Android dev academy Fragments, ViewPager, RecyclerView

...

Goran Luketić, COBE

Agenda

...

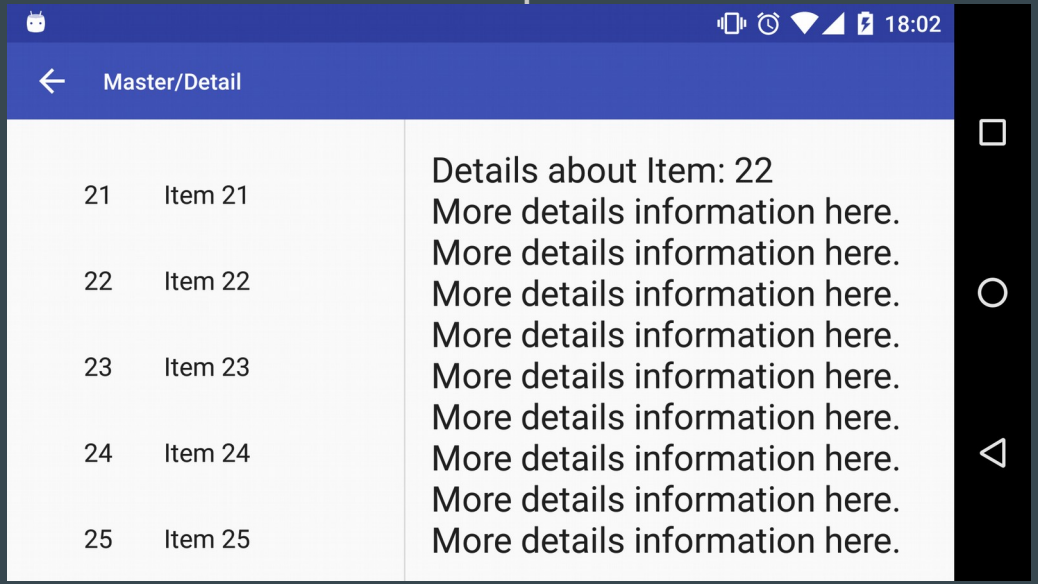
- Why fragments
- Lifecycle
- Fragment Transaction
- BackStack
- Handling state (fragment, view)
- ViewPager (PagerAdapter)
- TabLayout

What is Fragment

- A fragment represents a behaviour or a portion of user interface in an Activity.
- Android introduced fragment in android 3.0 (API 11)
- For lower API’s Fragments are inside support.v4 library
- To create a fragment, you must create a subclass of Fragment (or an existing subclass of it).
- Useful subclasses of fragment: DialogFragment, ListFragment, PreferenceFragment

Why?

- Multiple fragments can be combined in a single activity to build a multi-pane UI.
- Support dynamic and flexible UI design on tablets
- Fragment can be reused in multiple activities

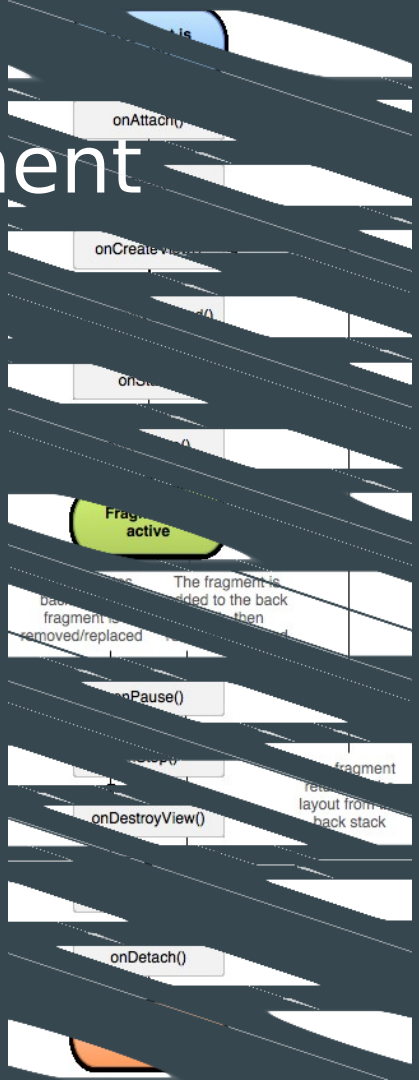


Lifecycle

- Fragment has his own lifecycle
- Since fragment must always be embedded in an activity fragment’s lifecycle is directly affected by the host activity’s lifecycle
- Fragment contains callback methods similar to an activity, such as onCreate(), onStart(), onPause() and onStop().
- If you are converting existing application to use fragments you might simply move code from activity’s callback methods into fragment lifecycle methods…

Lifecycle: Activity vs. Fragment

vs



Deep dive into fragment’s lifecycle

- onAttach() - called when the fragment has been associated with the activity (the Activity is passed in here before API 23. On API 23 Activity is deprecated and Context is passed. https://code.google.com/p/android/issues/detail?id=183358)
- onCreate() - the system calls this when creating the fragment. Within your implementation, you should initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed
- onCreateView() - The system calls this when it's time for the fragment to draw its user interface for the first time
- onActivityCreated() - activity's onCreate() method has returned
- onStart() - activity’s on start is called
- onResume() - the fragment is visible in the running activity
- onPause() - the first indication that the user is leaving the fragment
- onStop() - the fragment is not visible. Either the host activity has been stopped or the fragment has been removed from the activity but added to the back stack
- onDestroyView() - called when the view hierarchy associated with the fragment is being removed
- onDestroy() -activity’s onDestroy() is called
- onDetach() - called when the fragment is being disassociated from the activity

But that’s not all...

- onViewCreated() - called immediately after onCreateView, but before any saved state has been restored into the view. This method receives View as parameter and there is no return type as in onCreateView.
- onViewStateRestored() - introduced in Android API >= 17 . Called before onStart() and after onActivityCreated()
- onSaveInstanceState() - called to retrieve per-instance state from an activity before being killed so that the state can be restored in onCreate(Bundle)

Adding Fragment to an Activity

Two ways for adding fragment to UI:

- Through XML
- Programmatically (most common)

Fragment can also be added without UI supplying a unique string "tag" for the fragment, rather than a view ID)

Adding fragments through xml

- When the system creates this activity layout, it instantiates each fragment specified in the layout and calls the onCreateView() method for each one, to retrieve each fragment's layout.
- The system inserts the View returned by the fragment directly in place of the <fragment> element.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="horizontal"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <fragment
android:name="com.example.news.ArticleListFragment"
         android:id="@+id/list"
         android:layout_weight="1"
         android:layout_width="0dp"
         android:layout_height="match_parent" />
   <fragment
android:name="com.example.news.ArticleReaderFragment"
         android:id="@+id/viewer"
         android:layout_weight="2"
         android:layout_width="0dp"
         android:layout_height="match_parent" />
</LinearLayout>
```

Adding fragments programmatically

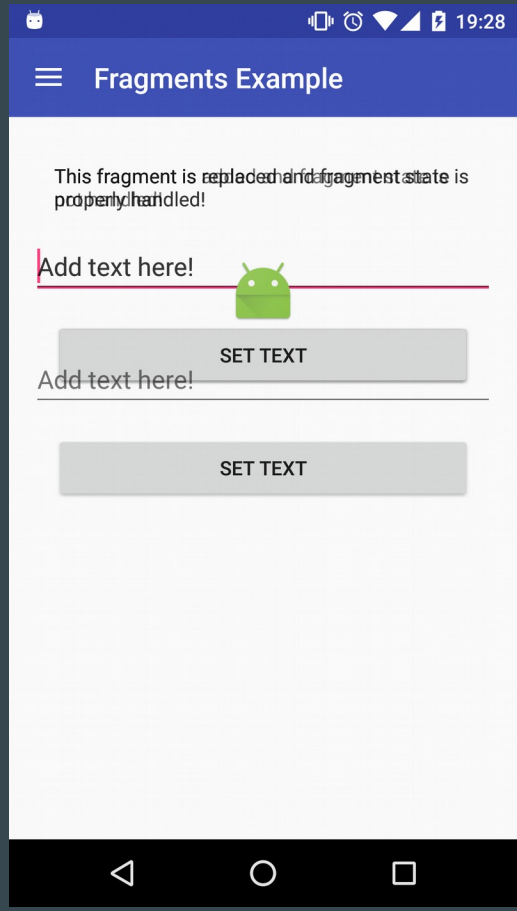
- At any time while your activity is running, you can add fragments to your activity layout. You simply need to specify a_ViewGroup in which to place the fragment.
- To make fragment transactions in your activity (such as add, remove, or replace a fragment), you must use APIs from FragmentTransaction.

https://developer.android.com/reference/android/app/FragmentTransaction.html

```
<FrameLayout
  android:id="@+id/fragment_container"
  android:layout_width="match_parent"

android:layout_height="match_parent" />
```

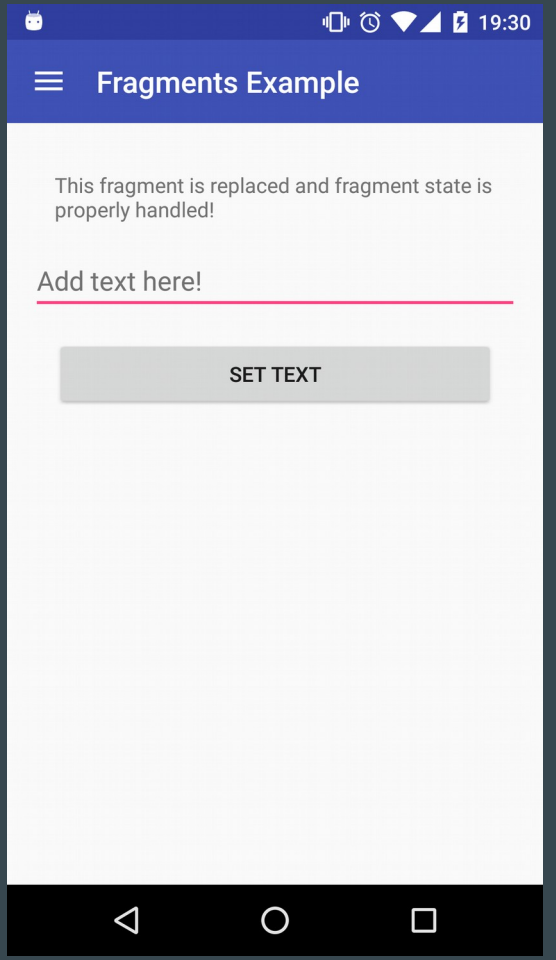
Fragment Trasaction: Add

```
protected void addFragment(int id, Fragment fragment) {
   FragmentManager fm = getFragmentManager();
   FragmentTransaction fragmentTransaction = fm.beginTransaction();
   fragmentTransaction
           .add(id, fragment, Extras.ADDED_FRAGMENT_TAG)

.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE)
           .commit();
}
```



Fragment transaction: Replace

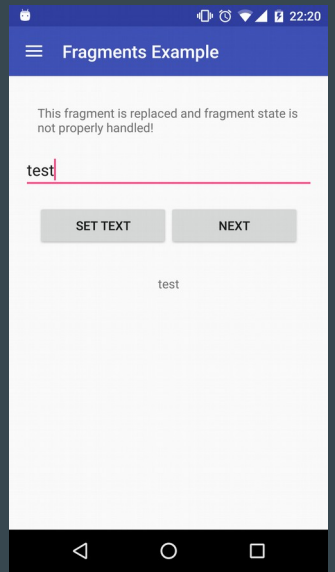
```
protected void replaceFragment(int id, Fragment fragment) {
   FragmentManager fm = getFragmentManager();
   FragmentTransaction fragmentTransaction =
fm.beginTransaction();
   fragmentTransaction
           .replace(id, fragment)

.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE)
           .commit();
}
```



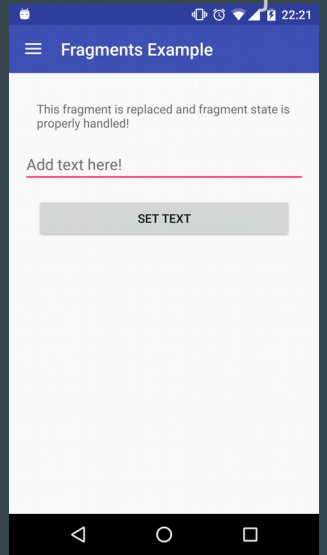
Add to BackStack

```
getFragmentManager().beginTransaction()
;
        .addToBackStack(null)
        .replace(id, fragment
        .commit();
```

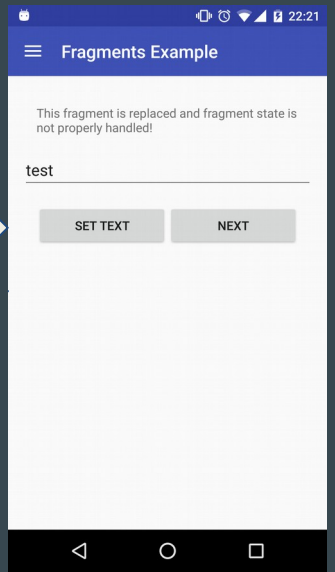
```
getFragmentManager().addOnBackStackChangedList
ener(
        new
FragmentManager.OnBackStackChangedListener() {
            public void onBackStackChanged() {
                // Update your UI here.
            }
```



NEXT



popBackStack()



When to commit transactions

IllegalStateException: Can not perform this action after onSaveInstanceState at android.support.v4.app.FragmentManagerImpl.checkStateLoss at android.support.v4.app.FragmentManagerImpl.enqueueAction at android.support.v4.app.BackStackRecord.commitInternal at android.support.v4.app.BackStackRecord.commit

- You should not call commit on a fragment transaction before an activity loads it's savedInstanceState or after it saves it's savedInstanceState

Safe commits:

- onCreate, onResumeFragments

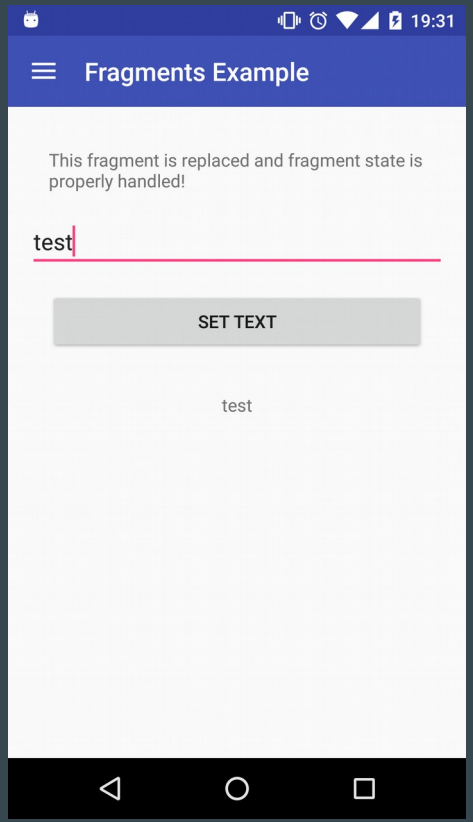
Unsafe commits:

- onActivityResult
- onRequestPermissionResult
- Async callbacks

Handle state

FRAGMENT state

mText



VIEW state

mEditView

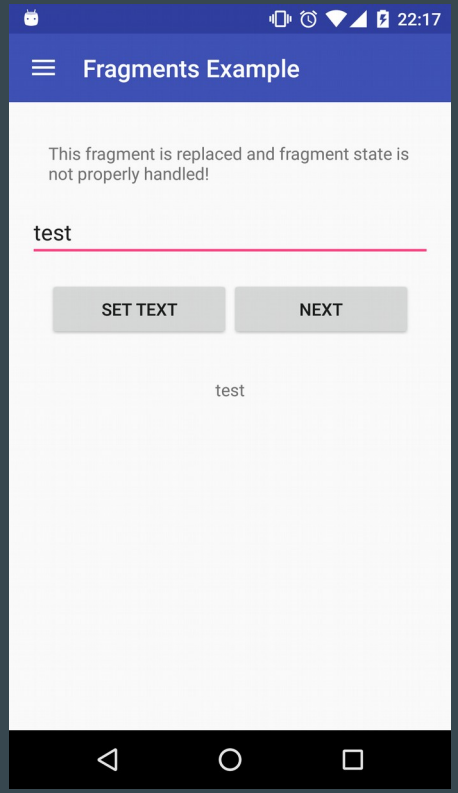
mResultView

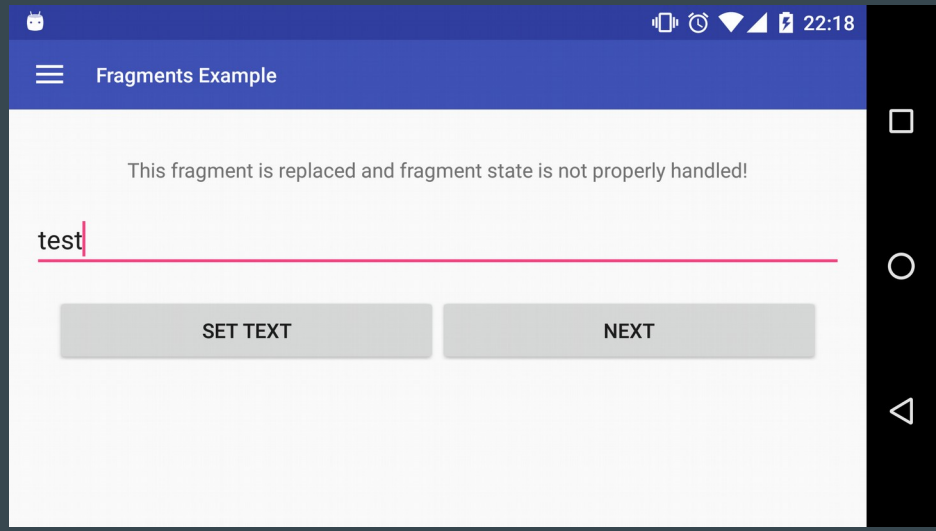
Simple example

```
private Unbinder mUnbinder;


@Nullable
@Override
public View onCreateView(LayoutInflater inflater,
ViewGroup container, Bundle savedInstanceState) {
   View v = inflater.inflate(R.layout.fragment_add,
container, false);
   mUnbinder = ButterKnife.bind(this, v);
   return v;
}


@Override
public void onDestroyView() {
   super.onDestroyView();
   mUnbinder.unbind();
}
```

```
@Optional
@OnClick(R.id.btn_set)
void handleSetBtn() {
   if (mEditView != null) {
      setText(mEditView.getText().toString());
   }
}


private void setText(String text) {
   if (mResultView != null) {
      mResultView.setText(text);
   }
}
```

Handle orientation - wrong



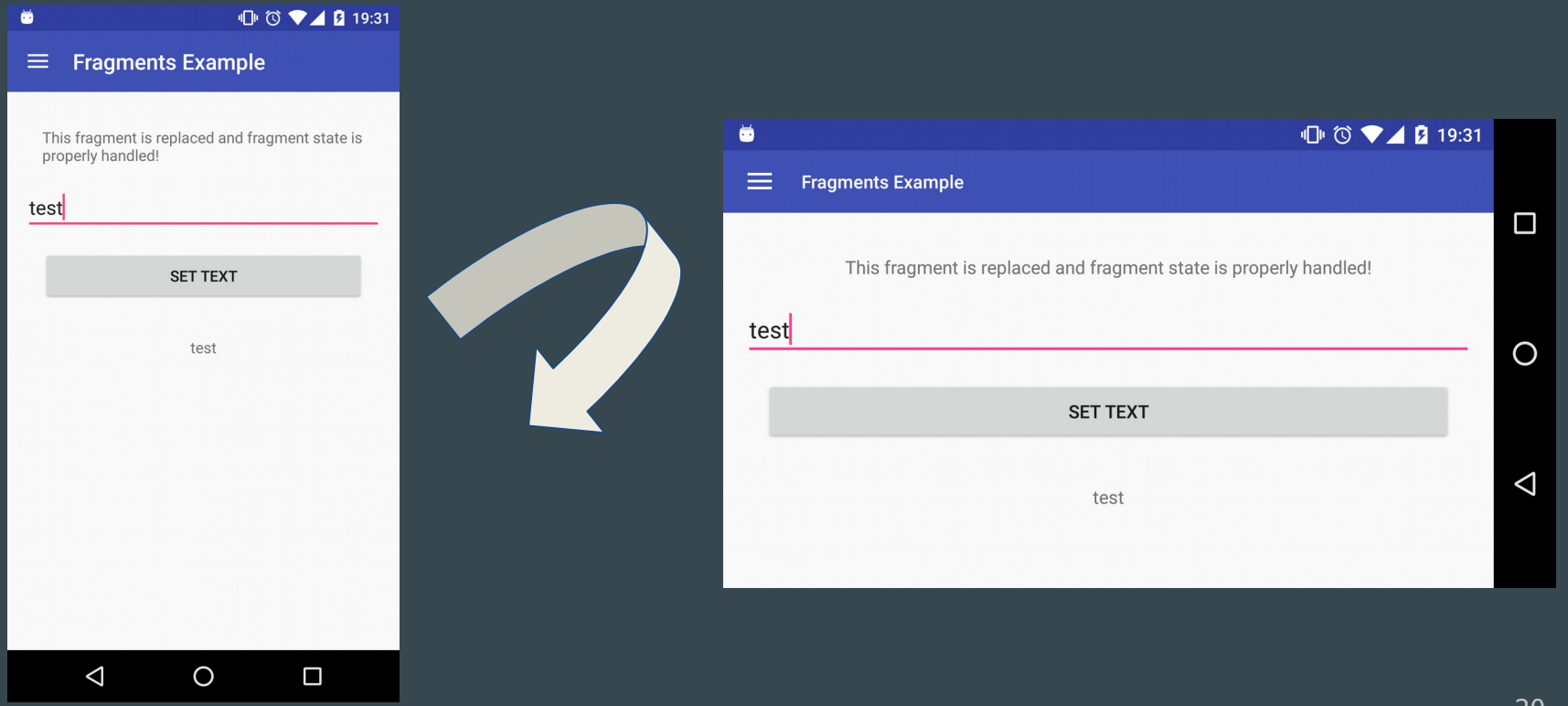


Example: handle orientation change

```
@Override
public void onSaveInstanceState(Bundle
outState) {
   super.onSaveInstanceState(outState);
   outState.putString(Extras.EXTRA_TEXT, mText);
}
```

```
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   if (savedInstanceState != null) {
      if
(savedInstanceState.containsKey(Extras.EXTRA_TEXT)) {
         mText =
savedInstanceState.getString(Extras.EXTRA_TEXT);
      }
   } else {
      Bundle b = getArguments();
      if (b != null) {
         if (b.containsKey(Extras.EXTRA_TEXT)) {
            mText = b.getString(Extras.EXTRA_TEXT);
         }
      }
   }
}
```

Handle orientation - correct



Handle state - Summary

- Don’t mix fragment and view state
- Fragment’s state - onCreate, onSaveInstanceState
- View’s state - onCreateView, onDestroyView
- Be careful with activity state

ViewPager

ViewPager

- Allows to move forward or backward through the pages by swiping across the screen.
- - You don’t have to deal with the slide animation and the swipe gesture detection.
- - Most common way is to use fragments for pages
- - Located only in support library
-
- <android.support.v4.view.ViewPager
- android:id="@+id/viewPager"
- android:layout_height="match_parent"
- android:layout_width="match_parent" />
- - Fragment instances are connected with the ViewPager using a PagerAdapter, which is an object that sits between the ViewPager and the data set containing the information you want the ViewPager to display

PagerAdapter - types

- **FragmentPagerAdapter** stores the fragments in memory as long as the user can navigate between them. When a fragment is not visible, the PagerAdapter will detach it, but not destroy it, so the fragment instance remains alive in the FragmentManager. It will release it from memory only when the Activity shuts down. This can make the transition between pages fast and smooth, but it could cause memory issues in your app if you need many fragments.
- **FragmentStatePagerAdapter** makes sure to destroy all the fragments the user does not see and only keep their saved states in the FragmentManager, hence the name. When the user navigates back to a fragment, it will restore it using the saved state. This PagerAdapter requires much less memory, but the process of switching between pages can be slower.

PagerAdapter - Example

```
class TaskPagerAdapter(fm: FragmentManager): FragmentPagerAdapter(fm) {


    override fun getItem(position: Int): Fragment {
        return when(position){
            0 -> TasksFragment.newInstance()
            1 -> Tasks2Fragment.newInstance()
        }
    }


    override fun getCount() = 2
}
```

```
val pagerAdapter = TaskPagerAdapter(supportFragmentManager)
viewPager.adapter = pagerAdapter
```

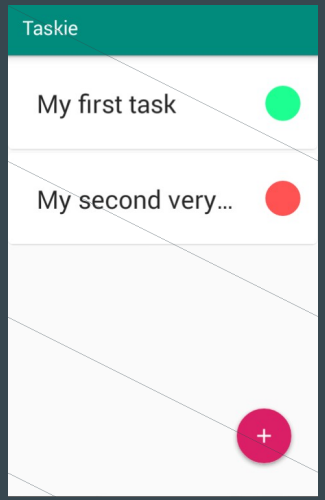
TabLayout

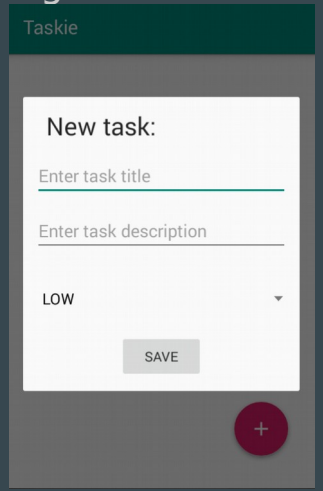
- makes it easy to explore and switch between pages.
- contains a tab for each page, which usually displays the page title.
- The user can tap on a tab to navigate directly to the desired page or can use a swipe gesture over the TabLayout to switch between pages.

```
<android.support.design.widget.TabLayout
        android:id="@+id/sliding_tabs"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:tabMode="fixed" />

tabLayout.setupWithViewPager(viewPager);

@Override
 public CharSequence getPageTitle(int position) {
        return tabTitles[position];
    }
```

Zadatak

- Postojeću aplikaciju (Taskie) prepraviti tako da glavni screen bude lista taskova.
- Cijeli screen neka bude napravljen unutar fragmenta
- Listu taskova napraviti pomoću RecyclerView-a.
- Dodati gumb za dodavanje novog task-a
- Dodavanje novog taska prepraviti tako da se otvara Dialog umjesto novog activitija
- Detalje pojedinog taska premjestiti unutar fragmenta. Taj fragment ucitavati unutar activitija koji će sluziti kao container i imat ce mogucnost dodavanja bilo kojeg fragmenta unutar sebe





Zadaća.

- Dovršiti sve stavke iz zadataka s predavanja
- Dodati navigaciju unutar aplikacije (BottomNavigation)
- Prvi navigation item neka bude lista taskova napravljena na predavanju
- Drugi navigation item neka bude "About"
- Unutar about screena dodati viewpager gdje ce se swipeati izmedju podataka o aplikaciji i autoru aplikacije. Dodati TabLayout kako bi user znao na kojem se tabu nalazi