6.1 THREADS

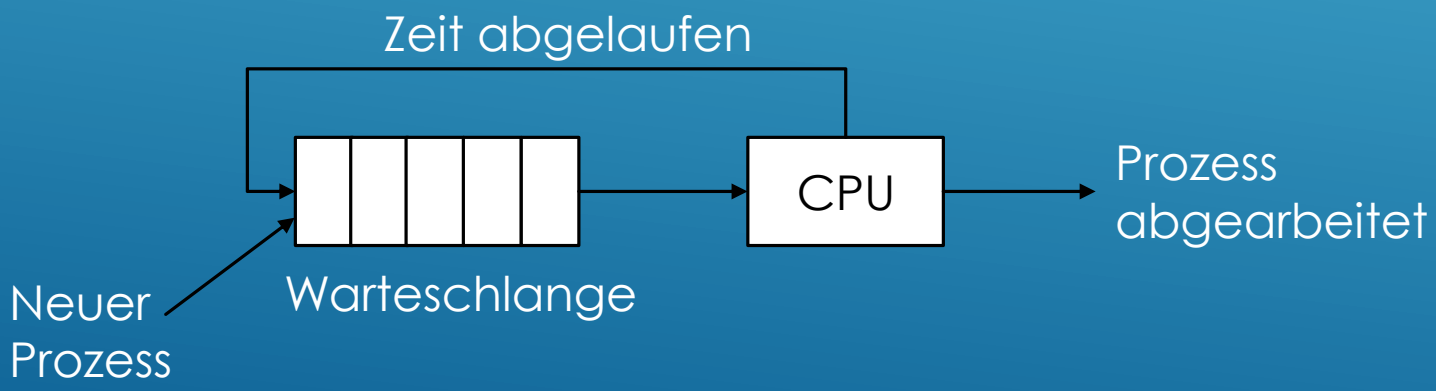
Vortrag von Björn Böing

AGENDA

- Vorwort
- Threads in Java
- Threads in Android
- TL;DR
- Fragen

VORWORT

- Android ist durch den Linuxkernel multitaskingfähig
- Komfortfunktionen erleichtern die Arbeit mit Threads
- Theoretisch pro CPU ein Prozess
- Praktisch „unbegrenzte“ Anzahl an Prozessen, die abwechselnd ausgeführt werden



Quelle: Eigene Darstellung

THREADS IN JAVA 1/3

- Die Basis:
  - Klasse: **java.lang.Thread**
  - Interface: **java.lang.Runnable**

```
Runnable r = new Runnable() {
    @Override
    public void run() {
        // Do something expensive
    }
};

Thread t = new Thread(r);
t.start();
```

- Schlüsselwörter:
  - **Volatile**
    Variable beinhaltet threadübergreifend immer den zuletzt geschriebenen Zustand
  - **Synchronized**
    Methode/Codeabschnitt kann nur von einem Thread gleichzeitig betreten werden

```
public synchronized void oneAtATime(){
    // have some time alone
}

public void comeIn(){

    synchronized (this) {
        // have some time alone
    }
}
```

THREADS IN JAVA 2/3

- Wichtige Methoden:
  - start(): Starten des Threads
  - interrupt(): Unterbrechen des Threads
  - getName()/getId()
  - isAlive(): Überprüfen des Status
  - sleep(int milliSeconds)
  - join(): Warten auf Beenden des Threads
  - yield(): Restzeit aufgeben und neu anstellen

```
volatile boolean mKeepRunning;

public void doSomeMagic() throws InterruptedException
{
    mKeepRunning = true;

    Runnable r = new Runnable() {
        @Override
        public void run() {
            while(mKeepRunning){
                // do some magic
            }
        }
    };

    Thread t = new Thread(r);
    t.start();

    // do something so the thread can run

    mKeepRunning = false;
    t.join();

    // Thread is dead -> Continue doing other stuff
}
```

THREADS IN JAVA 3/3

- Mehrere Threads per:
  - Interface: **java.util.concurrent.ExecutorService**

- **Threads brauchen viel Speicher**
  => Vorsicht! Weniger ist oft mehr!

```
int nThreads = Runtime.getRuntime().availableProcessors();
ExecutorService es = Executors.newFixedThreadPool(nThreads);

/*
ExecutorService es = Executors.newCachedThreadPool();
*/

es.execute(new Runnable() {
    @Override
    public void run() {
        // do some magic
    }
});

es.shutdown();

while(!es.isTerminated()){
    // wait
}
```

THREADS IN ANDROID 1/4

- Main-Thread:
  - Wird vom System beim Start einer App erzeugt
  - Zuständig für Programmlogik **UND** das User-Interface
  - Main-Thread == UI-Thread

- Ist der Main-Thread überlastet, dann **„Application not responding“**

Quelle: http://jdam.cd/async-android/

THREADS IN ANDROID 2/4

```
Runnable task = new Runnable() {
    @Override
    public void run() {
        // do some expensive work

        // show that you're done
        TextView tv = findViewById(R.id.done);
        tv.setText("I am done!");
    }
};

new Thread(task).start();
```

- **CalledFromWrongThreadException**:
  Only the original thread that created a view hierarchy can touch its views
- **=> Logik und UI-Aktualisierungen klar trennen!**

THREADS IN ANDROID 3/4

- **runOnUiThread(Runnable r)**

```
Runnable task = new Runnable() {
    @Override
    public void run() {
        // do some expensive work

        // show that you're done
        Runnable updateUI = new Runnable() {
            @Override
            public void run() {
                TextView tv = findViewById(R.id.done);
                tv.setText("I am done!");
            }
        };

        runOnUiThread(updateUI);
    }
};
```

- **Handler**

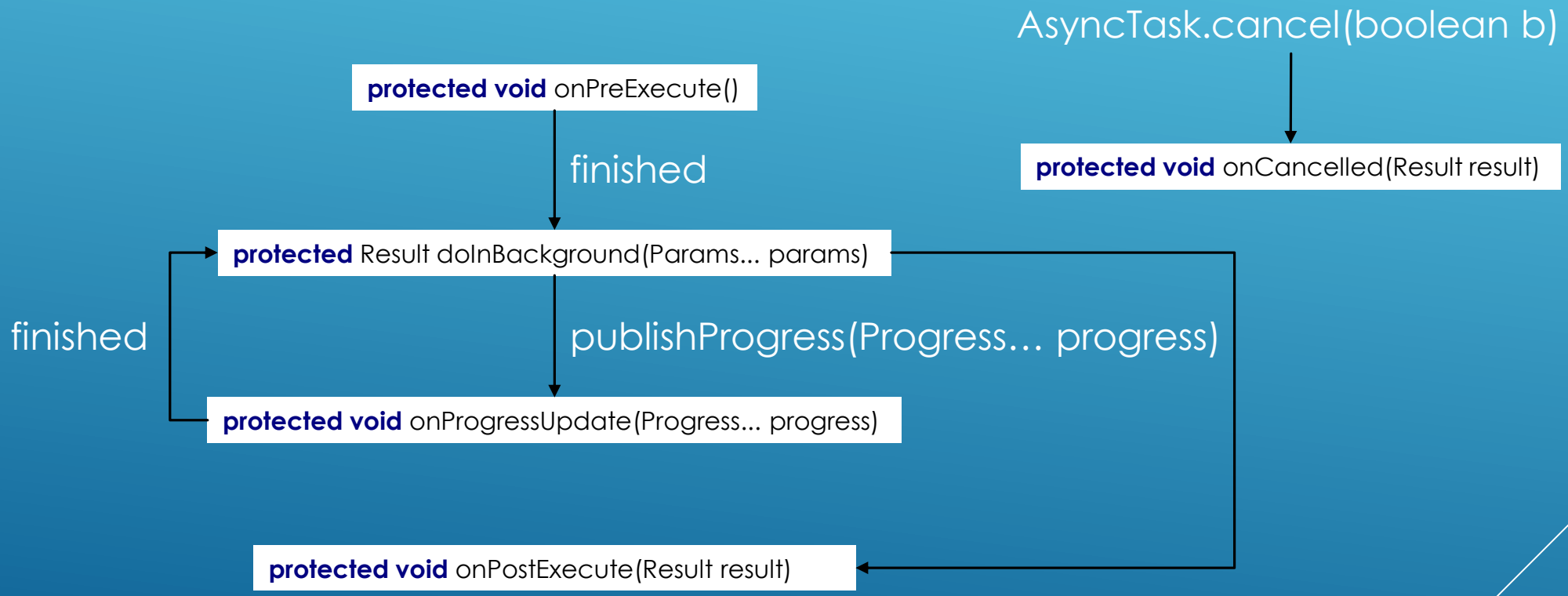
```
Runnable task = new Runnable() {
    @Override
    public void run() {
        // do some expensive work

        Handler refresh = new Handler(Looper.getMainLooper());

        refresh.post(new Runnable() {
            @Override
            public void run() {
                // show that you're done
                TextView tv = findViewById(R.id.done);
                tv.setText("I am done!");
            }
        });
    }
};
```

THREADS IN ANDROID 4/4

- Klasse: **android.os.AsyncTask<Params, Progress, Result>**
- Lifecycle eines AsyncTasks:



BEISPIEL

TL;DR

- MainThread == UI-Thread
- ExecutorService zum Verwalten von mehreren Threads
- UI-Aktualisierung muss im UI-Thread laufen
  => **runOnUiThread** oder **AsyncTask**
- **Weniger ist mehr!**

OFFENE FRAGEN?

QUELLEN

- Künneth, Thomas (2017): Android 7 – Das Praxisbuch für Entwickler
- Android User Guide (abgerufen 12/2017): Thread annotations - https://developer.android.com/studio/write/annotations.html#thread-annotations
- Android Developers (abgerufen 12/2017): Processes and Threads - https://developer.android.com/guide/components/processes-and-threads.html
- Oracle Docs (abgerufen 12/2017): ExecutorService - https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ExecutorService.html
- Oracle Docs (abgerufen 12/2017): Synchronized Methods - https://docs.oracle.com/javase/tutorial/essential/concurrency/syncmeth.html