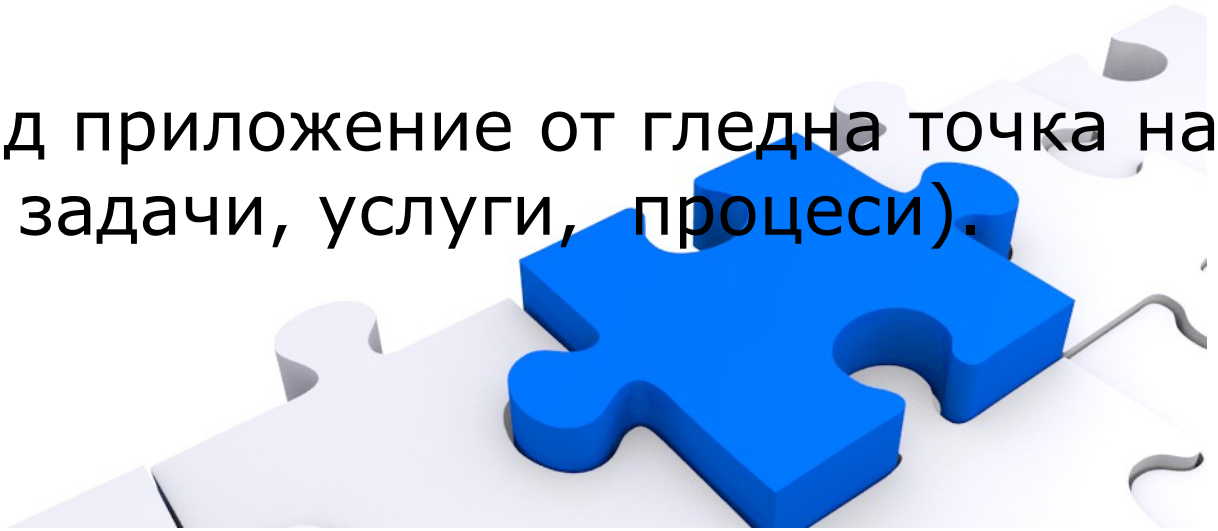
BackProc app (Мрежа + „Процеси“)

- Вместо съдържание:
- Желаем нашето проложение да използва мрежов ресурс?
- Най-лесният начин на за това – simple HTTP GET && POST заявки.
- Каква е „цената“ и как сме „задължени“ да го правим?
- Понятие за андроид приложение от гледна точка на системата (нишки, задачи, услуги, процеси).



Gimme the Net / 1

- За да използваме мрежов ресурс са ни необходими права. Тоест, най-напред – AndroidManifest.xml.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

- Клиент с който да реализираме това? - HttpURLConnection;
- https://developer.android.com/reference/java/net/HttpURLC
- Native за Java => Native за Android (_v1_get()).

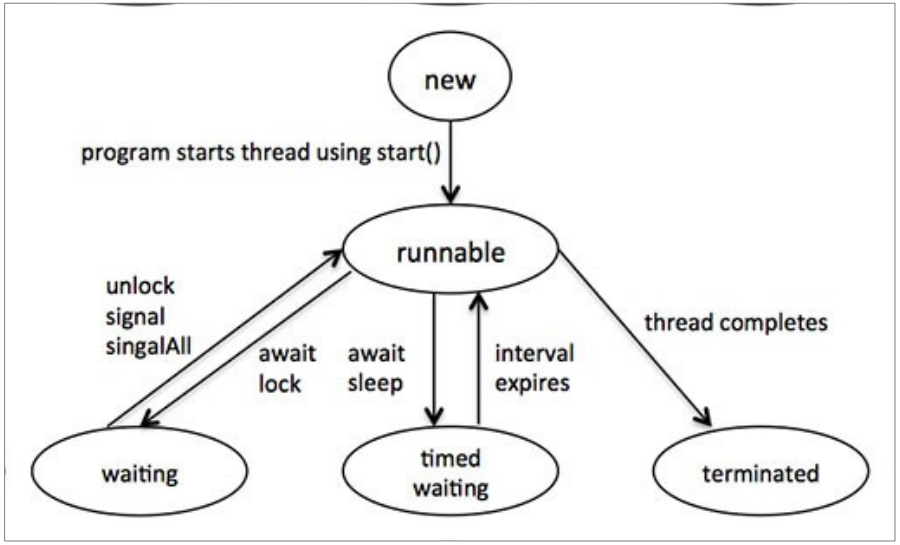
```
URL url = new URL("http://m.yaht.net/repo/f5/");
HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
    try {
        InputStream in = new BufferedInputStream(
            urlConnection.getInputStream()
        );
        readStream(in);
    finally {
        urlConnection.disconnect();
    }}
```

Gimme the Net / 2

- Раглеждаме _v1_get() извикването в _btn_start();
- При условие че сме задали/изискали Internet perms в AndroidManifest.xml;
- Успели сме да конструираме GET заявка (основни познания по HTTP протокола);
- Извикването се извършва в onClick() handler;
- Резултата, от изплънението на заявката е directory-listing-markup;
- Не и според Android обаче. Резултата е:
- **net.yaht.m.backproc V/=-= aHTTPGet(): android.os.NetworkOnMainThreadException**
- Тоест нямаме право на това в Main/UI thread-a;

Gimme the Net / 3

- Тоест имаме нужда от асинхронно изпъление на мрежовата операция в отделен Thread;
- Разчитайки на Java, разполагаме със java.lang.Thread, тоест native Java threads и в Android;



Gimme the Net / 4 – Threads

- Използвайки Threads, бързо и лесно разрешаваме този проблем (коментираният код след извикването на _v1_get());
- Неща над които да помислим, използвайки този toolset;
- Управление на нишките (.start(), .join());
- Runnable класове;
- Нужда от комуникация между тях, т.е. примитиви за синхронизация;
- Нужда от комуникация с UI(Main) thread;

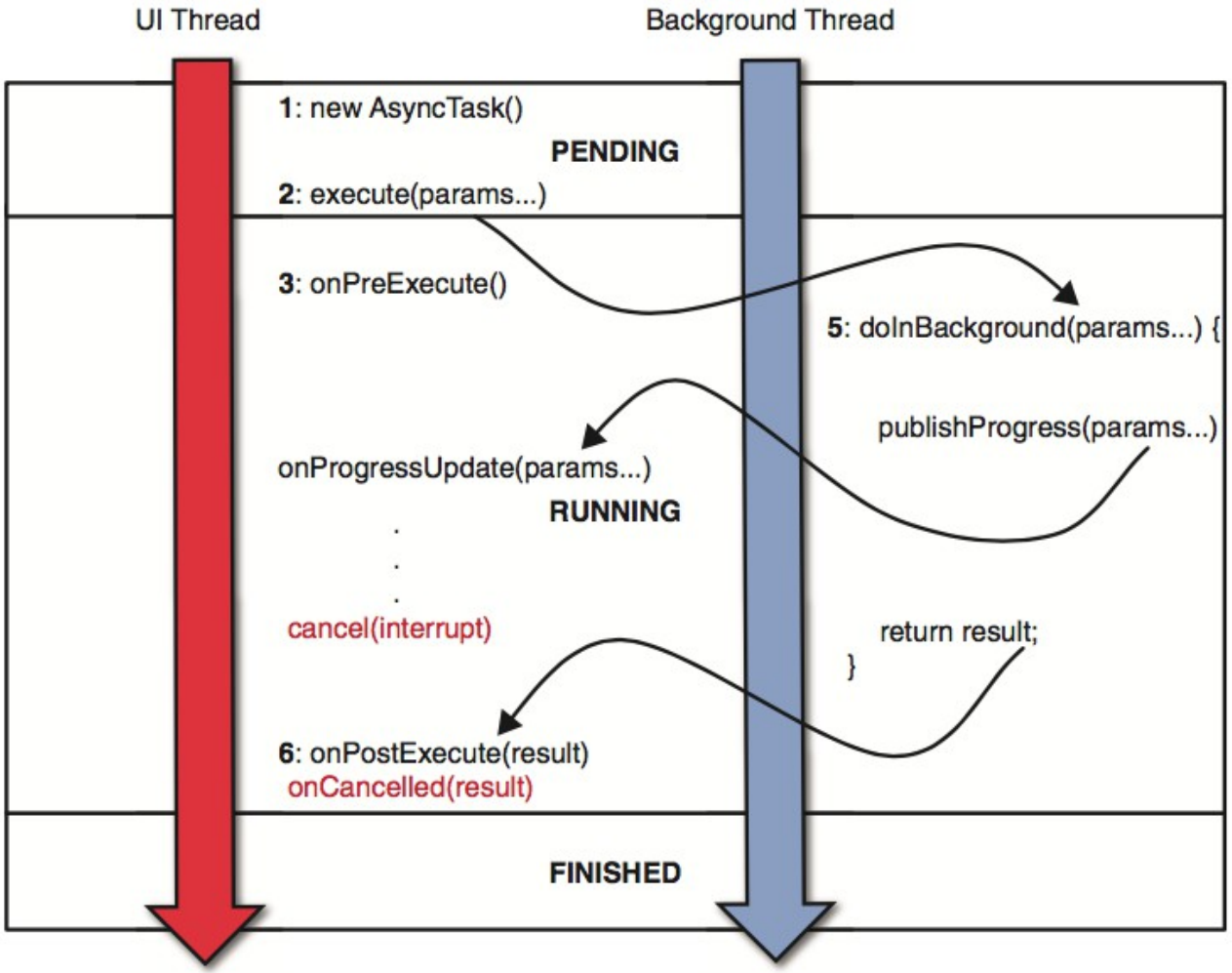
The Android Way – Async Tasks/1

- AsyncTask – първият и най-известен concurrency tool на Android;
- Достатъчно добър за определени типове задачи, но далеч не решаващ всички предизвикателства свързани с асинхронното изпълнение;
- При все това, търсейки за Thread и Android в StackOverflow, присъства като предложение в отговорите на почти всеки въпрос :)
- Основната задача на AsyncTask е да изпълни сегмент от код в background (!= main, foreground) thread;

Async Tasks / 2

- Реализация на класически design.pattern – type-safe template;
- AsyncTask класа е abstract. Единственият начин да го ползваме с subclassing;
- Дефиницията му има единствен abstract метод – doInBackground(<Type>... params);
- За да изпълним код в background, просто наследяваме AsyncTask като реализираме doInBackground(), като в него поставяме background кода;
- **android.os.AsyncTask<Params, Progress, Result>**

Async Tasks / 3 – Execution flow



Async Tasks / 4 - Notes

- Официалната Android документация за AsyncTask:
- https://developer.android.com/reference/android/os/AsyncT
- При все това – a picture is worth a thousand words;
- Използвайки AsyncTask, внимаваме за:
- cancel() логика - изисква се от нас;
- doInBackground(<Type>... params) – винаги трябва да приключва (no unhandled exceptions);
- Винаги се изпълнява в различен Thread; Внимаваме с refs на обекти, предадени към AsyncTask (clones);
- За какво все пак са подходящи – Autonomous Tasks && Cancelable Tasks;