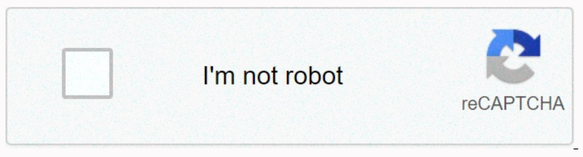

Continue

Android back button fragment

I think the easiest way is to create an interface and check if the track is of the interface type in Activity and, if so, look for its method to process pop. Here is the interface to be applied in the part. public interface BackPressedFragment { // Note for this to work, the name and tag must be set when the part is added to the back stack, for example // getActivity().getSupportFragmentManager().beginTransaction() // replace(R.id.fragment_container, MyFragment.newInstance(), MY_FRAG_TAG) // addToBackStack(MY_FRAG_TAG)) // .commit(); This is really an override. You should call popBackStack itself. void onPopBackStack(); } Here's how you can apply it. public class MyFragment supports Fragment BackPressedFragment @Override public void onPopBackStack() { /* Your code goes here, do everything you want. */ getActivity().getSupportFragmentManager().popBackStack(); } And activity, pop handled (probably both onBackPressed and onOptionsItemSelected), pop backstack using this method: public invalid popBackStack() { FragmentManager fm = getSupportFragmentManager(); // Look for onPopBackStack of the current track, if any. String fragmentTag = fm.getBackStackEntryAt(fm.getBackStackEntryCount() - 1).getName(); Fragment currentFragment = getSupportFragmentManager().findFragmentByTag(fragmentTag); if (currentFragment instance of BackPressedFragment) ((BackPressedFragment)currentFragment).onPopBackStack(); else fm.popBackStack(); } This is quite basic Java but is actually quite necessary for any part heavy application. My apps stick to one or two activities, and everything else is properly stored in modified parts as needed. It's less code complexity for me but find it lead to more expandability, but mileage and all that ... I have a standard container layout of XML, and the following call for swap pieces in:Of course adding to the back stack that opens when we press back by default, which is just nice. What was R.id.content_container another container in the same layout as our framework layout, but was it added in the same way? Let's just say we add content immediately after R.id.side_bar_container above. In that case, if I push the back button, it'd be nice if he blew both pieces at the same time, wouldn't it? Likewise, both parts can perform some extra code when the user hits back, and it would be nice if only the hosting activity we know when we press back. He must share this information to everyone he wants to know! Solution To tell the parts when the back button is pressed, you first need a basic part that it inherits from the permission of other parts. This basic part applies the following:Now in your hosting Event, search for any app of the interface before any part is detonated:That's it! Any part of the OnBackPressed() app can do this and activity will be told as soon as possible. I mean, in my case, to blow both pieces the same way. I just have an extra pop in the sidebar (which was last added) :P roblems or suggestions? You know what to :) Last Updated 24.6.2020This article describes how FragmentBack works under the heading. If you are looking much more prepared to use the solution than use this small (1.5 kB) library doing it yourself instead. In addition to the functionality described here, you can also define a priority for your parts in the FragmentBack library. BaseFragment The first step to creating back-sensitive parts is to define the interface that we can call to report specific parts about the back press. Includes a single method on BackPressed() that returns a value indicating whether the backpress event is consumed by the part. In this example, we use a base class, but you can also define it through the interface. public class BaseFragment Fragment { /** * Can hold back. * True when @return is pressed back */ public boolean onBackPressed() { return false; } Basic Activity The next step is an activity with the overwrwn Activity.onBackPressed() method. In its body, we list all the parts linked to the activity and for this we provide them with information about the new backpress event to implement our BaseFragment class/interface. If the reported part can show event consumption by returning correctly, in this case, it stops duplying and abandons the method. If no part events are consumed, we call the standard Activity.onBackPressed() app to close the activity. Now only by extending this BaseActivity will all back-press events be spread placed in all BaseFragments. public class BaseActivity, Activity { @Override public space onBackPressed() { List fragmentList = getSupportFragmentManager().getFragments(); boolean handled = false; for(Fragment f : fragmentList) { if(f instance of BaseFragment) { if(f instance of BaseFragment) { handled = ((BaseFragment)f).onBackPressed(); if(handled) { break; } } if(handled) { super.onBackPressed(); } The last step of MyFragment is to implement the back-sensitive part. There is nothing simpler, basefragment class/interface expand/implement and onBackPressed() handle back-press events that return smoothly or simply incorrectly to ignore it. public class MyFragment BaseFragment { /** * Send back from printed activity. * * If @return event is consumed, it will return correctly. */ @Override public boolean onBackPressed() { if (openedLine < 0) { return false; } else { closeDetail(); return true; } Finished! Need more detailed help? An advisory book: If you are into Android let us chat about it, do not hesitate and let us know your opinions in the comments section below. We also tend to search for android giant (iOS de), but we also have a chance to test a project manager or. Send us an email: skoumal@skoumal.net. How hard it was before to handle BackPress in pieces! Recently the giantSummit19 was introduced in the Android world one of the beautiful things was a new way always presses back the pieces that were a little bit of pain to apply. Previously a trailer, let's assume that a SearchFragment was required to respond back to press events and close SearchView, we had to review all the steps such as:Old:1-We should have had an interface:Interface: An interface that every part willing to block backpressed() event should implement2- All parts that wanted to block the BackPress event had to implement the interface that caused them to have onBackPressed() function. Now track BackPress is based on if they can respond to events and do things and if the event is consumed or not they can rotate right or wrong. Fragment:Fragment is the method required to process this backPressed() event:Logic backPress SearchFragment3-Now to apply logic logic to connect the point and process the activity when the time is clicked to override the OnBackPressed() function, we need to get the current part from fragmentManager and examine whether the type is BackPressHandler and if so we execute the onBackPressed() function on that particular part and check the response, if you are returning incorrectly, the next move is to call super.onBackPressed() or make a certain logic at the event, but if it turned correctly, then the track means that the event was not consumed and the event consumed, which means that we should skip the activity onBackPressed() logic execution , Let's See Action How to do it in:The structure of MainActivity's onBackPressed() function and if you're wondering what's inside the getTopFragment extension function:You can see that the Kotlin function extension is just a lot of work for something simple to get the track on top In a new way of doing this now we just need to add the following code to Fragment's onCreate() method:Fragments backpress processing the final method of demonstration, now all back-press events will be captured with handleOnBackPressed() method and parts will have the opportunity to react to Back Press events. But what about when we're not sure we need to consume back-press activity? Once we just need to check some situation and if then it applies to take some action? In such cases we can set the isEnabled variable to incorrect and look for the onBackPressed() method at the event manually again:Well, it is much easier to achieve this by saving OnBackPressedCallback samples to handle ComponentActivity.onBackPressed() callback with this component. All code is available on Github:Please share and :) You like the article 👉 button, Happy coding 👨‍💻👩‍💻Since: use Gson in your projects? Then maybe it's time 🚀🚀🚀 get away from 🚀🚀🚀: