ЛЕКЦИЯ I

Android: Background Work

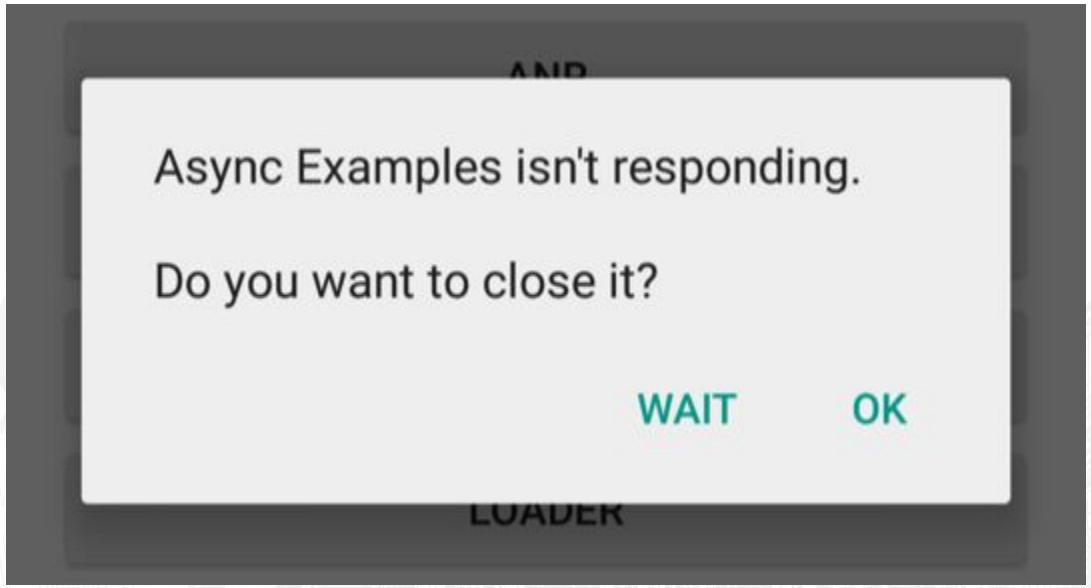
Теодор Костадинов

Съдържание

BACKGROUND WORK

В едно Android приложение:

- Имаме една основна нишка, която се нарича **MainThread/UI Thread**
- На тази нишка се изпълняват процесите, свързани с **живота** на нашето приложение, **рисуването** на екрана, засичането на **кликане**, **анимации**, преминаването от един екран на друг
- **Ако я натоварим с тежки операции, тя ще се блокира**, докато те не завършат, което значи, че приложението временно ще блокира/екранът ще замръзне

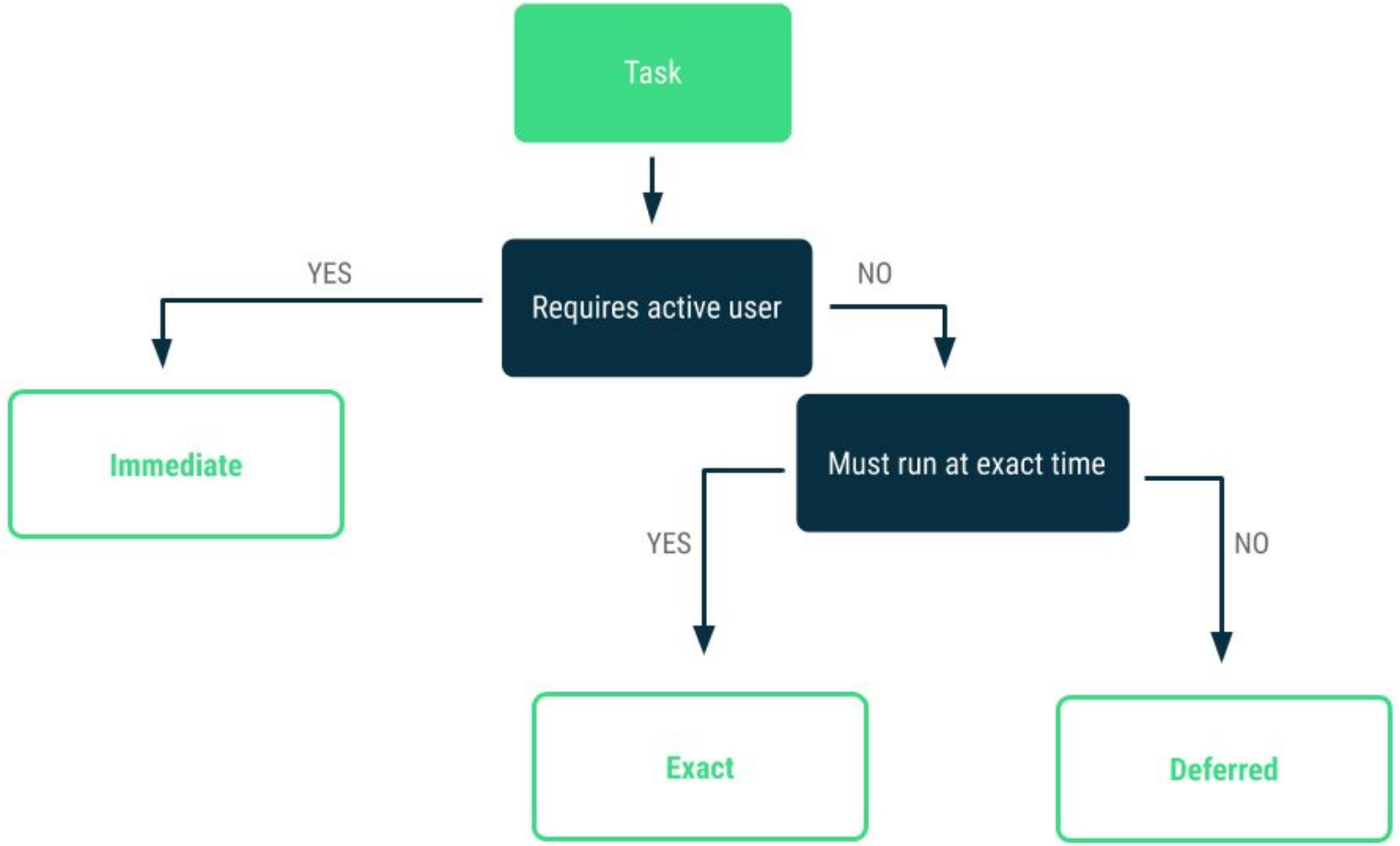


Когато нишката е блокирана, никакви събития не могат да бъдат изпращани, включително чертаенето по екрана. От потребителска гледна точка, приложението е блокирано и по никакъв начин не отговаря. Ако **UI нишката е блокирана за повече от 5 секунди**, се показва екаранът “**application not responding**”.

Бекграунд операции

- Във всички случаи при които операция ще отнеме повече от няколко милисекунди, трябва да се мисли за вариант тя да се изнесе в бекграунда.
- Примерни случаи са декодиране на снимка, достъп до базата или до хранилището на телефона, мрежови операции и други.

Видове бекграунд операции:



IMMEDIATE TASKS

Нишки

- Служат за **изпълнение на код, който няма да окаже влияние върху UI частта на приложението**
- Нишките използват т.нар. Handler-и, които се грижат за показването на екрана
- Ако използваме Java нишки, трябва сами да се погрижим за:
  - обработката на информацията и правилното показване на главната нишка
  - прекъсването на нишката
  - изграждането на thread pool

AsyncTasks

- Използват се за по-удобното и правилно изпълнение на тежки операции, които ще трябва да се покажат на екрана
- **Wrapper около Thread и Handler** - не се налага да ги манипулираме директно
- За **кратки операции** - в най-лошия случай за такива, отнемащи няколко секунди

```
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");
    }
}
```

Params

Progress

Result

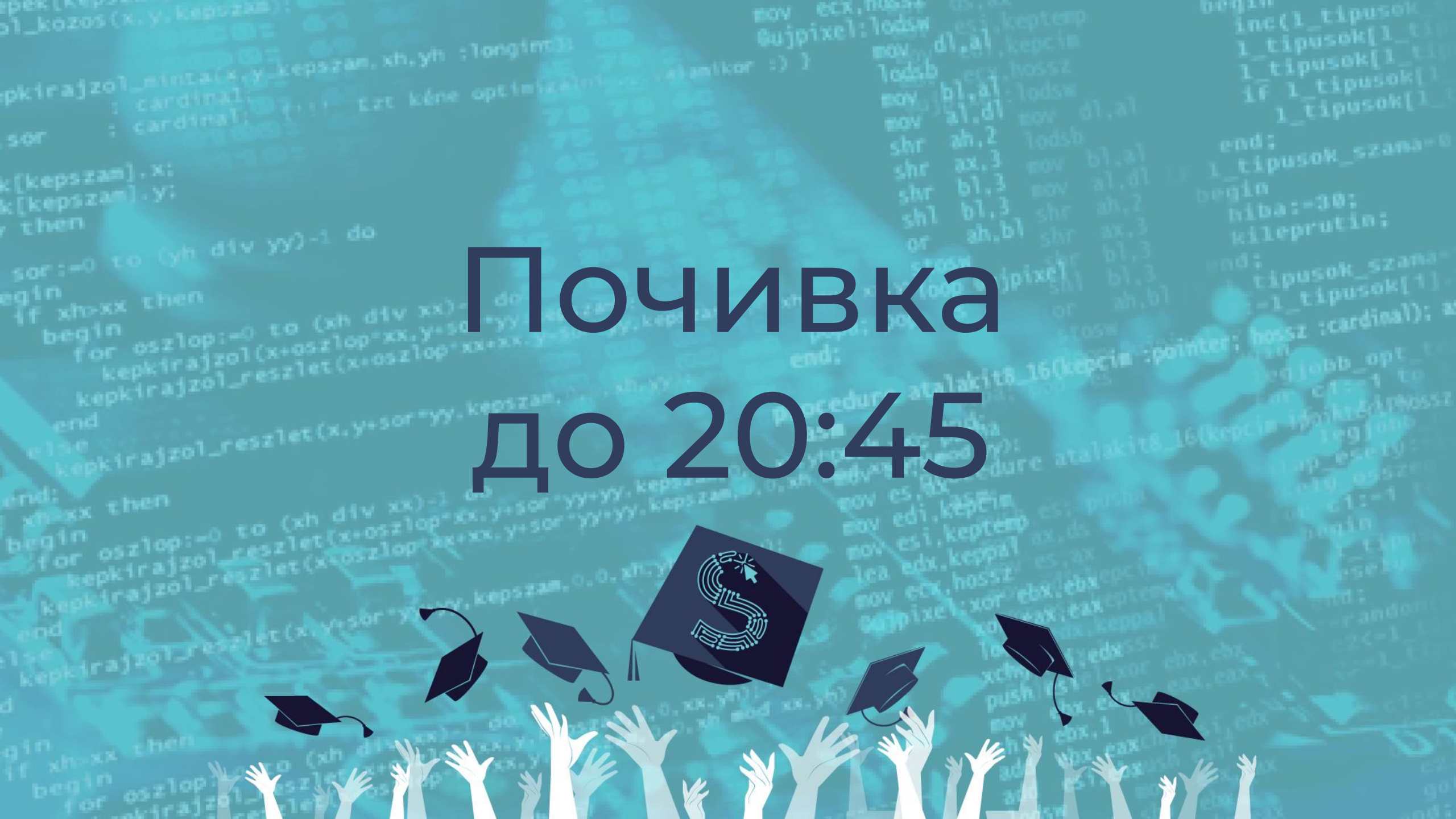
Каква задача искаме да бъде свършена като background work

Какво да се случи след като задачата е приключила

Веднъж създадена, задача може да бъде извикана така:

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

Почивка
до 20:45



AsyncTask's generic types

```
AsyncTask<Params, Progress, Result>
```

Params

Типът на параметрите, които се изпращат към task-а преди неговото започване. Входни данни, които ще бъдат обработвани/използвани.

Progress

Типът на unit-ите, с които ще се измерва прогреса на задачата, докато тя се изпълнява в background-а.

Result

Типът на резултата, които ще се получи, след като завърши изпълнението на задачата, която е пусната като AsyncTask.

Ако искаме да укажем, че даден тип няма да се използва, просто го заместваме с Void.

Threads

Създаването на нишки е скъпа операция. За това обикновено те се създават като част от група, още в началото на изпълнение на апп-а и след това се преизползват.

1. Трябва да създадем thread pool - колекция от няколко нишки

```
public class MyApplication extends Application {
    ExecutorService executorService = Executors.newFixedThreadPool(4);
}
```

Създаваме я в Application класа, за да имаме достъп навсякъде до нея.

2. Достъпваме екзекютура, където искаме да извършим нещо в бекгрунда

```
private final Executor executor = getApplication().executorService;
executor.execute(new Runnable() {
            @Override
            public void run() {
                Result response = makeSynchronousLoginRequest(jsonBody);
            }
        });
```

Threads

Кода, който изпълняваме в нишка трябва да е thread-safe.

3. За да публикуваме резултата на Main Thread-а трябва да го върнем чрез колбек

```
interface RepositoryCallback<T> {
    void onComplete(Result result);
}
```

4. Колбека може да се извика в бекграунд треда и да му се подаде резултата.

```
executor.execute(new Runnable() {
            @Override
            public void run() {
                    Result response = makeSynchronousLoginRequest(jsonBody);
                    callback.onComplete(response );
            }
        });
```

ВАЖНО е да се отбележи, че това само по себе си няма да изпълни резултата на UI thread.

Threads

За да изпълним нещо на UI thread-а, трябва да извикаме метода runOnUiThread. А той се достъпва чрез контекста. Т.е. колбека трябва да се имплементира на място, където имаме достъп до контекст.

5. Имплементираме колбека и минаваме от бекграунд тред на мейн тред:

```
makeLoginRequest(jsonBody, new RepositoryCallback<LoginResponse>() {
            @Override
            public void onComplete(Result result) {
               //Тук сме още на БГ тред
                context.runOnUiThread(new Runnable() {
                    void run() {
                        //Тук сме вече на UI Thread-а
                        postResultToView(result);
                    }
                });
            }
        });
```

https://developer.android.com/guide/background/threading

Deferred tasks

Deferred tasks

Всяка операция, която не е директно свързана с потребителя и може да се изпълни когато и да било, се определя като този тип.

Те ще се изпълнят дори апп-а да приключи или устройството се рестартира.

Използва се WorkManager. Той може:

- Да зададе точни условия, при които задачата да се изпълни
- Позволява изпълнение веднъж или по график
- Работи добре с Doze mode
- Няма нужда да мислим за рестартиране на устройството
- Има политики за повторение при провал
- Може да се навържат логически няколко задачи
- Всяка задача може да връща резултат, който да се предаде на следващата

```
def work_version = "2.5.0"
implementation "androidx.work:work-runtime:$work_version"
```

WorkManager

Всяка операция, се дефинира в отделен клас, който наследява Worker.

```
public class UploadWorker extends Worker {
    public UploadWorker(
        @NonNull Context context,
        @NonNull WorkerParameters params) {
        super(context, params);
    }

    @Override
    public Result doWork() {
      // Do the work here--in this case, upload the images.
      uploadImages();

      // Indicate whether the work finished successfully with the Result
      return Result.success();
    }
}
```

WorkRequest

Отделните Worker-и се групират в WorkRequests. След това рикуестите се подават на WorkManager-а, Който да ги изпълни.

```
WorkRequest uploadWorkRequest =
   new OneTimeWorkRequest.Builder(UploadWorker.class)
       .build();

WorkManager
    .getInstance(myContext)
    .enqueue(uploadWorkRequest);

WorkManager.getInstance(...)
    .beginWith(Arrays.asList(workA, workB))
    .then(workC)
    .enqueue();
```

Exact tasks

Exact tasks

Операция, която не е директно свързана с потребителя, но трябва да се изпълни в точно определен момент, се нарича Exact Task.

Те ще се изпълнят дори апп-а да приключи или устройството се рестартира.

Използва се AlarmManager. Този тип операция не се предпочита, защото не може да се оптимизира от системата, точи батерия и може да бутне сървърите на някое приложение ако се използва грешно.

https://developer.android.com/training/scheduling/alarms

Services

Service

Service за разлика от останалите начини за работа в бекграунда е Андроид компонент, подобен на Активити или Фрагмент.

- При него работата се случва на основната UI нишка
- Service-ът няма сам по себе си UI
- Служи за продължителна работа, без пряко участие на потребителя. Един сървис може да е пуснат в продължение на дни/седмици.
- Ако искаме някой таск от сървис да се изпълни на друга нишка, трябва да използваме WorkManager или Threads.
- Примери за сървиси са:
  - Бекграунд сървиса на месинджър, който държи отворен уеб сокет за нови съобщения
  - Бекграунд сървиса на гугъл мапс, който обновява локацията на период от време
  - Сървиса на спотифай, който позволява изпълнение на музика и във фонов режим

https://developer.android.com/guide/components/services

ВЪПРОСИ?

БЛАГОДАРЯ
ЗА ВНИМАНИЕТО!