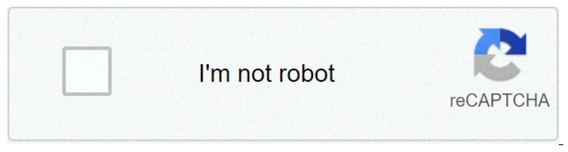

Continue

Start activity in kotlin

I learned Ã ° Ã¿ ~ "Hard way: there should be better ways to launch the activities of Kotlin Android. Go the Extra data through intent for serialization and de-serialization activity destination. Therefore, I have read many articles and pick up a couple of great ways to launch Android activity. the start-up activities in the Android app is a common task and a different developer uses different approaches. we now see the various types one by one. there traditional way are developers who still use the old way to launch an activity. we look at the typical traditional way of starting an activity. VAL = INTENT INTENT (context altertattivitÃ :: class.java) STARTATTIVITÃ (INTENT) Now if we need to overcome a topic intent becomes messy and unfriendly experienced developer. the main point is the serialization issues, de-serialization, security type and zero controls. the Val intent = ntent (context, Overctivity :: class.java) Intent ("name", user.name) Intent ("Email", utente.email) Intent ("ID", user.uuid) STARTATTIVITÃ (INTENT) The AL altrattivitÃ we need to do a lot 'of things. Leta â ¢ s See: Class Overctivity: AppCompaTattivitÃ () {Fun Override OnCreate (SavedInstancestate: Bundle) {super.onCreate (SavedInstancestate) setContentView (R.Layout.Activity_Ather) Val Name = intenzione.GetStringextra ("name")? : IllegalStateException ("name field is missing in order") and Val-mail = intenzione.getStringextra ("email") ?: Throw IllegalStateException ("E-mail of missing field in intention") Val id = intenzione.getStringextra ("ID ") ?: Throw illegally)? "Missing ID field in order")}} The example above will work for our example contrived but we can improve it? © Because there is always a better way to solve the problem. Creating a launch function prolongs Keyword reified The method useriamo is originally published here on this link. In this method, we create an extension function for the activity and the context class. I can show you how to launch activities with this approach. Inline Fun ACTIVITIES .Lainch (RequetCode: INT = -1, Options: Bundle? = NULL, INIT NOINLINE: Intent. () -> Unit = {}) {Val Intent Intent = (this) intenzione.init () if (build.version.sdk_int> = build.version_codes.jelly_bean) StartactivityforResult (Intent, RichiestaCode, options) Other STARTATTIFICEFORRESULTURSULT (INTENT, OPTIONAL)} Inline Fun context.Launch (options: Bundle? = null, init noinline: Intent. () -> Unit = {}) {Val Intent Intent = (this) intenzione.init () if (build.version.sdk_int> = build.version_codes. jelly_bean) STARTATTIVITÃ (INTENT, OPTIONS) Other startActivity (Intent)} Inline Fun Intent (context: the context): = Intent Intent (context, t :: class.java) After adding the extension function It can directly Use .Launch () in a task or in a classroom setting and use it this way: // in context reference reference class context.Launch () // in business class Activity of riferimento.Launch () // Pass Topics .launc activities. h {putextra ("name", user.name) putextra ("email", user.email) putextra ("ID", user.id)} // Start Activity for the result Activity .Launch (RequestCode = 25) // launch Activities for the result with Activity Argomenti.Launch (RichiestaCode = 25) {PUTEXTRA ("Name", User.Name) PUTEXTRA ("Email", User.Email) Plessextra (" ID ", user) .id)} // Start with transitions shared Val Options = ActivityOptionsCompat.makescenetrasitionanpat.makesCenetrasitionanimation (Activity, Profilepic," Profile ") Activity .Launch (Options = Options) InchiodÃ² Ã¿ Ã" Â ¨ Right! But for me, there is nothing but a better stagionazione () © because we still need to The extra parameter within the alterity class. Attractive class: AppCompatattività () {Override fun oncreate (SaveDinstancestate: Bundle) {super.oncreate (Saviewinstancestate) SetContentView (r.layout.Activity_ate) Val Name = intent.getstringeXtra ("name")?: Throw illegalstateexcetion ("Field missing intent ") Val eMail = = ?: Throw illegalstateException ("Missing field e-mail intention") Val ID = intention.getstringeXtra ("id")?: Throw illegalStateException ("missing field ID in intent")}} Leave it another to try and reduce it The boiler code to do deserialization. Launch activity with the Args parameter and the deserialization intent Let go crazy and create an ideal data class for our extra parameter to make serialization and deserialization. User of the Data Class (name of the Val: String, Val e-mail: String, Val uuid: UUID) Class ActivityArgs Costructor (Value Value: User) {Companion Object {Private CST Val Name = "_Name_" Private Const Val eMail = " _Email_ "Private Val Id =" _ID_ "Fun DeserieriLizefrom (Intent: Intent): User {return user (name = intention.getstringextra (name), e-mail = intention.getstringextra (e-mail), uuid = uuid.fromstringerextra (ID))}}} Next, create a generic ActistArgs interface with some custom methods. Actistisargs interface {fun intent (context: context): intent fun launch (context: context, options: bundle? = Null) = context.launch (intention = intention (context), options = options) Funny launch (activity: Activities, Options: bundle? = Null, requestcode: int = -1) = activity .launch (intent = intent (activity), requestcode = requestcode, options = options)} Now to make the creation of intention, we must implement the Actistings interface on Altriatactyargs class. OTHERACTIVITYARGS CLASSE Manufacturer (Private Val user: User): ActivityArgs {Companion Object {Private Const Name Val = "_Name_" Const Val Private Email = "_Email_" ID Const Val private = "_ID_" fun deserializefrom (intent intent): User {user Return (name = intention.getstringextra (name), e-mail = intention.getstringextra (e-mail), uuid = uuid.fromstring (intention.getstringextra (id)))}} override fun entertainment (context: context) = intent (context, aluitivity :: class.java). Apply {PUTEXTRA (name, user.name) PUTEXTRA (e-mail, user.email) PUTEXTRA (ID, user.uid.toString ())}} Before starting to use the ActistionARGS class we have to update our extension functions for The class of context and activity. Entertainment intent, requestcode)} fun context.launch (options: bundle? = null, intent: intent) {if (build.versiurs.sdk_int> = build.version_codes.jelly_bean) Startactivity (Intent, Options) Otherwise StarTactivity (Intent)} later This we can make some boot activities of the previous example as follows: Val args = oversactivityartargs (user) // pass the user object for intent creation args.launch (activity = this) later, we can do the deserialization easily in Otheractivity class. OTHERACTIVITY CLASS: AppCompatActivity () {User Val Private from Lazy {OtheractivityArgs.DesiaLizefrom (intention)} Override of fun Oncreate (SaveDinstancestate:? Bundle) {super.oncreate (saviewinstancestate) setcontentview (r.layout.Activity_other) // Use the User Object}} Boom! We have a solution that is easily maintained ... "There is only a class that manages serious and deserial arguments for the Lativeness class. Conclusion: The good thing is that, the method of construction intent does not require you to explicitly set the Context. It can only be called in or in context classes. So what do you think of the start-up activities, serialization and deserialization of data? I excite get your feedback or tips to further improve these methods. Please let me know through the Comments section. Thank you for being here and continue to ... from the beginner to advanced, our recommended coding formation is Treehouse. Treehouse is an online training service that teaches web design, web development and app development with videos, quizzes and interactive encoding exercises. The Treehouse mission is to bring technological training to those who cannot get it, and is committed to helping his students students jobs. If you are trying to activate the coding in your career, you should consider the tree tree. Disclosure of the material connection: some of the links in the above post are "frizzed connections". This means that if you click on the link and buy the item, we will receive an affiliate commission. Regardless of what, we recommend only products or services that we use personally and believe we can add value to our readers. When completing the previous lesson, you have an app that shows a business that consists of a single screen with a text field and a send button. In this lesson, you add a little code to the mainativity that starts a new activity to view a message when the user touches the Enter button. Note: This lesson presupposes that you use Android Studio V3.0 or higher. Follow these steps to add a method to the MainActivity class called when the send button is covered: in the App app> java> com.example.myfirstapp> MainActity, add the following method sendmessage () Stub method: Class Maineactivity: AppCompatativeness () {Override fun oncreate (SaviDinstancestate: Bundle?) {SUPER.ONCREATE (SAVIENSTANTATE) SETCONTENTVIEW (R.LAYOUT.ACTIVITY_MAIN)} / ** Named when the user touches the send button * / Fun SendMessage (View: View) {// Do something in response to button}} MainEtivity of the public class extends AppCompativity {@override protected void Oncreate (SaviDinstancestate bundle) {super.oncreate (SaviDinstancestate); setcontentview (r.layout.Activity_main); } / ** called when the user touches the send button * / public void sendmessage (View View) {View view) {// do something in response to button}} You can view an error because Android Studio can not Solve the display class used as a view topic. To delete the error, click the View statement, place the cursor on it, then press ALT + ENTER or OPTION + ENTER on a MAC, to perform a quick correction. If a menu is displayed, select Import class. Return to the Activity_Main.xml file to call the method from the button: select the button in the layout editor. In the Attributes window, locate the OnClick property and select SendMessage [mainalty] from your drop-down list. Now when the button is exploited, the system calls the SendMessage () method. Take note of details in this method. They are required for the system to recognize the method as compatible with the Android attribute: OnClick. Specifically, the method has the following features: Public access. An empty or, at Kotlin, a return value of the implicit unit. A view as a single parameter. This is the view that you clicked on the end of step 1. then, fill in this method to read the content of the text field and deliver that text to another activity. Build an intent An intent is an object that provides runtime binding between separate components, such as two activities. The intent represents an intention of an app to do something. You can use the intent for a wide variety of tasks, but in this lesson, your intent starts another activity. In the Mainativà, add the EXTRA_MESSAGE constant and the SendMessage () code, as shown: CONST VAL EXTRA_MESSAGE = "COM.EXAMPLE.MYFIRSTAPP.MESSAGE" MAINEACTIVITY CLASS: AppCompativity () {OVERRIDE ONCREATE (SAVIENSTANSTANCESTATE: BUNDLE?) {SUPER.ONCREATE (SaveDinstancestate) setcontentview (r.layout.Activity_main)} / ** called when the user steal the send button * / fun sendmessage (View: View) {Val EditText = Findviewbyid (r.id.edittittexttextpersonname) Val Message = EditText.text.Tostring () Val Intent = Intent (this, DisplayMessageActivity :: Class.java) {PUTEXTRA (EXTRA_MESSAGE, MESSAGE)} STARTActivity (INPT)}} MainEttivity of the public class Extend AppCompatativity {Final String Static Public EXTRA_MESSAGE = "com.example.myfirstapp.message"; @Overtinstancestate blank protected override (super.oncreate (SaviDinstancestate); setcontentview (r.layout.Activity_main); } / ** called when the user touches the send button * / Public Void SendMessage (View View) {Intent Intent = New Intent (this, DisplayMessageActivity.class); Displaymessageativity.class); EditText = (EditText) Findviewbyid (r.id.edittexttextpersonname); MESSAGE STRING = EditText.gettettext (). Tostring (); intention.Putextra (extra_message, message); STARTATIVITY € (intent); }} Android aspect Study for the meeting cannot resolve the symbol errors again. To delete errors, press Alt + Enter or Option + Return to a Mac. Your should end with the following imports: IMPORT Androidx.appcompat.app.AppCompatActivity Import Android.Content.intent import Android.S.Bundle Import Android.View .View import Android.widget.EditText Import Androidx.appcompat.app.appompatactivity; IMPORT Android.content.intent; IMPORT Android.os.bundle; Import Android.View.View; IMPORT Android.widget.Editext; An error persists for DisplayMessageActivity, but it's okay. To solve the problem in the next paragraph. Here is what is happening in Sendmessage (): The Intent manufacturer accepts two parameters, a context and a class. The Context parameter is used first because the activity class is a subclass of context. The class parameter of the app component, so the system offers intent, in this case, the activity to start. The PUTEXTRA () method adds the value of the EditTEXT to the intent. A guy can carry intent data as a key-value pairs called extra. Your key is a constant extra_message public, because the next activity uses the key to retrieve the text value. It is a good practice to define the keys for the extras intent with the name of your app package as a prefix. This ensures that the keys are unique, if your app interacts with other applications. The STARTActivity () method starts an instance of the displayMessageActivity which is specified by the intent. Subsequently, you need to create that class. Note: Component architecture navigation allows you to use the navigation editor to associate one business with another. Once the report is done, you can use the API to start the second activity when the user activates the associated action, such as when the user clicks on a button. For more information, see Navigation. Create the second activity to create the second activity, follow these steps: In the Project window, right-click on the Application folder and select New> Activities> Vacuum Activities. In the activity configuration window, enter "DisplayMessageActivity" for the name of activity. Leave all the other properties set to the default values and click Finish. Android Studio automatically performs three things: create the displaymessageActivity file. Create Activity_Display_Message.xml Layout file, which matches the DisplayMessageActivity file. Adds the required element in AndroidManifest.xml. If you run the application and touch the button on the first activity, the second starts activity, but it is empty. This is because the second activity uses the empty layout provided by the model. Add a text view Figure 1. The text view centered at the top of the layout. The new activity includes an empty layout file. Follow this procedure to add a text view to the point where the message is displayed: Open the App file> RES> Layout> activity_display_message.xml. Click Enable AutoConnection for Parent in the toolbar. This allows Autoconnect. See Figure 1. In the Palette panel, click Text, drag a TextView into the layout, and release it near the upper-center of the layout so that you click on the vertical line that appears. AutoConnect adds left and right restrictions in order to put the view in the horizontal center. Create a more constraint from the top of the text view at the top of the layout, so that it appears as shown in Figure 1. Alternatively, You can make some changes to the text style if you expand Textappearance in the common attributes of the window attribute panel, and the changing attributes such as Textsize and TextColor. View the message at this stage, you change the second activity to view the message that has been passed from the first action. In DisplayMessageActivity, add the following code to the Oncreate method (): Oncreate fun override (SaviDinstancestate:? Bundle) {super.oncreate (SaviDinstancestate) (SaveDinstancestate) // Get the intent that you started this activity and extract the message of the Val String Val = intent.getStringeXtra (extra_message) // capture the Layout's TextVist and set the string as a text Val TextView = FindViewbyid (R.ID. TextView). Apply {text = message}} @override protected void oncreate (SaviDinstancestate bundle) {SUPER.ONCREATE (SAVIENSTANCESTATE); setcontentview (r.layout.Activity_display_message); // Get the intent that started this activity and extract the intent of the string fee = getTentent (); String message = intention.getstringextra (mainactity.extra_message); // Capture layout TextView and set the string as text TextView TestView = FindViewbyid (R.DextView); TEXTVIEW.SETTEXT (Message); } Press ALT + ENTER or OPTION + RETURN ON A MAC, to import these other necessary classes: Import Androidx.AppCompat.App.AppCompatattività Import Android.content.intent IMPORT Android.S.Bundle IMPORT Androidx. Widget.textView Import AndroidX. Appcompat.app.AppCompatattività; IMPORT Android.content.intent; IMPORT Android.os.bundle; IMPORT Android.widget.textView; Add upward navigation Each screen in your app that is not the main input point, which are all the screens that are not the initial screen, they must provide navigation that directs the user to the logical parent screen in the app hierarchy. To do this, add a button up in the app bar. To add a button to, you need to declare what activity is the logical parent in the AndroidManifest.xml file. Open the file on App> Manests> AndroidManifest.xml, locate the tag for DisplayMessageActivity and replace it with the following: Android The system now automatically adds the button to the app bar. Run the app Click Apply changes in the toolbar to run the app. When you open, type a message in the text field and tap Send to see the message appear in the second activity. Figure 2. Open app, with text inserted on the left screen and displayed on the right. This is everything, you have built your first Android app! To continue learning the basics on the development of the Android app, returns to build your first app and follow the other links provided there. lÃ.