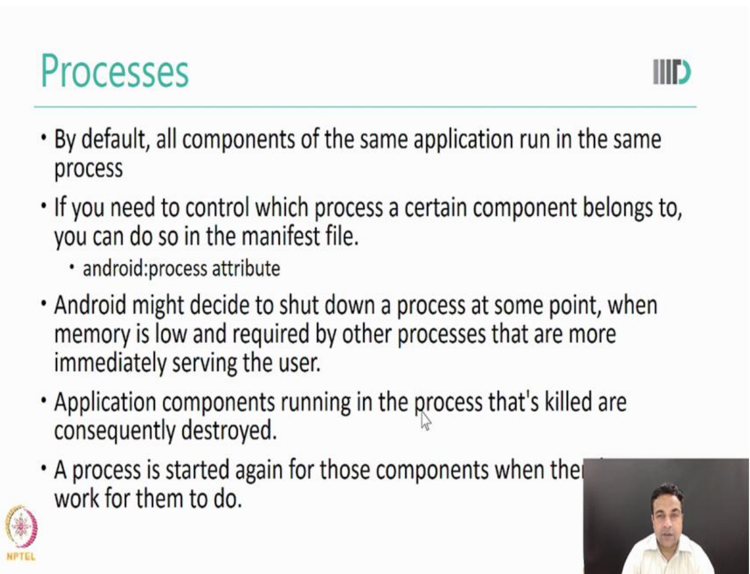
Mobile Computing
Professor Pushpendra Singh
Indraprastha Institute of Information Technology Delhi
Lecture 32
Processes and Threads

Hello, welcome to your new class, today we will go slightly in a different direction and we will be discussing how the Android processes and threads work when we are running an application and how does Android manages activities, services and other things in terms of processes and threads because these are the basic units of execution and we will try to use and understand how does the activities, app component anything in Android application works in terms of processes and threads.

(Refer Slide Time: 1:06)



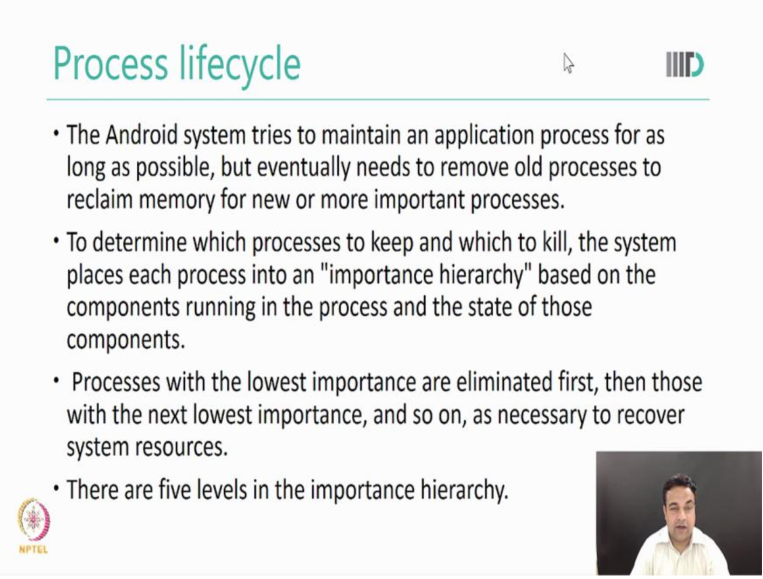
So let us get started, processes now let me remind a little bit from your operating system course. In operating system course we normally study something called a program, a process and a thread; can you recall what these were? Ok. So a program usually is nothing but some lines of code which are written. A program in execution is called a process and a thread is a line of execution. So process is something which is alive while program is not and thread is something which is a line of execution there could be multiple threads in the same process or there could simply be one thread, there will always be one thread but then there would also be multiple threads. So let us start now with our Android operating system.

So in Android by default all components of the same application run in the same process. However, if you need to control that which process a certain component belongs to, you can do so in the manifest file. There is an attribute given in android colon process and that helps

you achieve your way. Now Android might decide to shut down a process at some point of time, when the memory is low or required by other process that are more immediately serving the user.

Now, when Android decides that then any application component that is running in the process is killed as we can understand. And when when there is a work to do, then the process is started again for those components. So because we are working on a mobile platform where the resources are very scarce, the Android system may try to kill some processes and but if those processes are needed then those process will be restarted. For a normal desktop, a Linux does not really have to do this because there are plenty of resources.

(Refer Slide Time: 3:30)

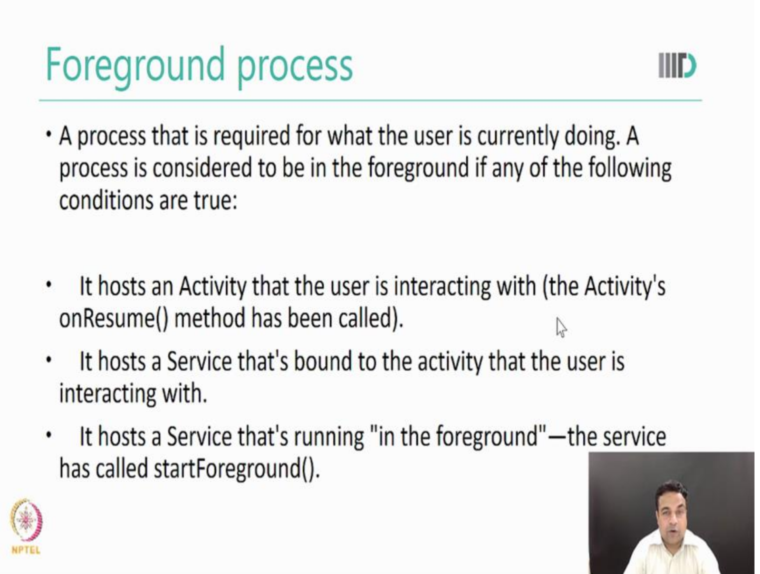


Now let us understand the process lifecycle in Android. So the Android systems try to maintain an application process for as long as possible, but because there could be multiple processes the system needs to eventually remove old processes to reclaim memory for new or more important processes. Now this is a decision which is similar to what for example an operating system usually you may have studied about schedulers and in case of scheduler, the scheduler needs to decide which process to run. And similarly for the memory page faults sometime the process has to decide which page is to throw out. So these are some decisions which are made throughout in the system in different stages.

So for Android, in order to determine that which process to keep and which to kill the system places each process into a kind of importance hierarchy, which is a hierarchal list and because it is hierarchal list the items set the bottom of the list will a more chance to getting killed then

items on the top of the list. So there are total five levels in this important hierarchy in an Android operating system. So let us try to see what those file levels are.

(Refer Slide Time: 5:04)



So the first or the top most level is the foreground process that is the process which is currently in the foreground process that is required for what the user is currently doing. Now this process is obviously the most important process and that is the reason that the foreground process are on the top of the hierarchy and these processes have a very less chance of getting killed. Now what are the conditions in which a process is considered to be in the foreground? So here are some following conditions and in any of these conditions prove we assume that the process is the foreground process. So number one, process hosts an activity that the user is interacting with, so that basically means that there is an activity which is currently active, visible to the user so the process that is hosting that activity is considered a foreground process.

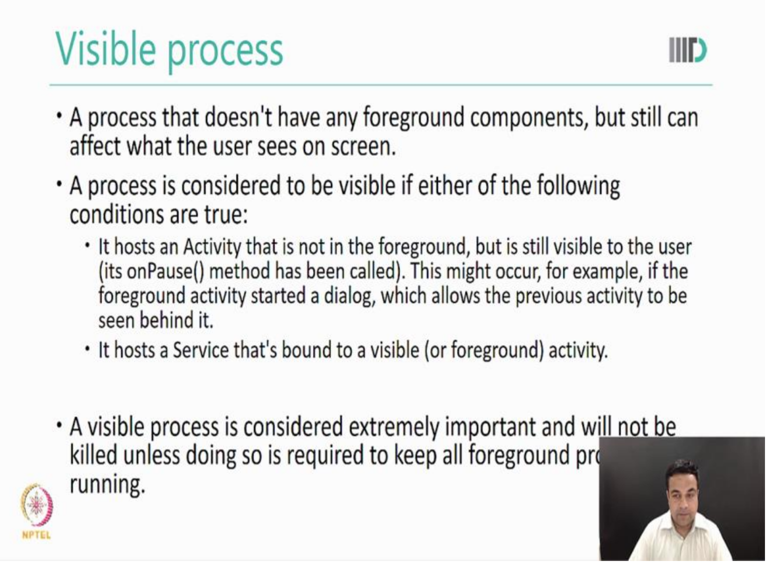
Similarly, the process that is hosting a service that is bound to the activity that the user is interacting with. Third is that it is hosting a service that is running in the foreground, so in last lecture we learned about services which user runs the background, but services which can also run in the foreground. So if there is a process which is hosting a service that is running in the foreground the processes is also considered foreground process.

And then a process that host a service that is executing one of its lifecycle call back so when a service is executing one of its lifecycle call backs at that time the process that is hosting is considered a foreground process and gets the highest priority. And then the process which

holds the broadcast receivers that is executing its own receives method. Now one thing that is common that you may have observed is that the foreground process is directly linked to the current activity that is going on the device.

So whether it is in form of an activity or in form of a service, or in form of a broadcast receiver, etc, etc. So because as you will see in the mobile you can see that there would only be very few foreground process at any given time. And foreground process are only killed as the last resource which means that if the memory is so low that the Android system has to reside within the foreground process then only foreground process will get killed. And if the system does not kill then the user interface will become non responsive. So that is so foreground process is killed only if and only if there is no other resource available to make sure that our Android application remains responsive.

(Refer Slide Time: 8:23)

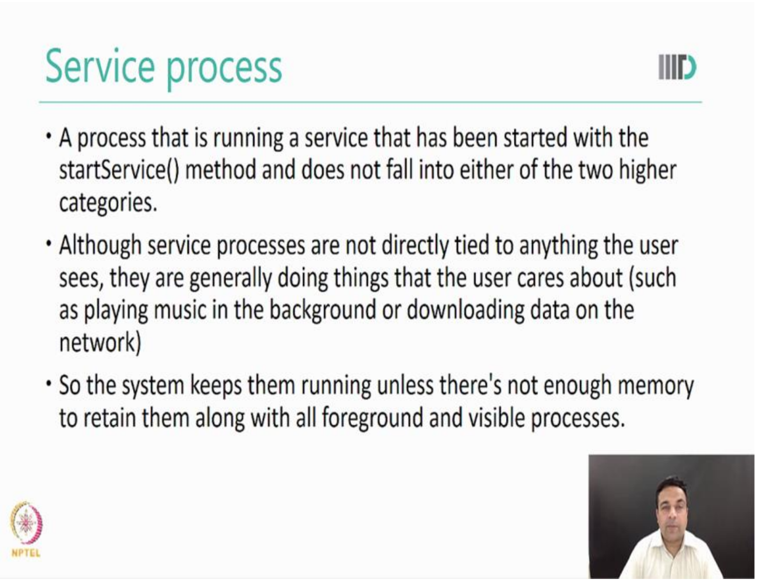


So the after the foreground process second type is of the visible process. So the visible process is a process that does not have any foreground component but still can affect what the user sees on screen. A process is considered to be visible if either of the following conditions are true. For example, if the process hosts an activity that is not in the foreground but is still visible to the user for example, an activity which is called onPause() method. This might occur, for example, if the foreground activity started a dialog which allows the previous activity to be seen behind it.

And the other condition in which a process could be considered as a visible process is that it hosts a service that is bound to a visible activity. So in either of the case we will call that the

process which hosts such an activity or a service is a visible process. So just like the foreground process a visible process is also considered extremely important and will not be killed unless doing so is required to keep all foreground processes running. So visible process priority is very high but not as high as foreground. So it will only be killed if there is no other option then killing it or killing the foreground process. So between a foreground and visible the visible has the lower priority.

(Refer Slide Time: 9:55)



And the third type of process is what is called a service process, a service process is a process that is running a service that has been started with the startService() method and does not fall into either of the two higher categories. The higher categories were the foreground process and the visible process. Now although these services processes are not directly tied to anything the user sees, they are generally doing the things that the user cares about for example, downloading file or playing the music. Now these are some services which if they are running and if they stop then you will not like the app, so that is why the Android does not try to kill a service process either and it tries to keep them running until there is not enough memory for the foreground and visible processes.

(Refer Slide Time: 11:13)

Background process

- A process holding an activity that's not currently visible to the user (the activity's onStop() method has been called).
- These processes have no direct impact on the user experience, and the system can kill them at any time to reclaim memory for a foreground, visible, or service process.
- Usually there are many background processes running, so they are kept in an LRU (least recently used) list to ensure that the process with the activity that was most recently seen by the user is the last to be killed.
- If an activity implements its lifecycle methods correctly, and saves its current state, killing its process will not have a visible effect on the user experience, because when the user navigates back to the activi activity restores all of its visible state.

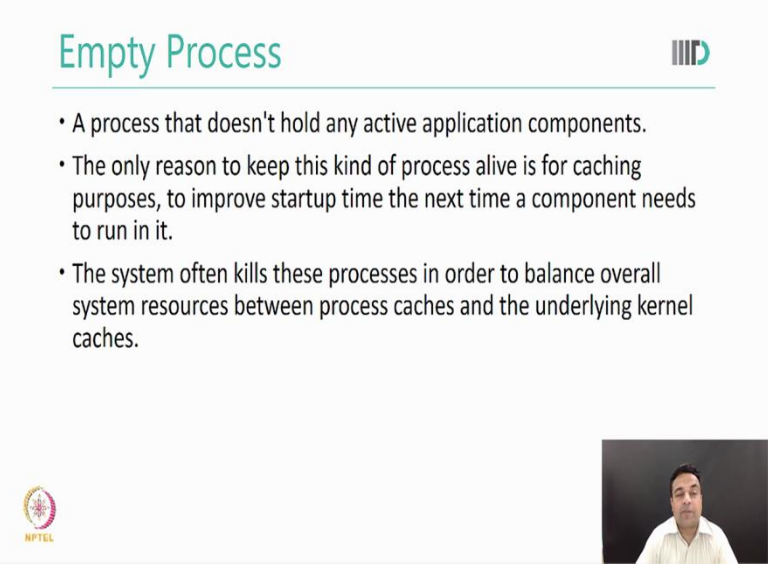
So unless there is such a memory crunch that it has to decide between link service and a visible or a foreground it does not kill the service process either because service process is also going to affect the user experience. The fourth type of process is the background process, so background process is a process that is holding an activity that is not currently visible to the user. For example, an activity use the onStop() method has been called. So if you see when we are going from an activity which was the active visible activity till the activity for which the onStop() method has been called. So these processes have no direct impact on the user experience, the background process and the system can kill them at any time to reclaim memory for a foreground visible or such process.

Now as you can imagine that usually there are many background processes running, so they are kept in an LRU list, LRU list is the (least recently used) list you must have studied in the operating system and a least recently used system a priority list is made depending on what was used this least recently that is the one that has just been used comes on top and the something which has not been used recently comes at the bottom and then the then the processes or pages which comes at the bottom are removed. So similarly if the background process in Android are kept in an LRU list to ensure that the activity that was most recently seen by the user is the last to be killed.

So if the if there is a process associated with an activity which was just seen that process will not be killed compared to the process which was is associated with an activity which was seen some time back, that is how the LRU works. Now if an activity implements its lifecycle methods correctly and saves its current state, killing its process will have not have a visible

effect on the user experience, because when the user navigates back to the activity the activity restores all of its visible state.

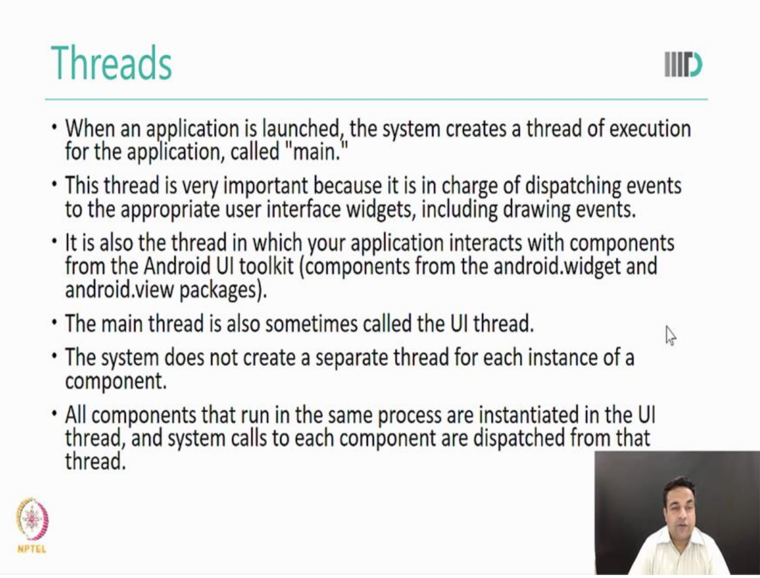
(Refer Slide Time: 13:21)



The fifth type of process is what is called empty process and such a process does not hold any active application component. The only reason for having an empty process is to for caching purposes that is you want to improve start-up time the next time the component needs to run it, so system often kills these processes in order to balance overall system resources between process caches and the underlying kernel caches. So empty process is just an attempt by Android operating system to optimize and if there is any need then these processes are killed.

Now let us come to threads. As you have studied in your course on operating system, threads are nothing but a line of execution. In operating system, we also study what is the difference between process and thread and normally we see that you know what threads do have their own stack and they are variables but they share few things for example, they share heap with the with other threads of the same process. So here let us try to see that what is what are threads in the Android operating system. We just studied about processes now we are going to study about threads.

(Refer Slide Time: 14:56)



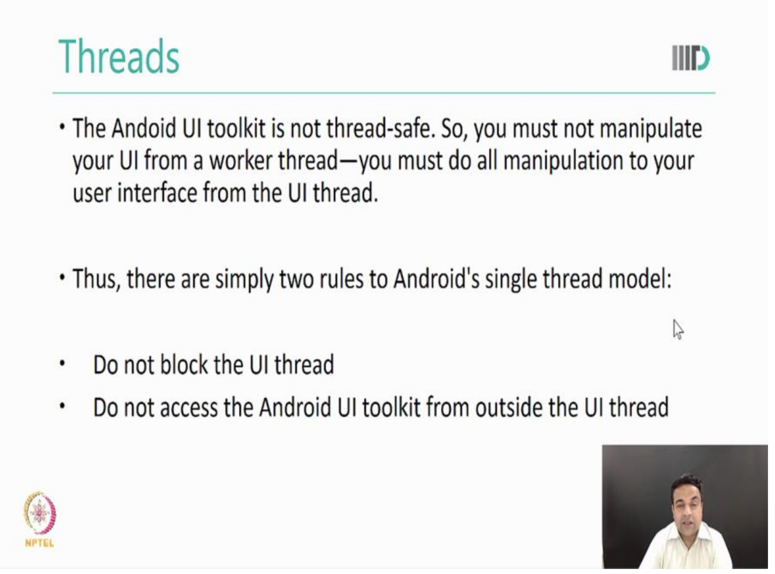
So in an Android when an application is launched, the system creates a thread of execution for the application, which is called main. Now this main thread is very important because it is in charge of dispatching events to the appropriate user interface widgets, including drawing events.

Now it also the thread in which your application interacts with components from the Android UI toolkit. And main thread is also sometimes called the UI thread. So UI thread or main threads are the name of the same thread. The system does not create a separate thread for each instance of a component and so the all components that run in the same process are instantiated in the UI thread or the main thread and the system call to each component are dispatched from that thread. So as you see that there is only one thread which is doing a lot of work which is (()) (15:53) app component interact creating other useful activities. For example, interacting components from Android UI toolkit is in this charge of dispatching events, etc, etc, etc, so this is a thread which is really over loaded.

So what is the affect that happens well when your app performs intensive work in response so let us say user interaction the single thread will lead to your performance because there is only one thread and there is just too much work to do? So if everything is happening in the UI thread, performing long operations such as network access, etc, will block the whole UI. So suppose we want to start a downloading some file or you want to start doing some network access because this is only one thread it will block and when a thread is block no events can be dispatched including drawing events, so for the user it looks like that the application has hang.

Now, the problems down time here, if the UI thread is blocked from more than few seconds you see the infamous dialog says application not responding. So you may have seen this multiple times and that usually happens when a UI thread is too busy to work and then it cannot really refresh the UI for more than 5 seconds and the Android gives us the warning. And if your application gives such a warning then the users may just want to uninstall the apps.

(Refer Slide Time: 17:49)



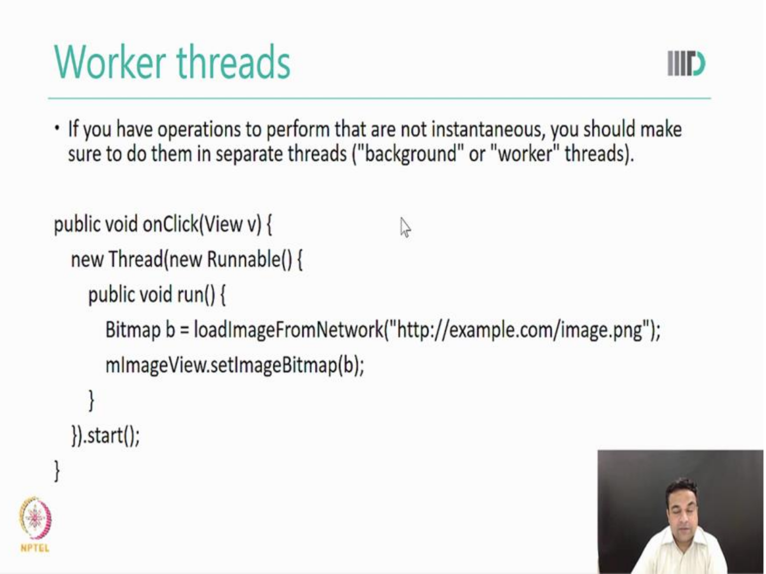
Now another important thing about threads, the Android UI toolkit is not thread-safe. So you must not manipulate your UI from a worker thread, you must do all manipulation to your user interface from the UI thread. The thread-safe if or not threads-safe, you may have studied in the operating system but let me revise it. A thread-safe something for which on which multiple threads can work simultaneously and you are assured that the consistency, etc, will be maintained.

But a non-thread-safe is something where such consistency, etc, is not maintained and it is the responsibility of the developer to do it. You may have already read many synchronization (()) (18:35) algorithms. For example, Petersons algorithms, you may have studies about Semaphores, monitors, etc, so all they work to ensure that you can use multiple thread in such a manner that it looks like that everything is thread-safe.

So if you are little weak in understanding what is tread is please try to go back to your operating system course. So let us come back to Android, so there are simply two rules for Android single thread model. Number one, do not block the UI thread, that is do not do

anything which is sleeping that is it and number two do not access the Android UI toolkit from outside the UI thread because it is not thread save and when it is not thread save you never know what can happen.

(Refer Slide Time: 19:31)



Now let us study a little bit of worker threads, so worker threads are also called background threads and if you have operations to perform that are not instantaneous, for example, let us say downloading, or uploading something, or updating a complex UI and you should make sure to do them in separate threads. So let us see a small example of an onClick() listener in this there is a onClick() listener I start a new thread which loads the image through the network and then sets the image bitmap.

Now, here we are using a separate thread to do this work and not the main thread only so we start our new thread which we call as worker thread here. Now it may look to you that this is perfectly fine because I am using a worker thread. Let us go back one more slide and we said we know do not access the Android UI toolkit from outside the UI thread but this is what exactly what we are doing, so this is not our UI thread and we are manipulating the UI. So let us try to find out a better way of doing it and Android definitely provides better ways.

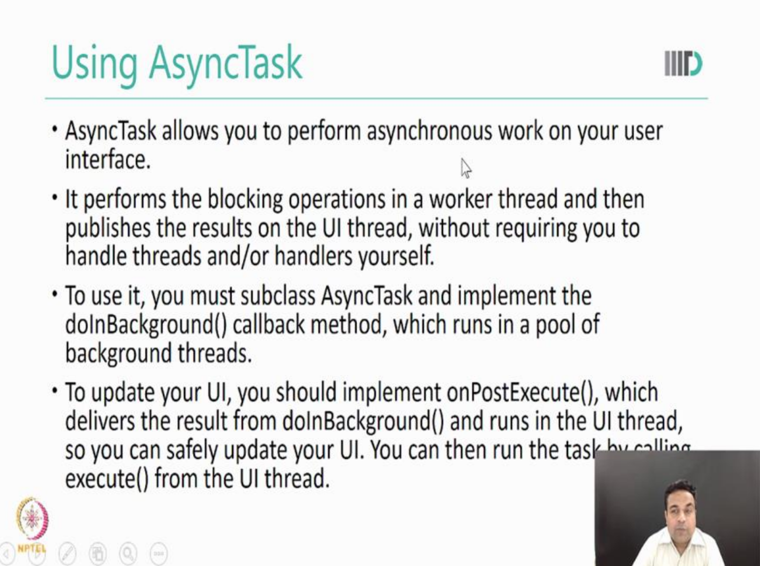
(Refer Slide Time: 21:04)

Worker Thread

```
public void onClick(View v) {
  new Thread(new Runnable() {
    public void run() {
      final Bitmap bitmap =
          loadImageFromNetwork("http://example.com/image.png");
      mImageView.post(new Runnable() {
        public void run() {
          mImageView.setImageBitmap(bitmap);
        }
      });
    }
  }).start();
}
```

So let us look at this example in this we again start a worker thread we do the load image from network but then we are doing something else. We are doing mImageView.post, which we were not doing earlier. So the View.post is something which fixes this problem because the network operation is done from separate thread while the image view is manipulated through the UI thread. Now this is possible because we were able to do using the View.post we were able to do it, however as this (()) (21:45) complex it is not always very intuitive to find out the right method. So, in order to overcome this Android provides us something which is called AsyncTask.

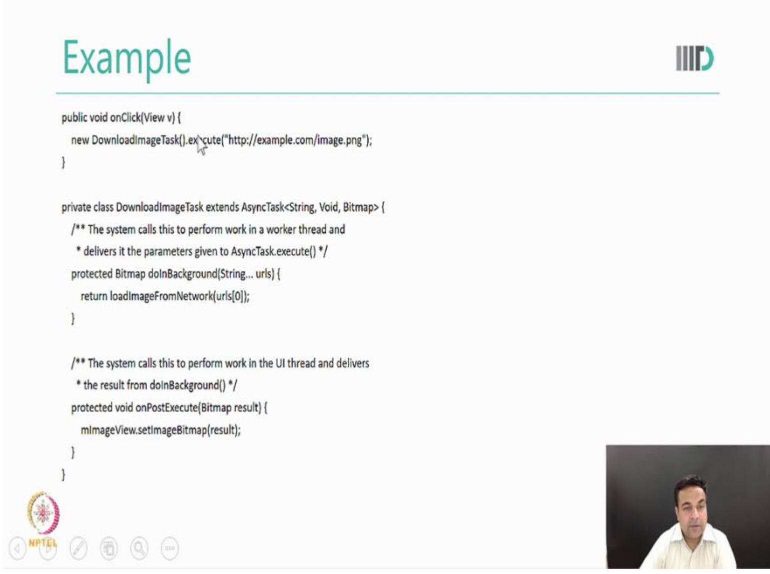
(Refer Slide Time: 22:02)



So the AsyncTask allows you to perform asynchronous work on your user interface. The asynchronous work is something which does not wait for it to finish in simpler words. So AsyncTask performs the blocking operation over the thread and then publishes the results on the UI thread. So basically it is a combination of UI thread and the worker thread when there is a too much of blocking operation that is done on the worker thread and then the results are post to the UI thread.

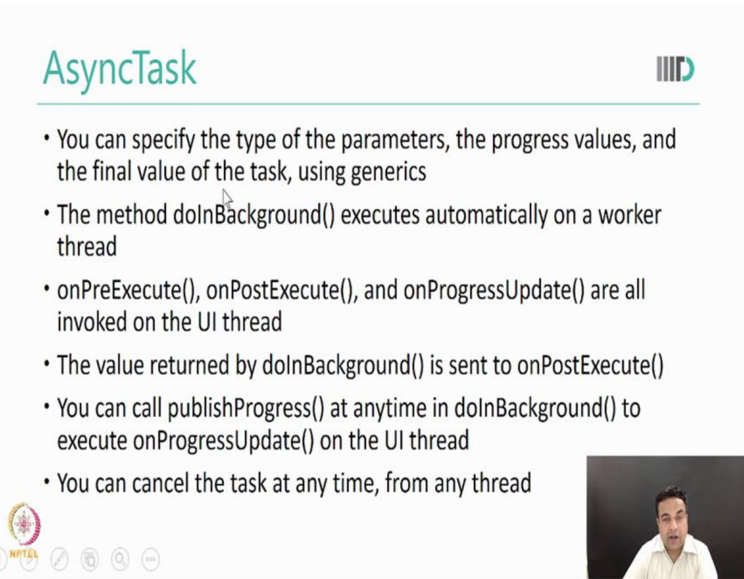
So your conditions for not touching UI toolkit other than the UI thread is fulfilled at the same time it also makes use of the worker thread tool into the performance. How do we use it we must subclass AsyncTask and implement the doInBackground() callback method that is it. And for updating your UI you should implement on POstExecute(), which delivers the result from doInBackground().

(Refer Slide Time: 23:09)



Let us try to see a simple example to understand it, this is the same example that we were doing earlier except that this time we are doing it in the method called doInBackground() as we said earlier and this time we are doing the a setImageBitmap, but because we are doing it in doInBackground() it is fine, so here we are and everything is ok.

(Refer Slide Time: 23:38)



So in AsyncTask you can specify the type of parameters, the progress value, and the final value of the task, using generics. the method doInBackground() will execute automatically on a worker thread and onPreExecute(), onPostExecute() and onProgressUpdate() are all invoked on the UI thread. So the value returned by doInBackground() is sent to onPostExecute() and you can call publish progress at any time doInBackground() to execute onProgressUpdate() on the UI thread.

(Refer Slide Time: 24:22)

So as you see that the AsyncTask is providing you a very good mechanism to move from worker thread to UI thread and UI thread to worker thread. AsyncTask we will cover in detail in later chapters today we are only giving introduction. So this was all about processes and threads in the Android this is to put you in the same context as any other operating system

and the concepts are very much same because Android is (()) (24:35) based on Linux operating system we will in the next few lectures we will be doing a lot of programming to understand all these concepts and discuss them in detail, thank you very much.