Runnable interface android studio

At the time of the launch of the itselef app, the system creates a stream called the main one in the app. This main stream is also called the UI stream, and it will only interact with the components of the Android user interface toolkit. Here's an example of using a ranchhide stream in a click event, When you click on the image will be uploaded, as it will take time to upload the image, we can use the stream as shown below: Javapublic Void onClick (View v) - a new thread (new Runnable () - public void start () start ()) to start (); Here we use the runable stream with the help of the handler. Javapublic class ProgressThread expands the activity implements Runnable - a private line result; Private TextView showResult; ProgressDialog's private progress; @Override public void onCreate (Bundle savedBundleInstance) - super.onCreate (savedBundleInstance); setContentView (R.layout.main); showResult (TextView) this.findViewById (R.id.main); showResult.setText (Click on any key to start the calculation); - @Override public boolean onKeyDown (int keyCode, KeyEvent keyEvent) - Progress Flow Flow - a new thread (it); thread.start(); Super.onKeyDown (keyCode, keyEvent); - Public invalid launch - myClass.calculate (800).toString (); handler.sendEmptyMessage (0); - Private handler - new handler - @Override public invalid penMessage (Message msg) - progress.dismiss(); showResult.setText (result); Running Stream can be called thus too Javafinal Runnable runner - the new Runnable ()) public void launch () - tv.append (Hello World); handler.postDelayed handler.postDelayed (runner, 1000); Otherwise, we can use the original runable thread that we use in the java JavaThread thread ()) @Override a public invalid run () - try while (truly) - sleep (1000); handler.post (runner); thread.start(); Have a good day. Other recommended messages on android,

handler, runnable, stream public interface Runnable runnable interface should be implemented by any class, copies of which are designed to run the stream. The class should define a method without arguments called running. This interface is designed to provide a general protocol for objects that want to run code while they are active. For example, Runnable is implemented by a class thread. Activity simply means that the thread is up and running and hasn't stopped yet. In addition, Runnable provides the class with the tools to be active, not to subclassize the flow. Runnable Reactable Class Can Work Without Subclassifying Flow instance of the flow and passing itself as a target. In most cases, the Runnable interface should be used if you only plan to override the startup method and no other flow methods. This is important because classes should not be subclassified if the programmer does not intend to change or enhance the fundamental behavior of the class. When the object implementing the Runnable interface is used to create a thread, the flow starts triggering an object start method that will be called in that individual thread execution. Public Methods for Public Abstract Void Start () When the interface of the object implementing the Runnable interface is used to create a thread, the flow starts to cause the object to start in that separate thread. The general contract to start the method is that it can take any action. In this example, we'll look at the Runnable interface in Java and how it can be used in connection with the Flow class to create and run new threads in your program. We'll also try to explain what Runnable is and what's different from the thread, and look at the different ways runnable is implemented in the code.1 Runnables and ThreadsLet start by talking about very common java confusion. Runnables don't thread. The Runnable interface is used to identify a specific task that we want to accomplish, and the entire implementation of this task is within its only method, the run () (which does not accept any arguments). The Runnable class itself does not create a separate thread because that's what the thread actually does. The thread is used to create a new run path (new thread) separate from the main program. By providing Runnable as an argument to a thread designer, you essentially provide access to the Runnable task (defined in the launch method) to a new thread object. At any time during the program, you can start a new thread using Thread.start and the Runnable code will start working.2. Why use Runnable? While you have the ability to create new threads using only the thread class that implements the Runnable interface itself, the proposed and much more accepted approach is to use Runnable for all the logic we want to provide for our threads. There are certain reasons for this: Modularity: when you run a thread and it finishes working, there is no way to restart it. This can lead to duplication of code in the event of multiple reads, when you need a specific task to run multiple times. Fortunately, you can create a Runnable instance that can be reused in any number of threads. Easy to use: Runnable has only one method, public launching of emptiness. He doesn't take any arguments and how Easier. The thread has many methods that need to be taken into account, making it very cumbersome to work, and additional overheads are always always More often than not, you'll have to use additional classes (through inheritance) to expand the functionality of your Runnable object. Java doesn't support multiple inheritances, so it's much more convenient to create a new class that simply implements Runnable (and allows you to expand a different class) than creating a new class that expands the flow and prevents you from inheriting anything else. Run implementation and example There are two ways to implement Runnable in Java. Let's take a quick look at them: Inheritance: You can create a class that implements the Runnable interface. You'll be forced to implement a start-up method that contains the logic/task code, and instantly use it in a standard Java way. As explained above, you can use this as an argument for the Thread example. Anonymous Internal Class: In some cases, you need to run a small snippet of just a few lines of code. In this case, you can always create a new anonymous internal class inside the flow designer and implement the startup method there as well. It's certainly not modular, and you can't reuse that code. Let's look at the code and see how these methods work! MyRunnableImplementation.java0102030405060708080910111114151617171819202 1package com.javacodegeeks.runnableexample;public class MyRunnableImplementation implements Runnable - @Override public void running () i lt; 5; i') - System.out.println (Thread.currentThread ().getName () - Twith Runnable: MyRunnableImplementation works... As you can see, we're creating a class that implements the Runnable interface. In the main program, we will immediately reshashe this class and move it as an argument to the flow designer, which will run a task that is a loop in this context. RunnableExampleMain.java0102030405060708080910111113111141141516171718117112222222222252525262626272829293031323334353637383940414 2package com.javacodegeeks.runnableexample;public-class RunnableExampleMain - Public Static Void Core (String) - System.out.println (Program Execution...); MyRunnableImplementation r - new MyRunnableImplementation Thread Flow1 - new thread (r, Thread 1); thread1.start(); Stream Flow2 - new thread (r, Thread 2); thread2.start(); Thread thread3 - new thread (new Runnable() - @Override public invalid launch () - for (int i y 0; i zlt; 5; i') - System.out.println (Thread.currentThread().getName() Thread 3); thread3.start(); Please note that we can use the same Runnable on more than one thread, without problems. In addition, we can implement Runnable anonymously inside the constructor.3.1 Java Runnable Example - OutputWe must take into account the fact that we are talking about different threads here that are unpredictable by definition. Depending on the implementation of JVM and the architecture of the running platform, threads can work in any order. In our example, in particular, you can see different output options, because although threads work simultaneously (so when you call Thread.start () a new thread starts running in parallel with the main program), they all have the task of printing something on a standard output, and we don't have a way to know which thread will use the output at any given time. Let's look at two different runs where we can easily see what streams are running and what runnable they use:0102030405060708091011111314151 6Executing program... Thread 1 with Runnable: MyRunnableImplementation works... 0Thread 1 with Runnable: MyRunnableImplementation works... 1Thread 1 with Runnable: MyRunnableImplementation works... 2Thread 1 with Runnable: MyRunnableImplementation works... 3Thread 1 with Runnable: MyRunnableImplementation works... 4Thread 2 with Runnable: MyRunnableImplementation works... 0Thread 2 with Runnable: MyRunnableImplementation works... 1Thread 2 with Runnable: MyRunnableImplementation works... 2Thread 2 with Runnable: MyRunnableImplementation works... 3Thread 2 with Runnable: MyRunnableImplementation works... 4Thread 3 with Runnable: Inner Class Runnable Works... 0Thread 3 with Runnable: Inner Class Runnable Works... 1Thread 3 with Runnable: Inner Class Runnable Works... 2Thread 3 with Runnable: Inner Class Runnable Works... 3Thread 3 with Runnable: Inner Class Runnable Works... 4Pretty standard, the output is something that someone might expect. Note that the first two threads used the same instance of MyRunnableImplementation without problems, while the third used an internal class. However, after running it again for several times, we got this output:0102030405060708091111121415 16Executing program ... Thread 1 with Runnable: MyRunnableImplementation works... 0Thread 1 with Runnable: MyRunnableImplementation works... 1Thread 1 with Runnable: MyRunnableImplementation works... 2Thread 1 with Runnable: MyRunnableImplementation works... 3Thread 1 with Runnable: MyRunnableImplementation works... 4Thread 3 with Runnable: Inner Class Runnable Works... 0Thread 3 with Runnable: Inner Class Runnable Works... 1Thread 3 with Runnable: Inner Class Runnable Works... 2Thread 2 with Runnable: MyRunnableImplementation works... 0Thread 2 with Runnable: It's working... 1Thread 3 with Runnable: Inner Class Runnable Works... 3Thread 3 with Runnable: Inner Class Runnable Works... 4Thread 2 with Runnable: MyRunnableImplementation works... 2Thread 2 with Runnable: MyRunnableImplementation works... 3Thread 2 with Runnable: MyRunnableImplementation works... 4 Prepare cool, right? Well, yes, but threads can sometimes cause a real headache, and in many cases, you need to know the order in which they work. Fortunately, Java contains ways to achieve synchronization and planning that go beyond this tutorial. Download the source code In this example, we've studied the Runnable interface in Java and how it can be used in connection with the Flow class to create and run new threads in your program. This was an example of the Runnable interface in Java.Last updated April 28, 2020 2020