Мобилни сервиси со Андроид програмирање (МСАП)

Проф. д-р Перо Латкоски

Вонр. проф д-р Даниел Денковски

Генерални информации за предметот

- Фонд на часови 2+2+1
- Кредити 6
- Консултации: pero@feit.ukim.edu.mk
- Е-курсеви
  - MSAP-3ФЕИТ12Л010
  - Self enrolment key: MSAP2023

Основна литература

Android Developer Fundamentals Course

Learn to develop Android Applications

Concept Reference

Developed by Google Developer Training Team

December 2016

Android Developer Fundamentals Course

Learn to develop Android Applications

Practical Workbook

Developed by Google Developer Training Team

December 2016

[Download Android Studio and SDK tools | Android Developers]

Генерални информации за предметот

- Начин на полагање
  - до прв колоквиум класични предавања
    - Со примери
- Прв колоквиум се полага на компјутер
  - Се креира андроид апликација според барањето на испитното ливче
- После прв колоквиум се работи проектна задача
  - Како работилница (workshop)
  - Во неколку чекори (се дефинираат таргетите)
  - Задолжителен дел за сите
- Наместо втор колоквиум се работи самостоен проект

Генерална тема на предавањето
Background Tasks

AsyncTask & AsyncTaskLoader

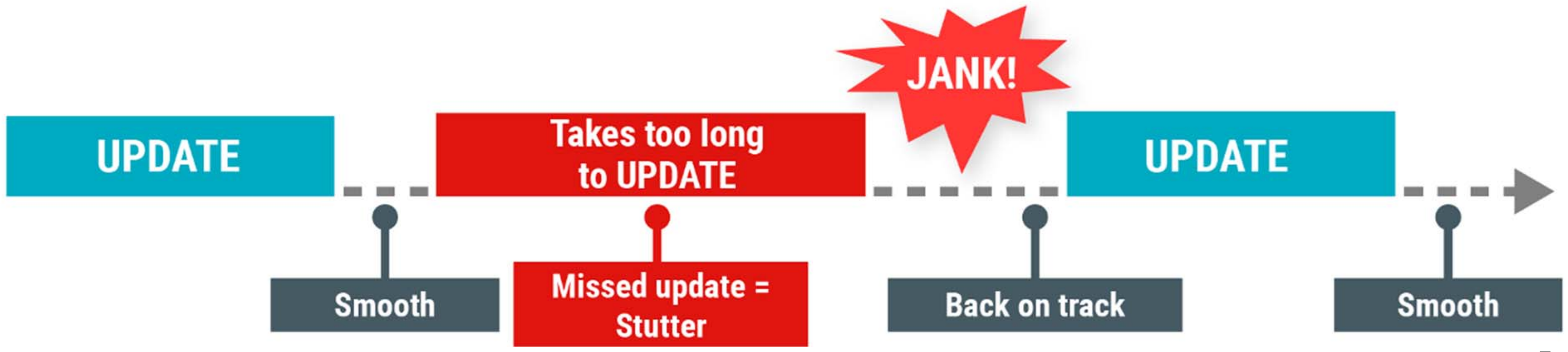
- Threads
- AsyncTask
- Loaders
- AsyncTaskLoader

Threads: The main thread

- Independent path of execution in a running program
- Code is executed line by line
- App runs on Java thread called "main" or "UI thread“
    - Оваа нишка се создава при активирањето на апликацијата
- Draws user interface (UI) on the screen
- Responds to user actions by handling UI events
    - The UI thread dispatches events to the appropriate user interface (UI) widgets,
    - it's where your app interacts with components from the Android UI toolkit (components from the android.widget and android.view packages)

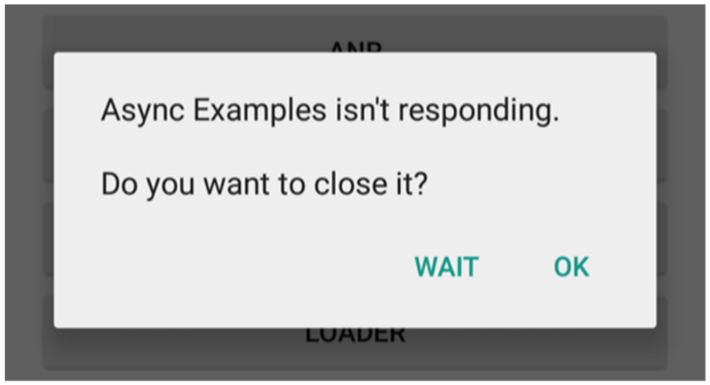
The Main thread must be fast

- The UI thread needs to give its attention to drawing the UI and keeping the app responsive to user input
- Hardware updates screen every 16 milliseconds (60 Hz)
- UI thread has 16 ms to do all its work
- If it takes too long, app stutters or hangs (user experience - UX)



Users uninstall unresponsive apps

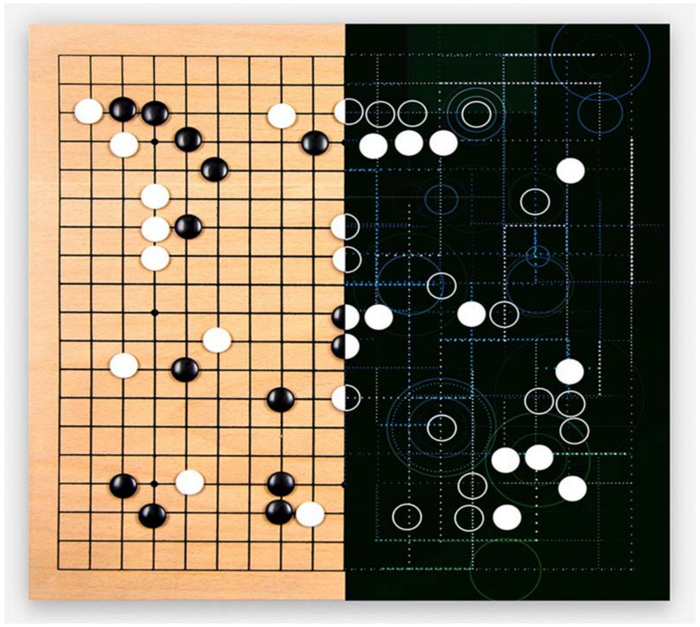
- If the UI waits too long for an operation to finish, it becomes unresponsive
- The framework shows an Application Not Responding (ANR) dialog if the UI thread were blocked for more than a few seconds (about 5 seconds currently)


Async Examples isn't responding.

Do you want to close it?

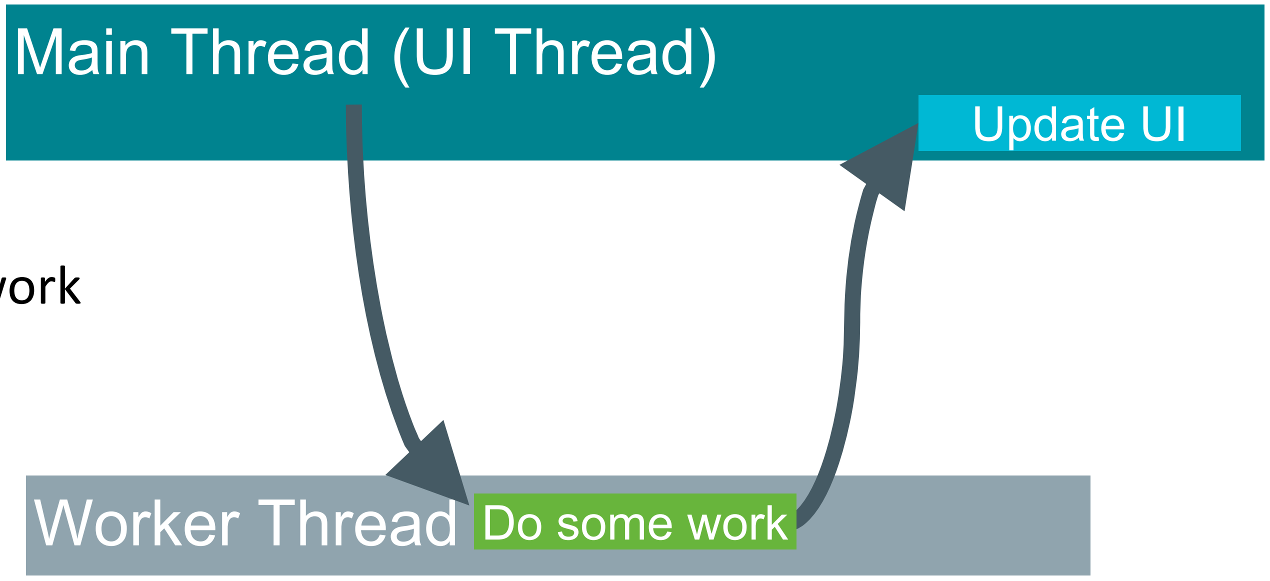
What is a long running task?

- Network operations
- Long calculations
- Downloading/uploading files
- Processing images
- playing media
- Loading data (database queries)
- computing complex analytics
- Machine learning or AI algorithms



Background threads

- Execute long running tasks on a background thread



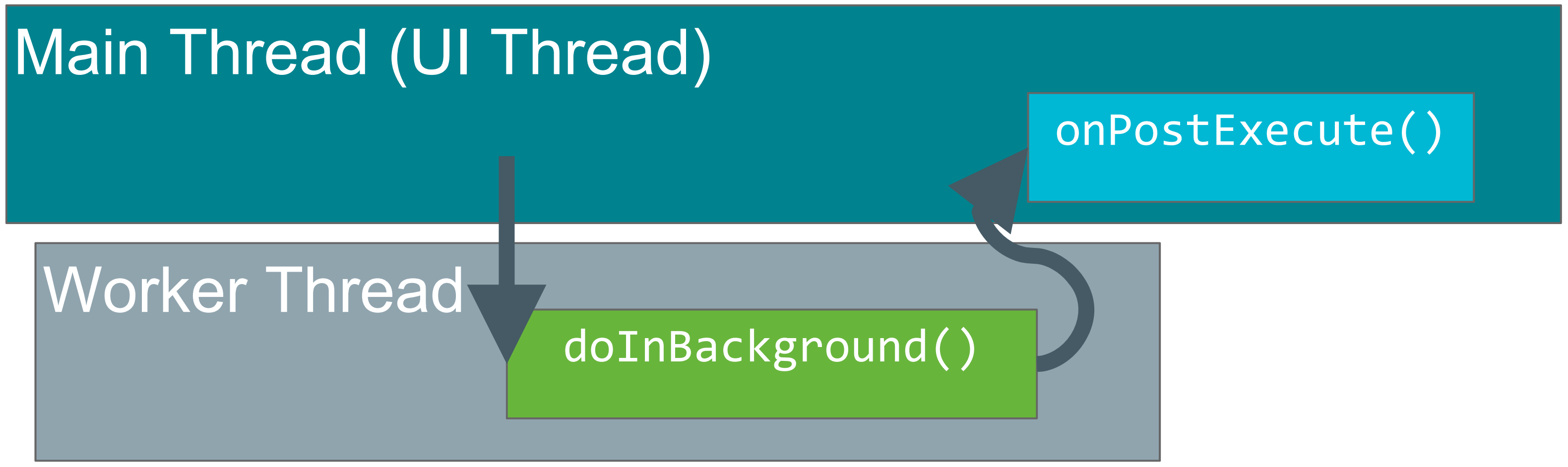
- AsyncTask
- The Loader Framework
- Services

Two rules for Android threads

- Do not block the UI thread
  - Complete all work in less than 16 ms for each screen
  - Run slow non-UI work on a non-UI thread
    - Implement long tasks on a background thread using AsyncTask (for short or interruptible tasks) or AsyncTaskLoader (for tasks that are high-priority, or tasks that need to report back to the user or UI).
- Do not access the Android UI toolkit from outside the UI thread
  - because the Android UI toolkit is not thread-safe
  - Do UI work only on the main thread

What is AsyncTask?

- Use AsyncTask to implement basic background tasks (asynchronous, long-running task on a worker thread)
  - A *worker thread* is any thread which is not the main or UI thread.
- AsyncTask allows you to perform background operations and publish results on the UI thread without manipulating threads or handlers



Override two methods

- doInBackground()—runs on a background thread
  - All the work to happen in the background
  - Performs a background computation, returns a result, and passes the result to onPostExecute().
- onPostExecute()—runs on the main thread when work done
  - Process results
  - Publish results to the UI

AsyncTask helper methods

- onPreExecute()
  - Runs on the main thread before the task is executed.
  - This step is normally used to set up the task, for instance by showing a progress bar in the UI.
  - doInBackground(Params...) is invoked on the background thread immediately after onPreExecute() finishes

- onProgressUpdate()
  - Runs on the main (UI) thread after publishProgress(Progress...) is invoked.
  - Receives calls from publishProgress() from background thread
  - Use onProgressUpdate() to report any form of progress to the UI thread while the background computation is executing.
    - For instance, you can use it to pass the data to animate a progress bar or show logs in a text field.

Diagram of AsyncTask methods calling order

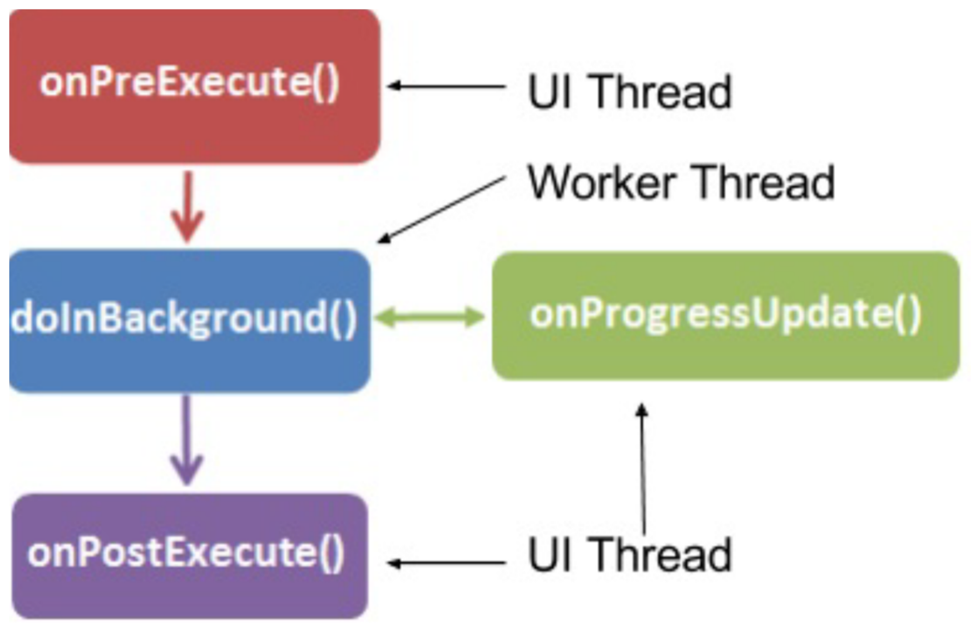
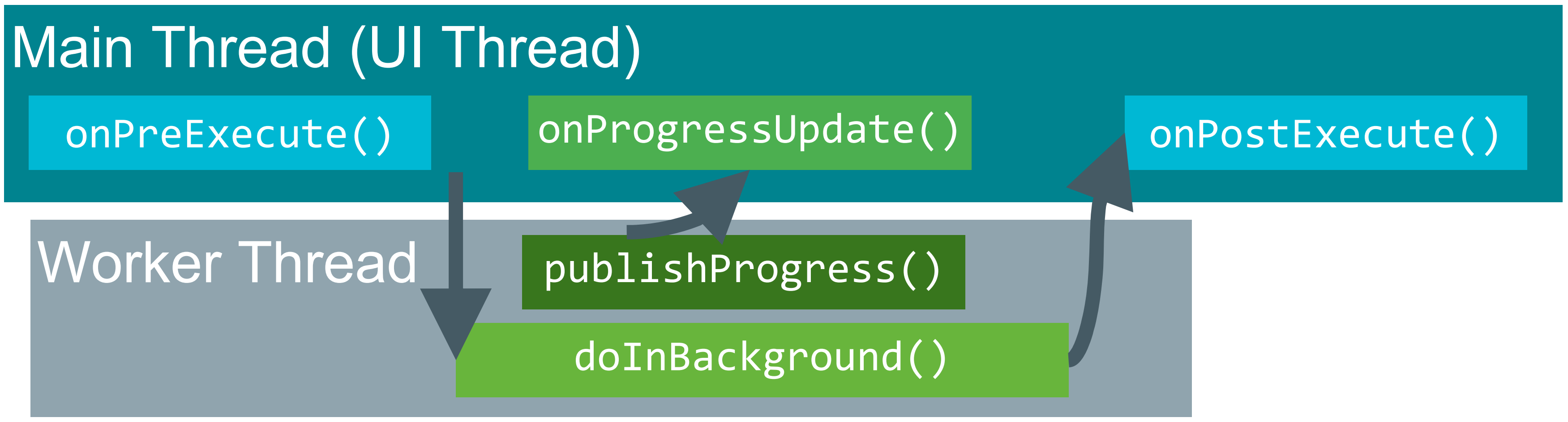


Diagram of AsyncTask methods calling order



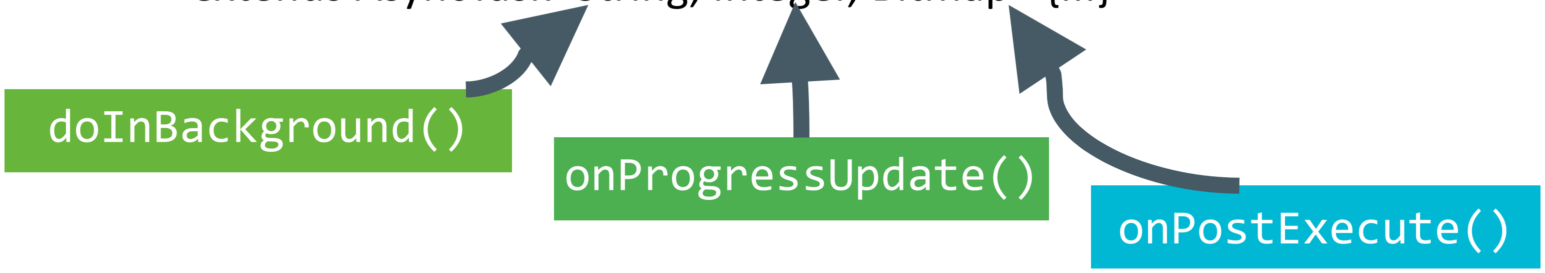
Creating an AsyncTask

- To use the AsyncTask class (abstract java class), define a subclass of AsyncTask that overrides the doInBackground(Params...) method and usually the onPostExecute(Result) method as well.
- Provide data type sent to doInBackground(Params...)
    - "Params" specifies the type of parameters passed to doInBackground() as an array.
- Provide data type of progress units for onProgressUpdate(Progress...)
    - "Progress" specifies the type of parameters passed to publishProgress() on the background thread. These parameters are then passed to the onProgressUpdate() method on the main thread.
- Provide data type of result for onPostExecute(Result)
    - "Result" specifies the type of parameter that doInBackground() returns. This parameter is automatically passed to onPostExecute() on the main thread.

private class MyAsyncTask extends AsyncTask<URL, Integer, Bitmap> {...}

MyAsyncTask class definition

private class MyAsyncTask
   extends AsyncTask<String, Integer, Bitmap> {...}



- String—could be a query, URI (Uniform Resource Identifier) for filename
  - means that MyAsyncTask takes one or more strings as parameters in doInBackground()
- Integer—percentage completed, steps done
- Bitmap—an image to be displayed
  - MyAsyncTask returns a Bitmap in doInbackground() , which is passed into onPostExecute()
- Use Void if no data passed

onPreExecute()

```
protected void onPreExecute() {
    // display a progress bar
    // show a toast
}
```

doInBackground()

```
protected Bitmap doInBackground(String... query) {
   // Get the bitmap
   return bitmap;
}
```

onProgressUpdate()

```
protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }
```

onPostExecute()

```
protected void onPostExecute(Bitmap result) {
  // Do something with the bitmap
}
```

Start background work (Executing an AsyncTask)

- After you define a subclass of AsyncTask , instantiate it on the UI thread.
  - Then call execute() on the instance, passing in any number of parameters. (These parameters correspond to the "Params" parameter type discussed above).

```
public void loadImage (View view) {
   String query = mEditText.getText().toString();
   new MyAsyncTask(query).execute();
  }

//in this case, the AsyncTask has a Constructor that accepts the query
```

Cancelling an AsyncTask

- You can cancel a task at any time, from any thread, by invoking the cancel() method.
- to cancel the task, you call the cancel(true) method on the instance of the AsyncTask.
- The true parameter tells the task to interrupt the thread that's running it.
- You can also pass false to the cancel() method to request that the task be cancelled without interrupting the thread.
    - However, this will only work if the task is regularly checking the isCancelled() flag and stopping its execution when it's set to true.

Cancelling an AsyncTask

- The cancel() method returns false if the task could not be cancelled, typically because it has already completed normally. Otherwise, cancel() returns true.
- To find out whether a task has been cancelled, check the return value of isCancelled() periodically from doInBackground(Object[]) , for example from inside a loop. The isCancelled() method returns true if the task was cancelled before it completed normally.
- After an AsyncTask task is cancelled, onPostExecute() will not be invoked after doInBackground() returns. Instead, onCancelled(Object) is invoked. The default implementation of onCancelled(Object) simply invokes onCancelled() and ignores the result.

пример

```java
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
     protected Long doInBackground(URL... urls) {
         int count = urls.length;
         long totalSize = 0;
         for (int i = 0; i < count; i++) {
             totalSize += Downloader.downloadFile(urls[i]);
             publishProgress((int) ((i / (float) count) * 100));
             // Escape early if cancel() is called
             if (isCancelled()) break;
         }
         return totalSize;
     }

     protected void onProgressUpdate(Integer... progress) {
         setProgressPercent(progress[0]);
     }

     protected void onPostExecute(Long result) {
         showDialog("Downloaded " + result + " bytes");
     }
 }
```

```java
new DownloadFilesTask().execute(url1, url2, url3);
```

пример

- The example above goes through three of the four basic AsyncTask steps:
  - doInBackground() downloads content, a long-running task. It computes the percentage of files downloaded from the index of the loop and passes it to publishProgress() . The check for isCancelled() inside the for loop ensures that if the task has been cancelled, the system does not wait for the loop to complete.
  - onProgressUpdate() updates the percent progress. It is called every time the publishProgress() method is called inside doInBackground() , which updates the percent progress.
  - doInBackground() computes the total number of bytes downloaded and returns it. onPostExecute() receives the returned result and displays it in a dialog.
- The parameter types used in this example are:
  - URL for the "Params" parameter type. The URL type means you can pass any number of URLs into the call, and the URLs are automatically passed into the doInBackground() method as an array.
  - Integer for the "Progress" parameter type.
  - Long for the "Result" parameter type.

Limitations of AsyncTask

- When device configuration changes, Activity is destroyed
  - When device configuration changes while an AsyncTask is running, for example if the user changes the screen orientation, the activity that created the AsyncTask is destroyed and re-created. AsyncTask cannot connect to Activity anymore
- New AsyncTask created for every config change
  - The AsyncTask is unable to access the newly created activity, and the results of the AsyncTask aren't published.
- Old AsyncTasks stay around
- App may run out of memory or crash
  - If the activity that created the AsyncTask is destroyed, the AsyncTask is not destroyed along with it. For example, if your user exits the application after the AsyncTask has started, the AsyncTask keeps using resources unless you call cancel() .

When to use AsyncTask

- Short or interruptible tasks
- Tasks that do not need to report back to UI or user
- Lower priority tasks that can be left unfinished

- For all other situations, use AsyncTaskLoader , which is part of the Loader framework.