Java Advanced

Multithreading

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Multithreading

1. Doel
2. Toepassingen
3. Implementatie
4. Thread lifecycle
5. Thread scheduler
6. Synchronisatie

Multithreading

Thread = sub-proces met taak

- 1 taak tegelijk
- “main thread”

Single-threaded applicaties

- Langdurige taak blokkeert main thread

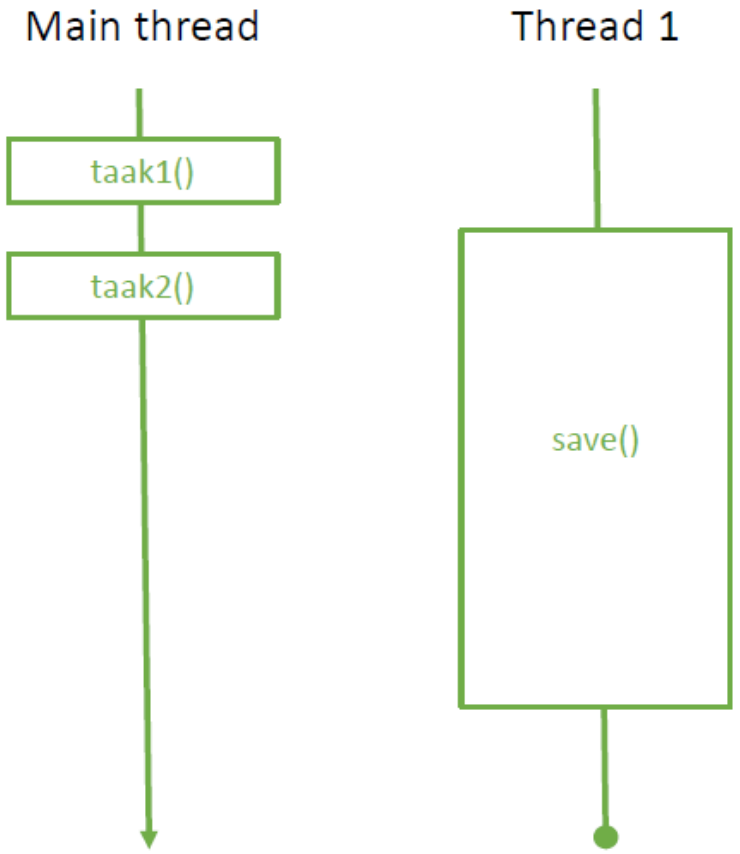
Multi-threaded:

- Parallelle threads voor deeltaken

Multithreading

Multithreading:

- Threads met deeltaak
- Parallel uitgevoerd
- Main thread blijft responsief
- Thread afgesloten na uitvoering

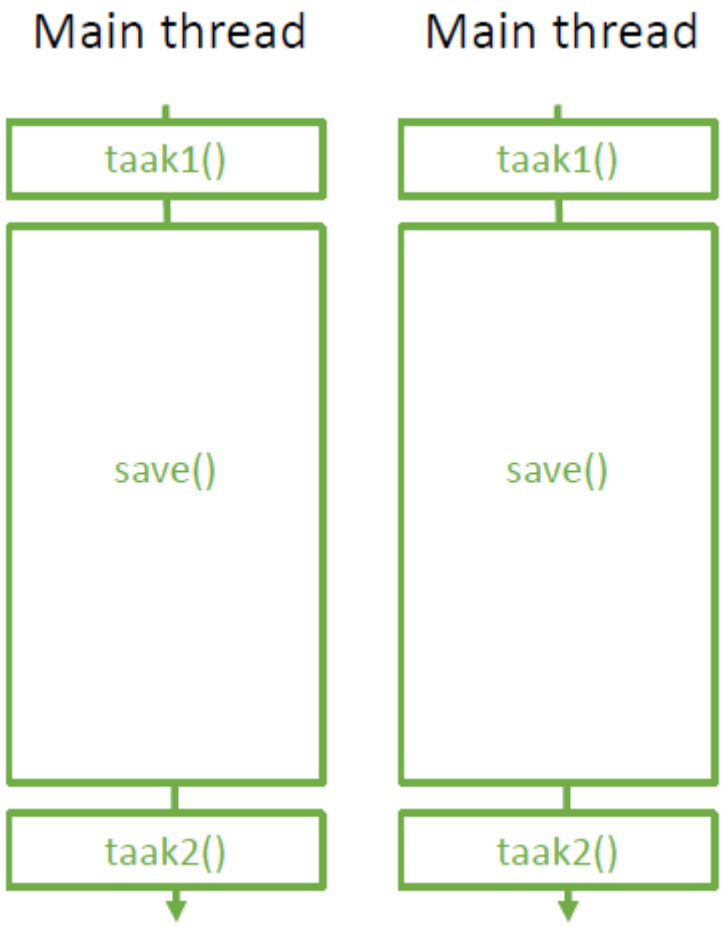


Multiprocessing

Multithreading != Multiprocessing

Multiprocessing:

- Meerdere programma's
- Onafhankelijk van mekaar
- Meerdere JVM's



Toepassingen

Wanneer is multithreading nuttig?

Waar kan multithreading gebruikt worden?

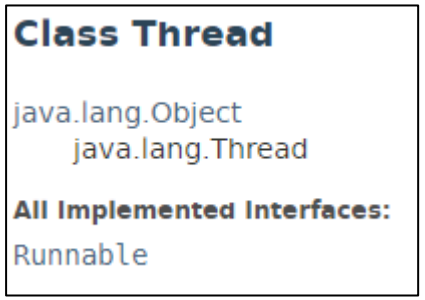
Implementatie

- *Runnable* interface implementeren
  - *run()* methode bevat code die door thread wordt uitgevoerd
  - Minimale vereiste om thread te maken

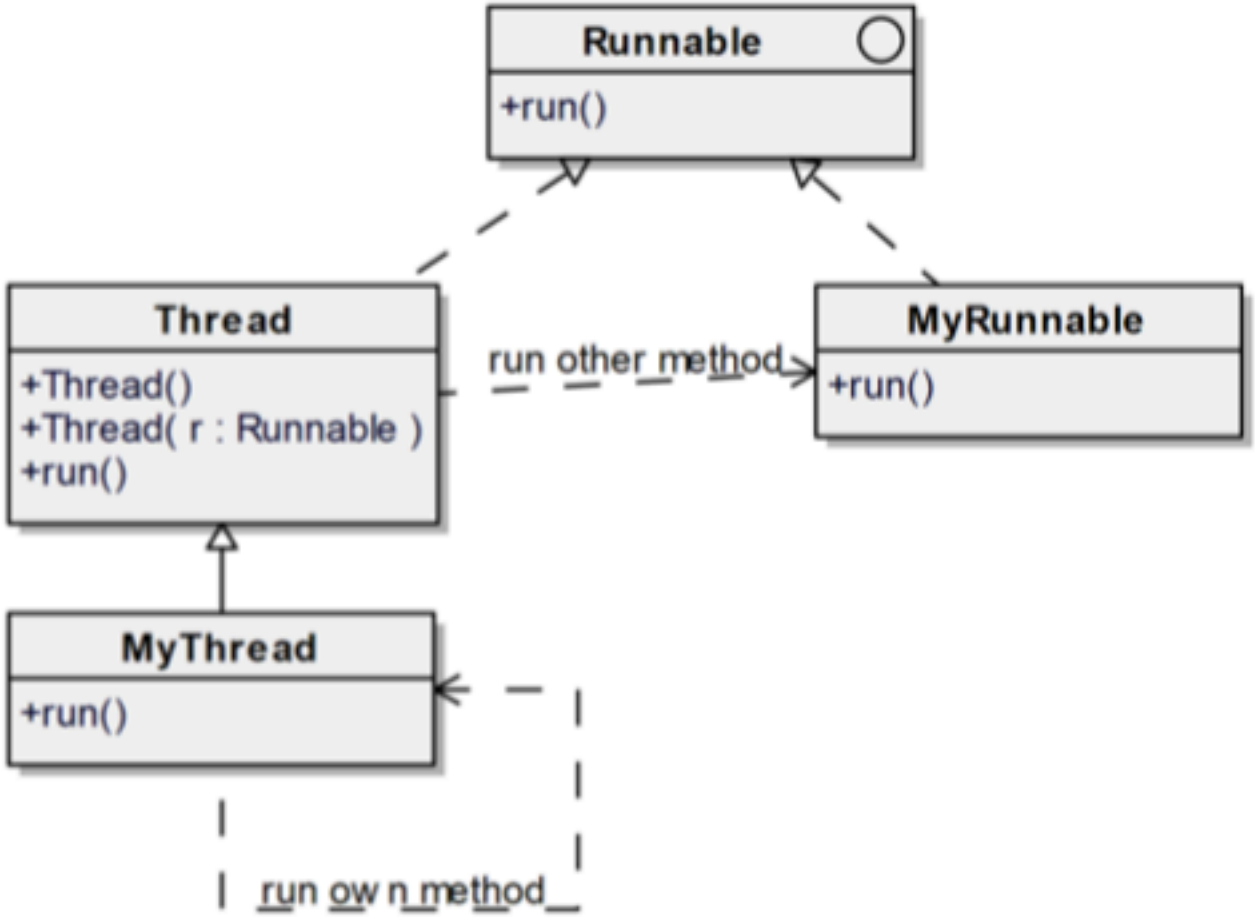
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Runnable.html

- *Thread* klasse overerven
  - Implementeert zelf *Runnable*
  - Voegt extra functies toe

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Thread.html



Implementatie



Thread

```
public class WorkerThread extends Thread {

    @Override
    public void run() {
        System.out.println("Executing thread");
    }

    public static void main(String args[]) {
        new WorkerThread().start();
    }
}
```

Runnable

```
public class WorkerThread implements Runnable {

    @Override
    public void run() {
        System.out.println("Executing thread");
    }

    public static void main(String args[]) {
        (new Thread(new WorkerThread())).start();
    }
}
```

Runnable (with lambda)

```
public class WorkerThread {

    public static void main(String args[]) {
        new Thread(() -> System.out.println("Executing thread")).start();
    }
}
```

Runnable vs Thread

?

Runnable vs Thread

- Thread lijkt ‘handiger’
- Extra methoden, makkelijk bruikbaar

- **Maar:**
  - Klasse kan maar overerven van 1 andere klasse
  - Runnable makkelijker toe te voegen aan bestaande klasse

run vs start

- Thread implementatie
  → run()

- Thread opstarten/uitvoeren
  → start()

- start()
  - De thread opstarten
  - De code geimplementeerd in de run() method zal uitgevoerd worden

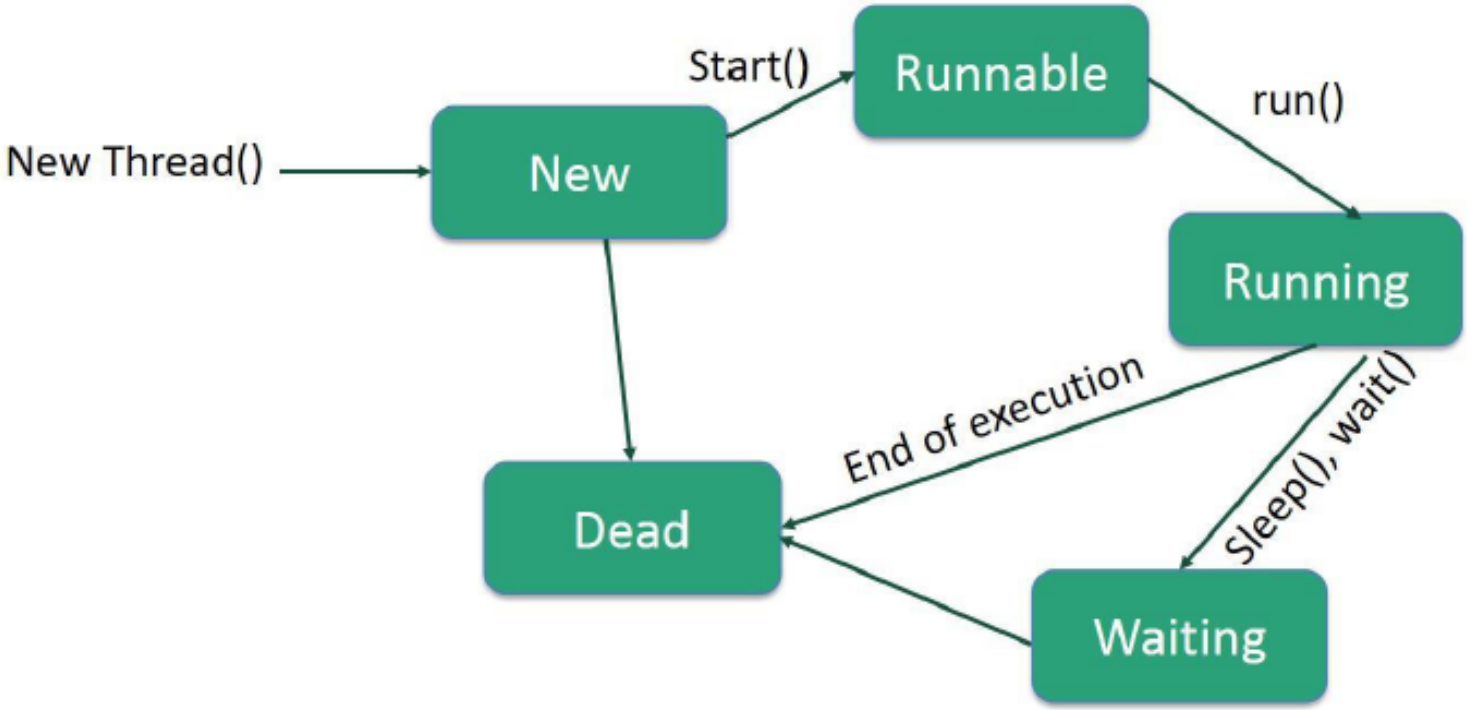
Opgave 1a

- Maak een klasse ***Talker*** die overerft van *Thread*. Aan de constructor kan je een ID mee geven.
- Bij uitvoeren van de thread, moet 10x het ID afgeprint worden, met telkens een halve seconde er tussen. (*sleep(500);*)
- **Maak en start** in de main 4 instanties van *Talker.*

Opgave 1b

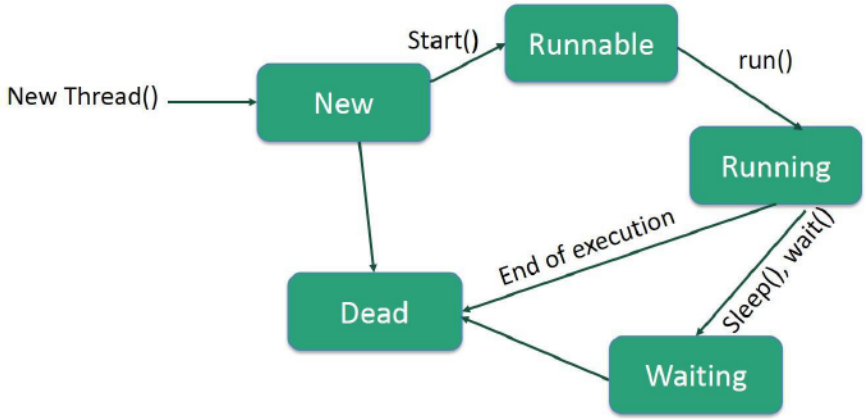
- Doe de nodige aanpassingen om *Talker* nu de *Runnable* interface te laten gebruiken.
- Wat moest er veranderen?

Thread lifecycle



Thread lifecycle

- Uitvoering
  - Opstarten
  - Uitvoeren taak (*run()*)
  - Beëindigen



- States
  - NEW: aangemaakt, nog niet gestart
  - RUNNABLE: Gestart, nog niet ‘ingepland’
  - RUNNING: Uitvoeren van taak, actief
  - WAITING: Uitvoering gepauzeerd
  - DEAD: Taak uitgevoerd

Thread lifecycle

Welke states worden uitgeprint?

```
public static void main(String args[]) {
    Talker talker = new Talker();
    System.out.println(talker.getState());

    talker.start();
    System.out.println(talker.getState());

    try {
        talker.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println(talker.getState());
}
```

Thread scheduler

- 1 actieve thread per processor
- Veel threads, ‘weinig’ processoren
- Threads delen processor

Thread scheduler

- Bepaalt welke thread mag uitvoeren (en hoe lang)
- Verschillende mechanismes spelen rol
- Onderdeel van JVM
- In samenspraak met onderliggend OS

Thread priority

- Prioriteiten toekennen aan threads
  - .setPriority()
  - Scheduler beïnvloeden
  - Geen garantie

- Thread priority beïnvloeden:
  - Thread.sleep(): thread inactief voor bepaalde tijd
  - Thread.join(): deze thread voorrang geven
  - Thread.yield(): andere threads voorrang geven

Thread sleep

- Tijdelijk in wachttoestand
  - Thread.sleep*(milliseconds)*
  - Altijd van toepassing op huidige Thread
- Running → Waiting
  - Uit wachttoestand halen
    - Door timeout
    - Aanroepen method interrupt()
- Nooit wachtlus gebruiken maar sleep !
- Opdracht 5 (TimeBomb) pagina 196

Thread join

- Thread voorrang geven
- Wachten op beeindiging van andere Thread
- .join()

- Demo voorbeeld pagina 198

Volgende week

- Synchronisatie
- TimerTask
- Concurrency
- Parallellisme met streams

Oefeningen

- Opgave: zie Blackboard