UI Programming

Activity

- Android Application Component for UI
  - Provides a window for user interaction
  - Content area for views and widgets
  - Requires lifecycle and state management

Example: HelloWorldActivity (Java)

```java
package com.example.smd.helloworld;

import android.app.Activity;
import android.os.Bundle;

public class HelloWorldActivity extends Activity
{
    /** Called when the activity is first created.*/
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

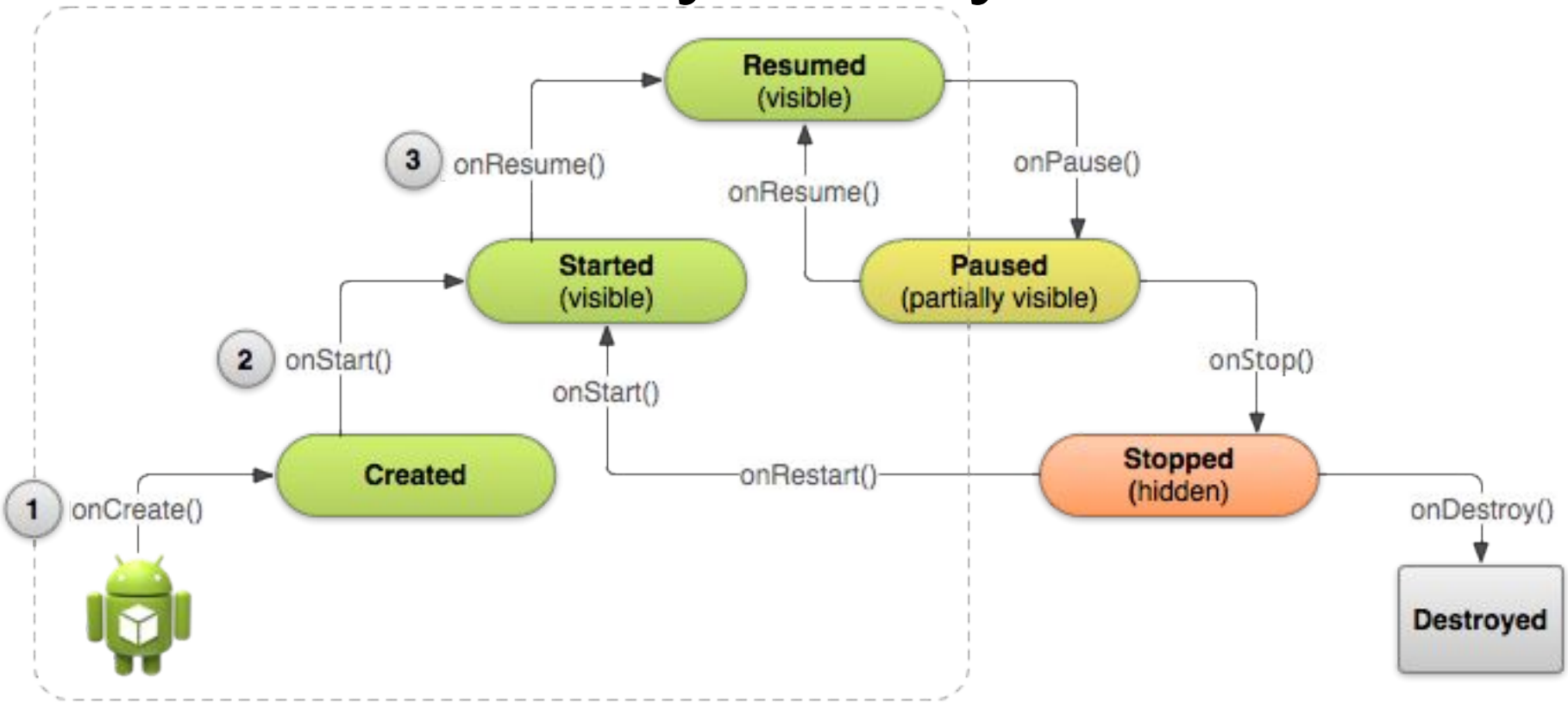
Example: HelloWorldActivity (Kotlin)

```
package com.example.smd.helloworld

import android.app.Activity
import android.os.Bundle

class HelloWorldActivity : Activity() {
    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        setContentView(R.layout.main)

    }
}
```

Example Manifest declaration

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.smd.helloworld"
   android:versionCode="1"
   android:versionName="1.0" >

   <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"

      <activity
         android:name="HelloWorldActivity"
         android:label="@string/app_name" >
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>

   </application>

</manifest>
```

Activity Lifecycle



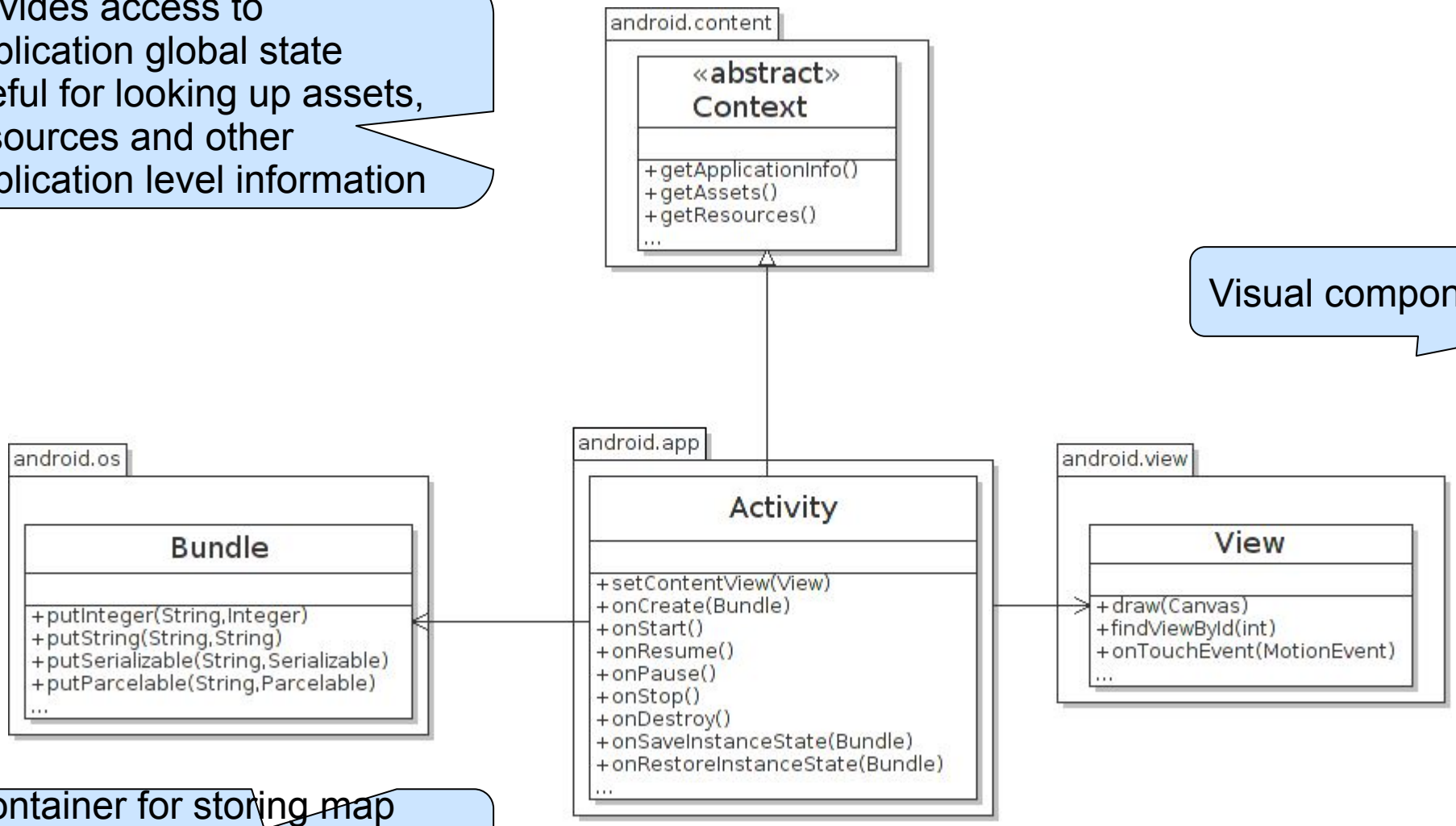
- Primary states
  - Resumed : Activity in foreground – user can interact
  - Paused : At least partially obscured by another activity – activity may be partially visible but loses focus.
  - Stopped : Completely hidden – not visible to user. Activity may exist in memory, but in low memory situation, can be killed by OS.

Lifecycle and state management

- Need
  - Stop animations / video and other actions consuming CPU
  - Release system resources such as handles to sensors
  - Commit unsaved changes (if required)
- Method
  - onPause() / onResume()
    - release / acquire resources, stop / start CPU utilization
  - onStop() / onStart() / onRestart()
    - save / handle persistent data
  - onSaveInstanceState() / onRestoreInstanceState()
    - save / load transient data

Basic Activity Relationships

Provides access to application global state
Useful for looking up assets, resources and other application level information

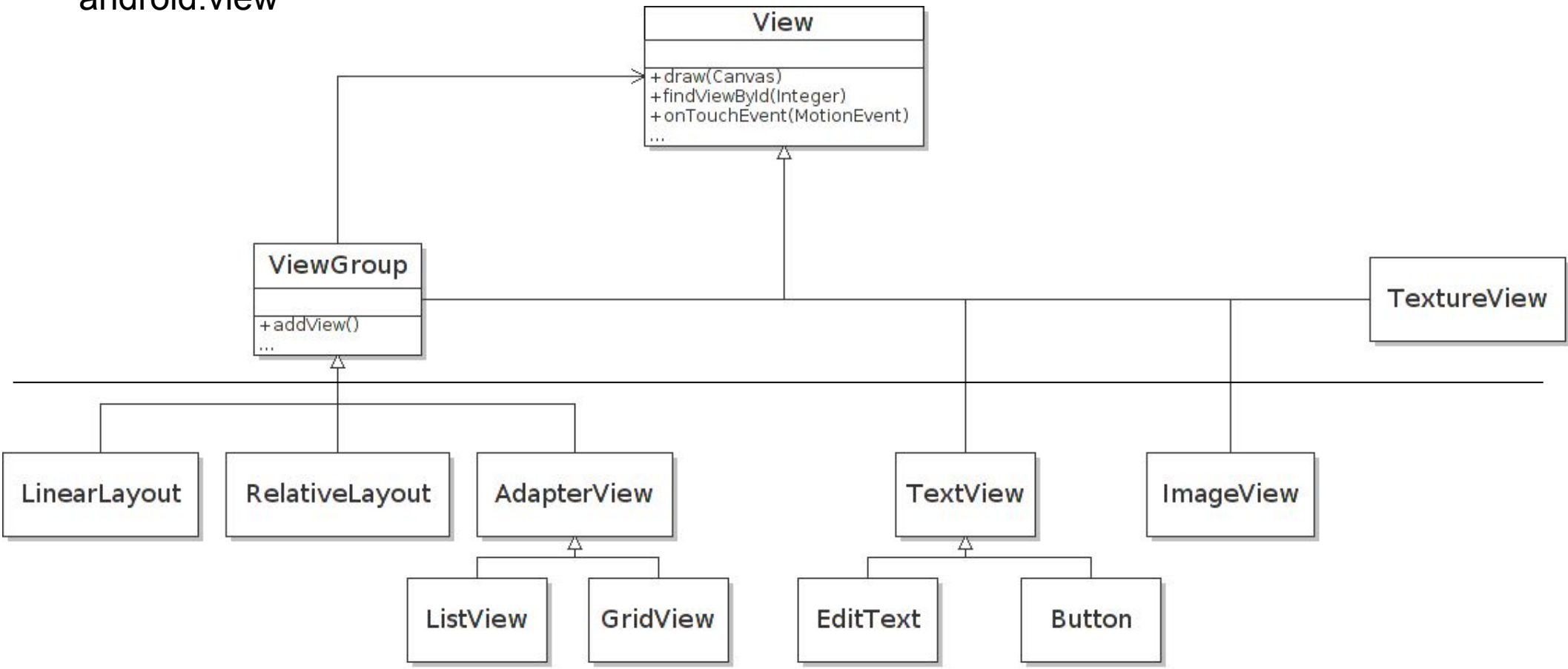


Visual components

A container for storing map data
Useful for state management

Views and Widgets

android.view

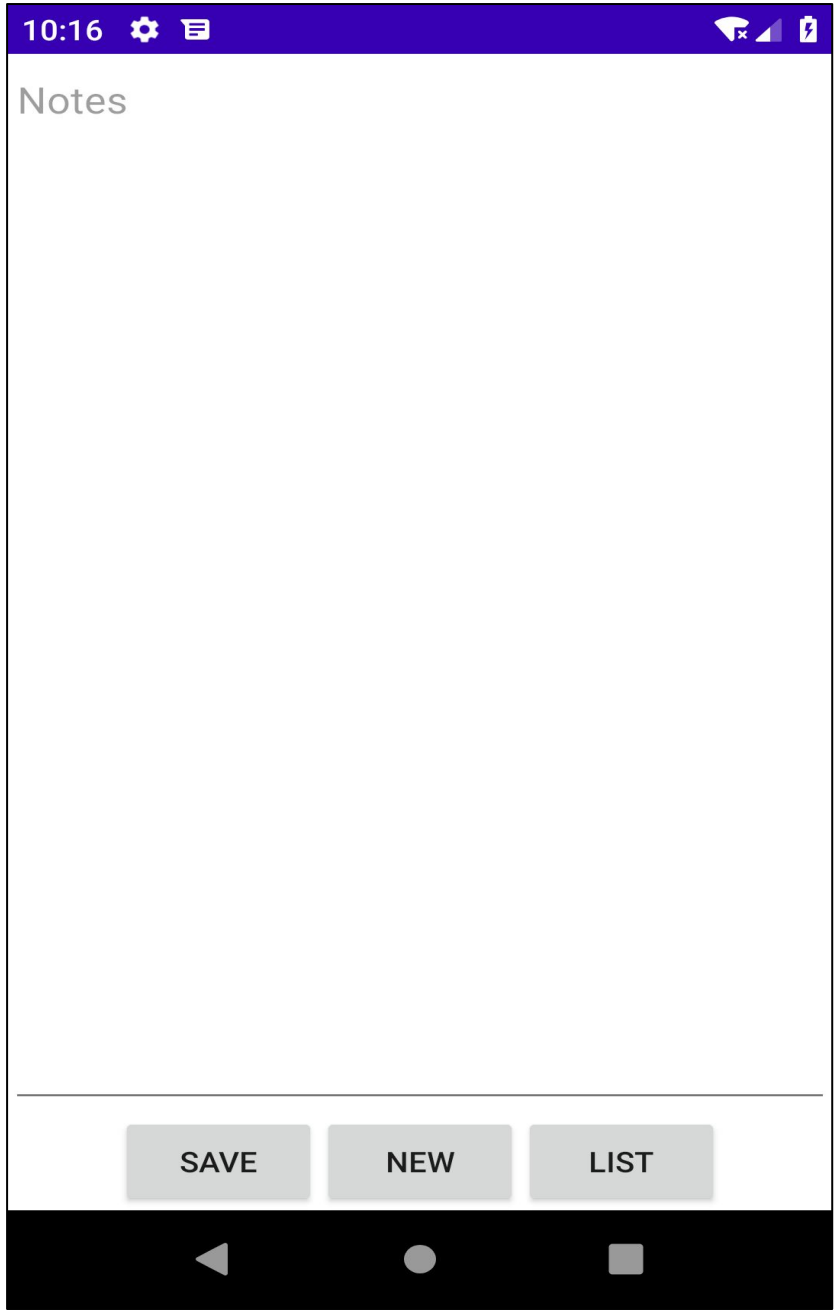


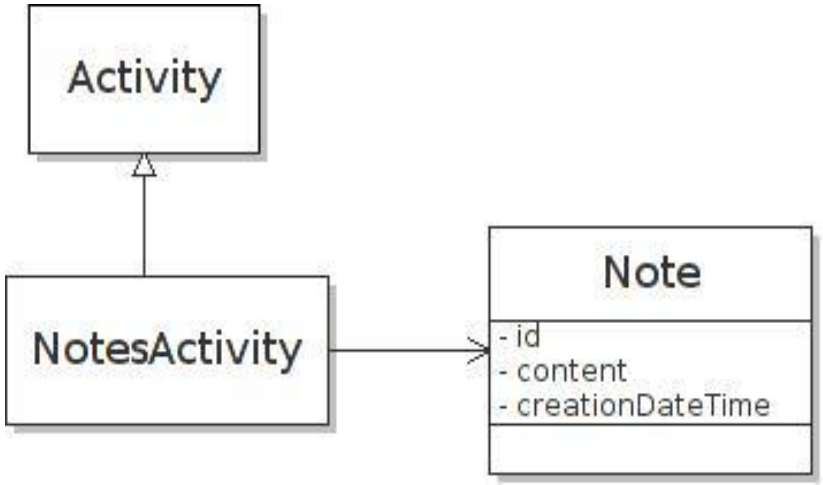
android.widget

UI Development Approaches

- Programmatic
  - Using Java/Kotlin code
  - Dynamic view generation
  - Slightly complicated to maintain
- Declarative
  - Using XML
  - Static views – may be further adjusted through programmatic approach
  - Easier to maintain

Example: Notes App






```
package com.example.smd;

import java.util.Date;
import java.util.UUID;


public class Note{

  private String id;
  private String content;
  private Date creationDateTime;

  public Note(){
    init();
  }

  public Note(String content){
   init();
   this.content = content;
  }

  private void init(){
    this.id = UUID.randomUUID().toString();
    this.creationDateTime = new Date();
  }

  public void setContent(String content){
    this.content = content;
  }

}
```

Programmatic Approach

```java
package com.example.smd;

import java.util.ArrayList;
import android.app.Activity;
import android.os.Bundle;
...

public class NotesActivity extends Activity
{
    ArrayList<Note> notes;
    Note currentNote;
    EditText textArea;

   public void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      //setContentView(R.layout.main);
      createUi();
      notes = new ArrayList<Note>();
   }

   private void saveNote(){
         String content = textArea.getText().toString();
         if(currentNote == null){
                  currentNote = new Note(content);
                  notes.add(currentNote);
         }
         currentNote.setContent(content);

         String text = "Note saved successfully";
         Toast toast = Toast.makeText(this,text,Toast.LENGTH_SHORT);
         toast.show();
   }

   private void newNote(){
         saveNote();
         textArea.setText("");
         currentNote = null;
   }

   private void listNotes(){
         String text = "Total " + notes.size() + " notes";
         Toast toast = Toast.makeText(this,text,Toast.LENGTH_LONG);
         toast.show();
   }

   private void createUi(){ ... }
   private void createMenu(){ ... }
}
```

Programmatic Approach

```
private void createUi(){
	LinearLayout outerLayout = new LinearLayout(this);
	outerLayout.setLayoutParams(new LayoutParams(LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT));
	outerLayout.setOrientation(LinearLayout.VERTICAL);

	textArea = new EditText(this);
	textArea.setLayoutParams(new LinearLayout.LayoutParams(LayoutParams.MATCH_PARENT, LayoutParams.WRAP_CONTENT,1f));
	textArea.setHint("Notes");
	textArea.setGravity(Gravity.TOP);

	outerLayout.addView(textArea);
	outerLayout.addView(createMenu());
	setContentView(outerLayout);
}
```

Programmatic Approach

```
private ViewGroup createMenu(){
        LinearLayout layout = new LinearLayout(this);
        layout.setLayoutParams(new LayoutParams(LayoutParams.MATCH_PARENT,LayoutParams.WRAP_CONTENT));
        layout.setOrientation(LinearLayout.HORIZONTAL);
        layout.setGravity(Gravity.CENTER);

        LayoutParams params = new LayoutParams(LayoutParams.WRAP_CONTENT,LayoutParams.WRAP_CONTENT);

        Button saveButton = new Button(this);
        saveButton.setLayoutParams(params);
        saveButton.setText("Save");
        saveButton.setOnClickListener(new OnClickListener() {

      @Override
      public void onClick(View arg0) {
        saveNote();
      }
    });

        Button newButton = new Button(this);
        newButton.setLayoutParams(params);
        newButton.setText("New");
        newButton.setOnClickListener(new OnClickListener() {

      @Override
      public void onClick(View arg0) {
        newNote();
      }
    });

        Button listButton = new Button(this);
        listButton.setLayoutParams(params);
        listButton.setText("List");
        listButton.setOnClickListener(new OnClickListener() {

      @Override
      public void onClick(View arg0) {
        listNotes();
      }
    });

        layout.addView(saveButton);
        layout.addView(newButton);
        layout.addView(listButton);

        return layout;
  }
```

Declarative Approach

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   >
 <EditText
   android:id="@+id/text_area"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:layout_weight="1"
   android:hint="Notes"
   android:gravity="top"
   />
 <LinearLayout
   android:orientation="horizontal"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:gravity="center"
   >
  <Button
   android:id="@+id/button_save"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Save"
   android:onClick="buttonClick"
   />
  <Button
   android:id="@+id/button_new"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="New"
   android:onClick="buttonClick"
   />
  <Button
   android:id="@+id/button_list"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="List"
   android:onClick="buttonClick"
   />
 </LinearLayout>
</LinearLayout>
```

Declarative Approach

```
package com.example.smd;

import java.util.ArrayList;
import android.app.Activity;
...
public class NotesActivity extends Activity
{
   ArrayList<Note> notes;
   Note currentNote;
   EditText textArea;

   public void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      textArea = (EditText) findViewById(R.id.text_area);
      notes = new ArrayList<Note>();
   }

   private void saveNote(){
         String content = textArea.getText().toString();
         if(currentNote == null){
                  currentNote = new Note(content);
                  notes.add(currentNote);
         }
         currentNote.setContent(content);
     ...
   }

   private void newNote(){
         saveNote();
         textArea.setText("");
         currentNote = null;
   }

   private void listNotes(){
         String text = "Total " + notes.size() + " notes";
         Toast toast = Toast.makeText(this,text,Toast.LENGTH_LONG);
         toast.show();
   }

   public void buttonClick(View v){
    if(v.getId() == R.id.button_save){
       saveNote();
    } else if(v.getId() == R.id.button_new){
       newNote();
    } else if(v.getId() == R.id.button_list){
       listNotes();
    }
   }
}
```

Resource

- Additional files (other than code) used by app
- Types
  - layout
    - defines UI layouts and components
  - drawables
    - bitmaps and other graphics used in app
  - values
    - string constants
    - colors
    - styles, etc
  - other resources
    - menu, animations, etc

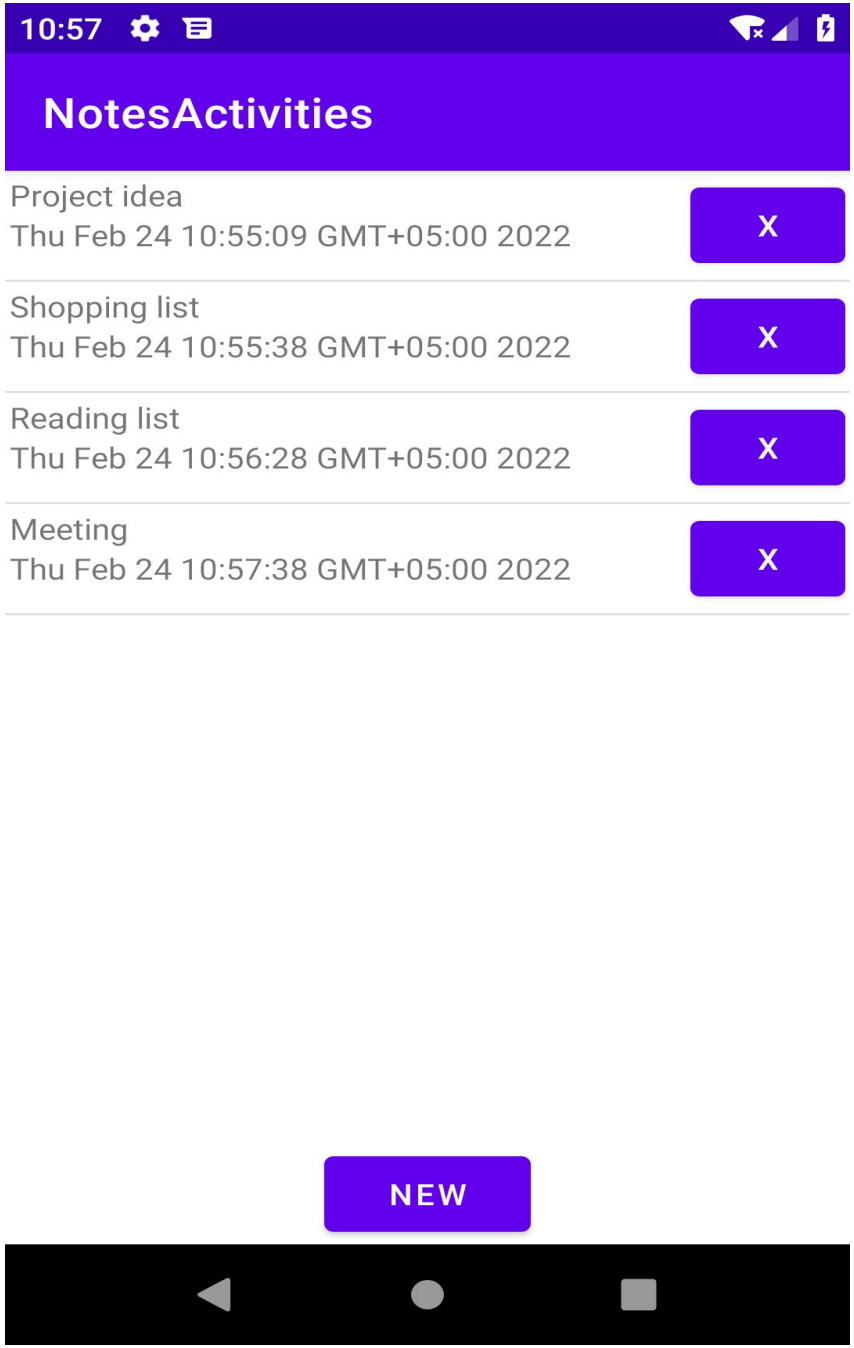
Strings

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Notes</string>
    <string name="new">New</string>
    <string name="save">Save</string>
    <string name="list">List</string>
    <string name="notes">Notes</string>
</resources>
```

UI Pattern: Lists and Grids

- Presents large data sets in form of lists or grids
- Features
  - Scrollable
  - Customization of UI as well as data source
  - Recycling support
- Implementation
  - Consider Adapter design pattern

Example: RecyclerView



Adapter Design Pattern

- Design pattern
  - Makes incompatible interfaces work with each other
  - Converts the interface to the one client expects
  - Allows creating decoupled general-purpose implementations that can work with each other using adapter
- Applicability
  - Adapters for third-party libraries

```
public class NoteAdapter extends
RecyclerView.Adapter<NoteAdapter.NoteViewHolder> {
   private ArrayList<Note> mDataset;

   public NoteAdapter(ArrayList<Note> ds) {
      mDataset = ds;
   }

   public NoteAdapter.NoteViewHolder
         onCreateViewHolder(ViewGroup parent,
                                    int viewType) {

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

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

   public void
         onBindViewHolder(NoteViewHolder holder,
                                 int position) {
      Note note = mDataset.get(position);
      String content = note.getContent();
      String ts = note.getTimeStamp();
      String title = content.substring(0,content.indexOf("\n"));
      holder.title.setText(title);
      holder.timestamp.setText(ts);
      holder.remove.setTag(position);
   }

   public int getItemCount() {
      return mDataset.size();
   }
```

```
// create as an inner class of NoteAdapter

  public class NoteViewHolder extends
    RecyclerView.ViewHolder {

    public TextView title;
    public TextView timestamp;
    public Button remove;

    public NoteViewHolder(View v) {
      super(v);
       title = (TextView) v.findViewById(R.id.title);
       timestamp = (TextView)
                        v.findViewById(R.id.timestamp);

       remove = (Button)
                    v.findViewById(R.id.button_remove);

       remove.setOnClickListener(
           new View.OnClickListener() {

              public void onClick(View v) {
                int pos = (int) v.getTag();
                mDataset.remove(pos);

                //adapter method
                notifyDataSetChanged();
              }

           });
    }

  } // end of view holder class

} // end of adapter class
```

```
public class MainActivity extends AppCompatActivity {
    private RecyclerView recyclerView;
    private RecyclerView.Adapter mAdapter;
    private RecyclerView.LayoutManager layoutManager;
    ArrayList<Note> dataSet = new ArrayList<Note>();


    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.list);
        recyclerView = (RecyclerView) findViewById(R.id.list);
        recyclerView.setHasFixedSize(true);

        // use a linear layout manager
        layoutManager = new LinearLayoutManager(this);
        recyclerView.setLayoutManager(layoutManager);

        // specify an adapter
        mAdapter = new NoteAdapter(dataSet);
        recyclerView.addItemDecoration(new DividerItemDecoration(this,
LinearLayoutManager.VERTICAL));
        recyclerView.setAdapter(mAdapter);
    }

    public void clickHandler(View v){
        if (v.getId() == R.id.button_new){
            Note note = new Note("Note " + (dataSet.size()+1) + "\nLorem ipsum ...");
            dataSet.add(note);
            mAdapter.notifyDataSetChanged();
        }
    }
}
```

Launching another Activity

- Using an Intent
  - An action intended for the android platform
  - One of the uses is to start another activity
  - Specify Activity class to be invoked
  - Use **ActivityResultLauncher** to launch a **child** activity returning a result back to **parent**
    - » **startActivityForResult** used **in older** APIs is deprecated

```
public class MainActivity extends AppCompatActivity  {
    // other attributes ...
    ActivityResultLauncher<Intent> notesLauncher;

    protected void onCreate(Bundle savedInstanceState) {
        // other UI creation related tasks
        //  ....

        //register
        notesLauncher = registerForActivityResult(
                new ActivityResultContracts.StartActivityForResult(),
                new ActivityResultCallback<ActivityResult>() {

                    public void onActivityResult(ActivityResult result)
{
                        if(result.getResultCode() == RESULT_OK){
                            Intent data = result.getData();
                            String id = data.getStringExtra("id");
                            // handle incoming data from child
                        }
                    }
                });

    }

    public void clickHandler(View v){
        if (v.getId() == R.id.button_new){
            Intent intent = new Intent(this,NotesActivity.class);
            intent.putExtra("id","");
            notesLauncher.launch(intent);
        }
    }
```

```
public class NotesActivity extends Activity
{
    EditText textArea;
    String noteId;

    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.note);
        textArea = (EditText) findViewById(R.id.text_area);

        Intent intent = getIntent();
        String content = intent.getStringExtra("content");
        textArea.setText(content);
        noteId = intent.getStringExtra("id");
    }

    public void saveNote(View v){
        Intent result = new Intent();

result.putExtra("content",textArea.getText().toString());
        result.putExtra("id",noteId);
        setResult(RESULT_OK,result);
        finish();
    }

    public void cancelNote(View v){
        Intent result = new Intent();
        setResult(RESULT_CANCELED,result);
        finish();
    }
}
```

UI Pattern: Searching and Filter

- Filter
  - A convenient widget for filtering result sets
  - Performs complex processing in a worker thread
  - Return result to UI thread
- User interaction
  - EditText combined with TextWatcher
  - Other views e.g. toggle buttons, SearchView, etc.

```
public class NoteAdapter extends RecyclerView.Adapter<NoteAdapter.NoteViewHolder> implements Filterable {
    private Filter filter;
    private ArrayList<Note> notes;
    private ArrayList<Note> filteredNotes;
    ...

    public Filter getFilter() {
        if (filter == null){
            filter = new NotesFilter();
        }
        return filter;
    }

    public class NoteViewHolder extends RecyclerView.ViewHolder {
        ...
        public NoteViewHolder(View v) {
            super(v);
             ...
            v.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    int pos = (int) v.getTag();
                    listener.onClick(filteredNotes.get(pos));
                }
            });
        }
    }

    public NoteAdapter(ArrayList<Note> ds, NoteItemClickListener ls) {
        notes = ds;
        filteredNotes = ds;
        listener = ls;
    }

    public NoteViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
       ...
    }

    public void onBindViewHolder(NoteViewHolder holder, int position) {
        String content = filteredNotes.get(position).getContent();
             ...
    }

    public int getItemCount() {
        return filteredNotes.size();
    }
    ...
}
```

Declared private as generally implemented as an inner class

```
private class NotesFilter extends Filter{

 @Override
 protected FilterResults performFiltering(CharSequence constraint) {
  FilterResults results = new FilterResults();
  if(constraint != null && constraint.length() > 0){
   ArrayList<Note> filteredList = new ArrayList<Note>();
   for(int i=0; i < notes.size(); i++){
    if(notes.get(i).getContent().contains(constraint)){
     filteredList.add(notes.get(i));
    }
   }

   results.count = filteredList.size();
   results.values = filteredList;

  }
  else{
   results.count = notes.size();
   results.values = notes;
  }

  return results;
 }

 @Override
 protected void publishResults(CharSequence constraint, FilterResults results) {
  filteredNotes = (ArrayList<Note>) results.values;
  notifyDataSetChanged();
 }

 }
}
```

```
search = (EditText) findViewById(R.id.search);
search.addTextChangedListener(new TextWatcher() {

    public void beforeTextChanged(CharSequence charSequence, int i, int i1, int i2) {

    }


    public void onTextChanged(CharSequence charSequence, int i, int i1, int i2) {
       filterable.getFilter().filter(search.getText().toString());
    }


    public void afterTextChanged(Editable editable) {

    }
});
```

Invoking filter

Basic state management

```
public class Note implements java.io.Serializable
{
    ...

}
```

```
public class MainActivity extends Activity
{
    ...

    public void onSaveInstanceState(Bundle savedInstanceState){
        super.onSaveInstanceState(savedInstanceState);

        try{
            savedInstanceState.putSerializable("noteslist",notes);
        }
        catch(Exception ex){ }

    }

    public void onRestoreInstanceState(Bundle savedInstanceState){
        super.onRestoreInstanceState(savedInstanceState);

        try{
            notes = (ArrayList<Note>) savedInstanceState.getSerializable("noteslist");
        }
        catch(Exception ex){ }

    }

    ...

}
```

Bundle available to **onSaveInstanceState** : suitable for lightweight transient data

View Model

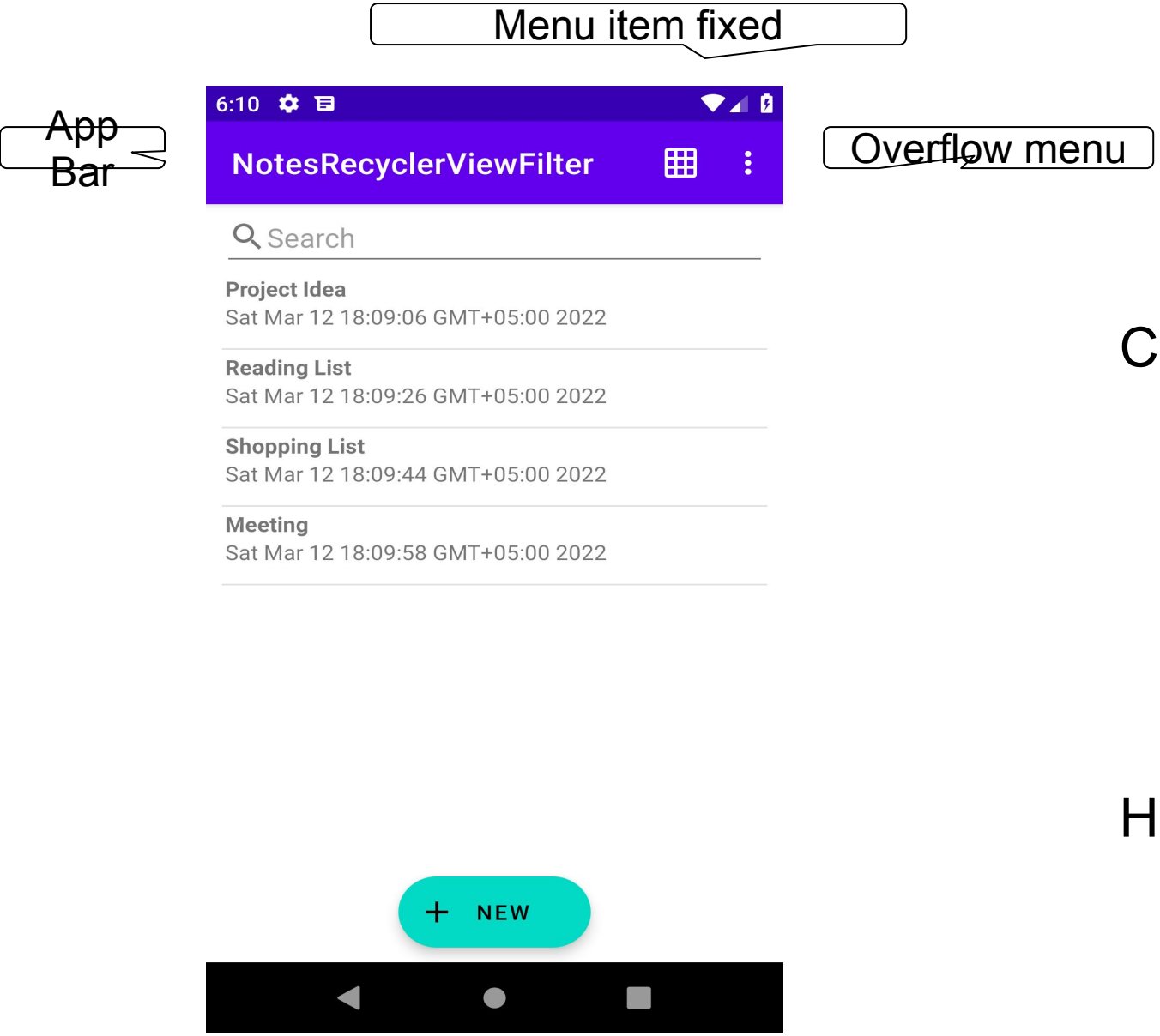
- In-memory storage structure for UI state
- Associated with Activity's life-cycle
- Survives screen rotations but not system-initiated death

```
public class UsersViewModel extends ViewModel {
    private List<User> users;
    public List<User> getUsers() {
        if (users == null) {
            users = new ArrayList<User>();
            // other related operations ...
        }
        return users;
    }
}
```

UI Pattern: Menus

- Options Menu
  - Primary collection of menu items for an activity
  - Manages global navigation for activity / application
  - Activities can enable / disable / change menu items
  - Application-wide generalization can be done through an abstract base Activity and a Template Method design pattern
- Context Menu
  - Floating menu that appears on long-click of an element
  - Provides options for manipulating a selected item
  - Contextual Action Mode can be used for manipulating multiple items simultaneously

Options Menu



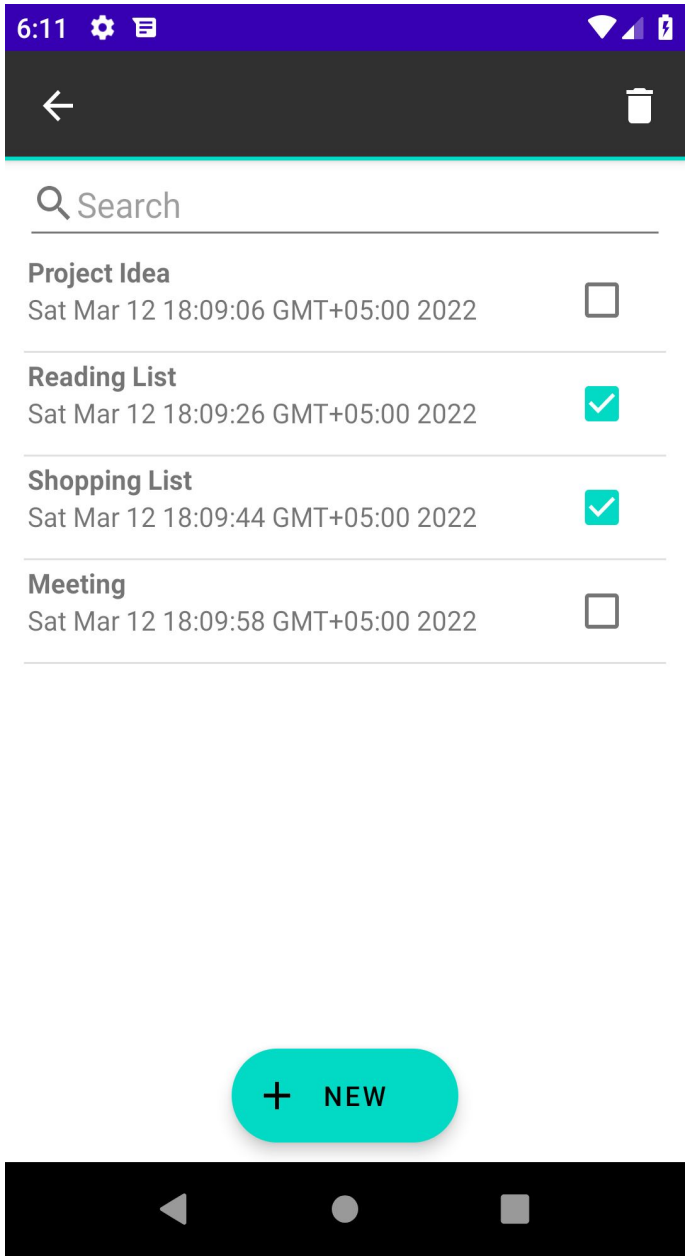
Create Menu Items

- menu resource file
- Override *onCreateOptionsMenu*
- Inflate menu using *MenuInflater*

Handle click events

- Override onOptionsItemSelected

Contextual Menu and Action Mode



Contextual action bar as an action mode

Recyclerview layout changed to allow selection

- Implement *ActionMode.Callback*
  - Inflate menu
  - Handle actions
- Initiate action mode
  - View's long-click listener
  - call *startActionMode*