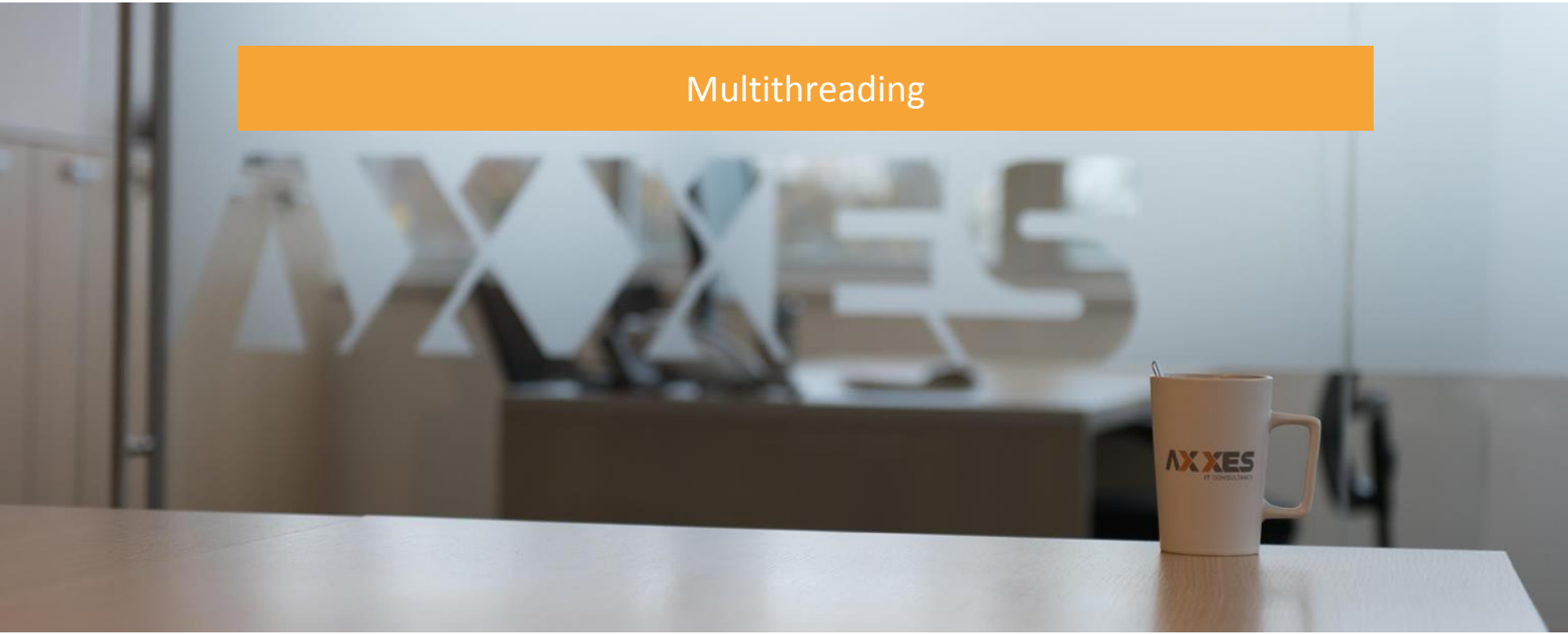
AXXES
IT CONSULTANCY

Multithreading



AXXES
IT CONSULTANCY

Inhoud

1. Intro tot multithreaded software
2. Basisbegrippen
3. Building blocks
4. ExecutorService
5. Completable Future API

IT IS ABOUT PEOPLE

INTRO

AXXES
IT CONSULTANCY

Multithreaded software

- Concurrent execution
- Hogere performantie
- Moeilijker
- Bugs moeilijk te reproduceren

AXXES
IT CONSULTANCY

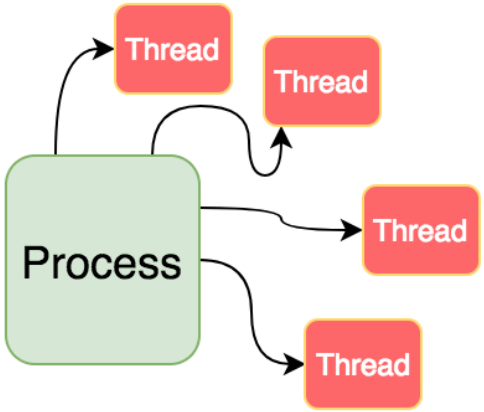
Waarom dan?

- “Writing software is hard, writing concurrent software is even harder”
- Singlethreaded software ondenkbaar tegenwoordig
- Webservers, GUI applicaties, crawlers, garbage collection,...

AXXES IT CONSULTANCY

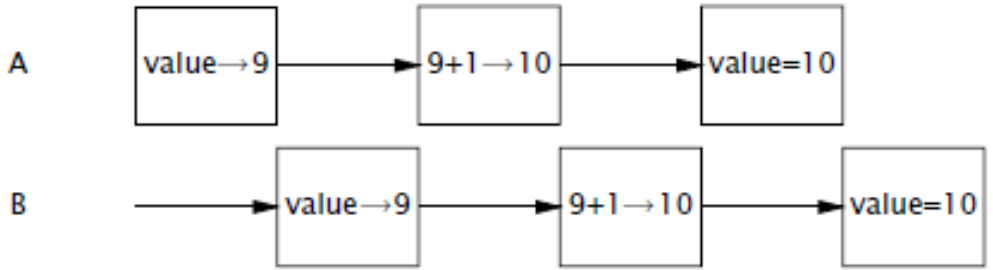
Threads

- Basic unit of scheduling
- Onderdeel van een proces
- Sequentiële uitvoering op 1 processorkern
- Hogere throughput



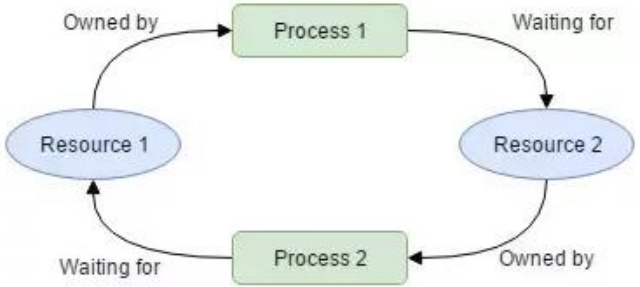
Risico’s - Incorrectness

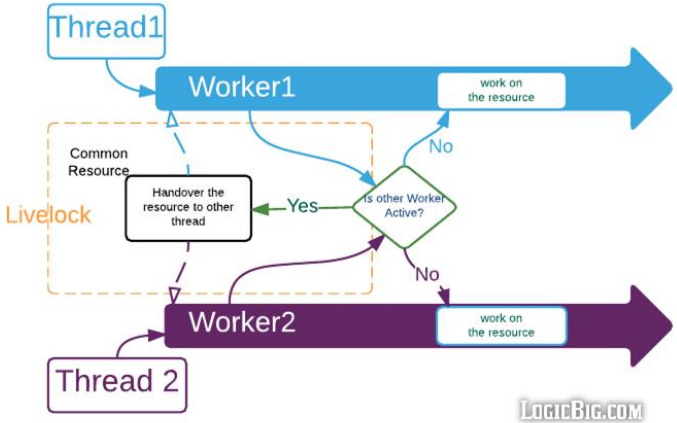
- Interleaving
- Race conditions
- *“Nothing bad ever happens”*



Risico’s - Liveness

- *“Something good eventually happens”*
- Deadlock
- Livelock
- Starvation





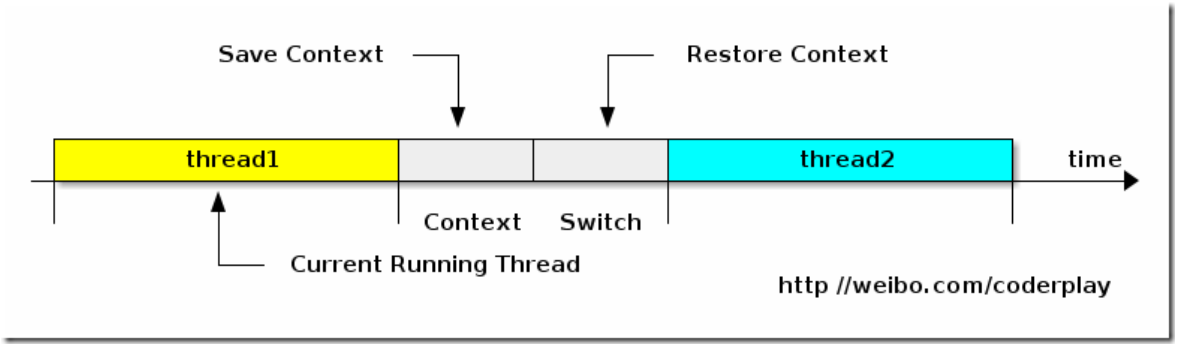
AXXES
IT CONSULTANCY

Risico’s - Starvation

- Eén of meerdere threads krijgen geen CPU tijd
- Teveel contention

Risico’s - Performance

- *Overhead*
- *Thread-switching*
- *Synchronisatie*



IT IS ABOUT PEOPLE

Oefeningen

IT IS ABOUT PEOPLE

Basisbegrippen

AXXES
IT CONSULTANCY

Thread-safety

“A class is thread-safe if it behaves correctly when accessed from multiple threads, regardless of the scheduling or interleaving of the execution of those threads by the runtime, and with no additional synchronization or other coordination on the part of the calling code”

- Java Concurrency in Practice

Correct gedrag

- Specificatie
- Invarianten
- Postcondities

```
public class Counter {
        private int count = 0;

        public int value() {
                return
count;
        }

        public void
increment() {
                count++;
        }
}
```

AXXES IT CONSULTANCY

State

- Waarde van een object
- Thread safety = controlled access to shared, mutable state

Stateless

- Altijd thread-safe
- Merk op: long[] factors is state

  -> Niet shared

  -> *thread confinement*

```
public class StatelessFactorizer {

        public long[] factorize(long i) {
                long[] factors =
factors(i);
        return factors;
    }

    private long[] factors(long i) {
                // Calculate factors
    }
}
```

AXXES IT CONSULTANCY

Stateful

- Synchronisatie nodig om thread-safe te zijn

```
public class CountingFactorizer {

        private long count = 0;

        public long getCount() { return count; }

                public long[] factorize(long i) {
                count++;
                long[] factors = factors(i);
                return factors;
        }
}
```

Compound actions

- Read-modify-write (CountingFactorizer)
- Check-then-act (LazySingleton)
- Atomische operatie

```
public class LazySingleton {

        private LazySingleton instance = null;

                public LazySingleton getInstance() {
                        if (instance == null) {
                                instance =
new LazySingleton();
                        }
                        return instance;
                }
}
```

Atomicity

- Ondeelbare actie
- JDK biedt enkele atomic classes aan

```
public class AtomicCountingFactorizer {

        private AtomicLong count = new AtomicLong();

        public long getCount() { return count.get(); }

                public long[] factorize(long i) {
                count.getAndIncrement();
                long[] factors = factors(i);
                return factors;
        }
}
```

```
public class CachingFactorizer {

        private AtomicLong lastValue = new AtomicLong();
        private AtomicReference<List<Long>> lastFactors = new AtomicReference<>();

                public List<Long> factorize(long i) {
                if (i == lastValue.get()) {
                                return lastFactors.get();
                }
                List<Long> factors = calulateFactors(i);
                lastValue.set(i);
                lastFactors.set(factors);
                return factors;
        }
}
```

AXXES
IT CONSULTANCY

Locks

- Intrinsic/monitor lock
- Synchronized block/method
- mutex (**mut**ually **ex**clusive)

```
synchronized (monitor) {
        // code goes here
}

public synchronized void someMethod() {
        // code goes here
}

public void someMethod() {
        synchronized(this) {
                // code goes here
        }
}
```

```
public class CachingFactorizer {

        private AtomicLong lastValue = new AtomicLong();
        private AtomicReference<List<Long>> lastFactors = new AtomicReference<>();

                public synchronized List<Long> factorize(long i) {
                if (i == lastValue.get()) {
                                return lastFactors.get();
                }
                List<Long> factors = calculateFactors(i);
                lastValue.set(i);
                lastFactors.set(factors);
                return factors;
        }
}
```

```
public class CachingFactorizer {

        private AtomicLong lastValue = new AtomicLong();
        private AtomicReference<List<Long>> lastFactors = new AtomicReference<>();

                public List<Long> factorize(long i) {
                synchronized (this) {
                        if (i == lastValue.get()) {
                                        return lastFactors.get();
                        }
                }
                List<Long> factors = factors(i);
                synchronized (this) {
                        lastValue.set(i);
                        lastFactors.set(factors);
                }
                return factors;
        }
}
```

AXXES
IT CONSULTANCY

Critical section

- Deel code dat nood heeft aan synchronized access
- Is omvat door een locking mechanism

AXXES
IT CONSULTANCY

Explicit locks

- Advanced API
- Reentrancy
- Fairness (longest waiting thread)

```
public class LockDemo {

        private Lock lock = new
ReentrantLock();

        public void someMethod() {
                lock.lock();
                try { // code goes here }
                finally { lock.unlock(); }
        }
}
```

AXXES
IT CONSULTANCY

ReadWrite locks

- Multiple readers
- Single writer

```
public class ReadWriteLockDemo {

        private ReadWriteLock lock = new
ReentrantReadWriteLock();
        private Lock readLock = lock.readLock();
        private Lock writeLock = lock.writeLock();
}
```

IT IS ABOUT PEOPLE

Oefeningen

IT IS ABOUT PEOPLE

Building Blocks

AXXES IT CONSULTANCY

Synchronized Collections

- Elke public method synchronized
- Wel threadsafe, niet performant
- Compound actions vereisen nog steeds externe synchronisatie

```
Collections.synchronizedList(new ArrayList<>());
Collections.synchronizedSet(new HashSet<>());
Collections.synchronizedMap(new HashMap<>());
```

```
public Object getLast(List list) {
        int lastIndex = list.size() - 1;
        return list.get(lastIndex);
}
```

AXXES
IT CONSULTANCY

Concurrent Collections

- Java 1.5
- java.util.concurrent package
- Collection implementaties optimized voor concurrent access

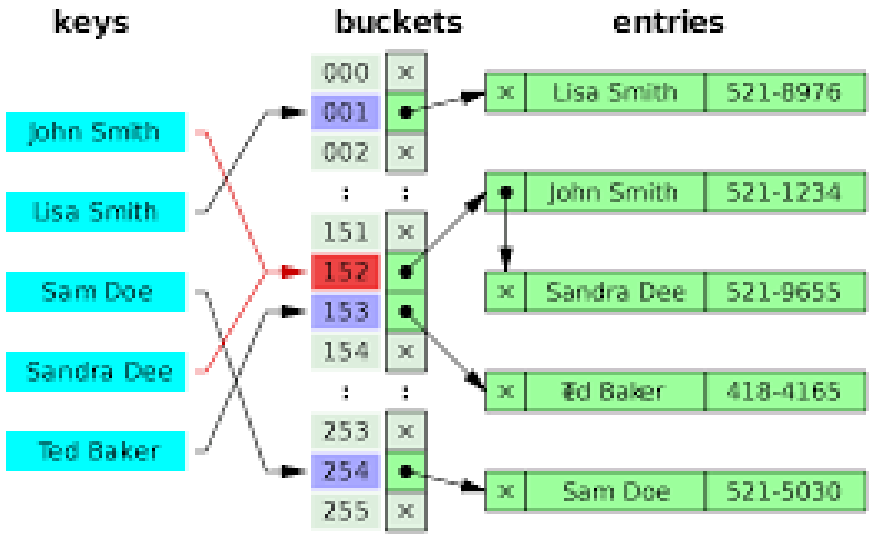
AXXES
IT CONSULTANCY

CopyOnWriteArrayList/HashSet

- Geen locking voor reads
- Bij write wordt een kopie gemaakt
- Optimized voor collecties die veel vaker uitgelezen dan aangepast worden

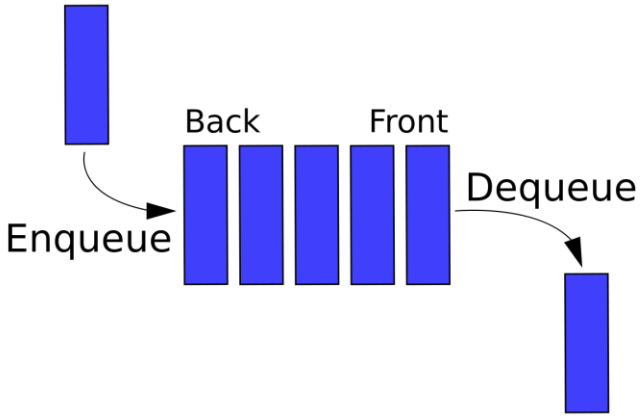
ConcurrentHashMap

- Geen locking voor reads
- 16 concurrent writers mogelijk
- Geen custom compound actions mogelijk
- put-if-absent, remove-if-equal, replace-if-equal zijn voorzien



Queue & BlockingQueue

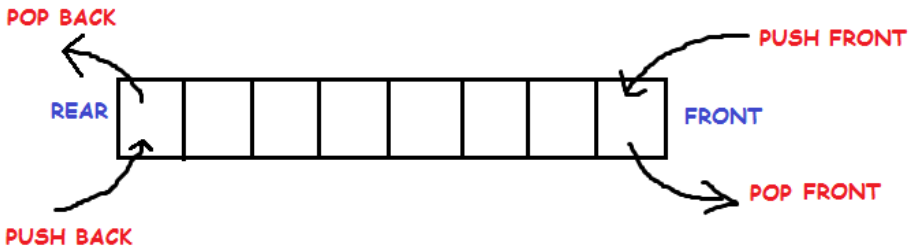
- Collectie om elementen te queuen en op andere thread uit te lezen
- FIFO
- offer() en poll()
- put() en take()



AXXES IT CONSULTANCY

Deque & BlockingDeque

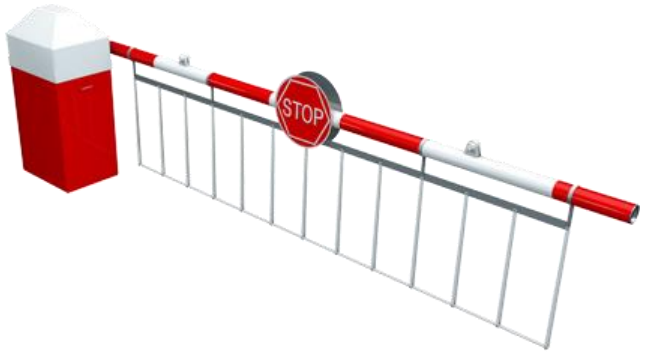
- Deque = Double-ended Queue
- offerFirst/Last() en pollFirst/Last()
- putFirst/Last() en takeFirst/Last()



AXXES
IT CONSULTANCY

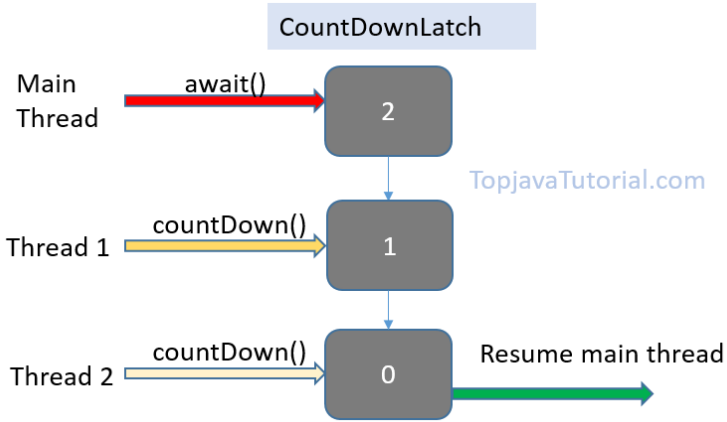
Synchronizers

- Bewaken toegang tot resources
- Hebben state en logica om te bepalen of een thread door mag of moet wachten



Latch

- Gesloten tot een bepaalde *terminal state* bereikt wordt
- Daarna continu open
- new CountDownLatch(2);



AXXES IT CONSULTANCY

Semaphore

- Beheert een aantal virtual permits
- Handig om connection pools af te schermen
- Intrinsic lock = semaphore met 1 permit

```
Semaphore semaphore = new Semaphore(10);
semaphore.acquire();
try {
        // some code
} finally {
        semaphore.release();
}
```

AXXES IT CONSULTANCY

Barrier

- Om te wachten tot andere threads een bepaald punt bereikt hebben
- Barrier action
- Simulaties

```
CyclicBarrier barrier = new CyclicBarrier(2);
new Thread(() -> {
        barrier.await();
        // more code
}.start();
new Thread(() -> {
        barrier.await();
        // more code
}.start();
```

IT IS ABOUT PEOPLE

Oefeningen

IT IS ABOUT PEOPLE

Executor Framework

AXXES
IT CONSULTANCY

Tasks

- Discrete unit of work
- Task boundaries
- Onafhankelijk
- Execution policy

Sequential Execution Policy

- Single-threaded
- Geen thread-safety nodig
- Slechte performantie

```
public class WebServer {

        public static void main(String[] args) {
                ServerSocket socket = new
ServerSocket(80);
                while (true) {
                        Socket request =
socket.accept();

        handleRequest(request);
                }
        }
```

Unbounded Thread Creation

- Nieuwe thread per task
- Submitting los van verwerking
- Veel overhead
- Instabiel

```
public class WebServer {

        public static void main(String[] args) {
                ServerSocket socket = new
ServerSocket(80);
                while (true) {
                        Socket request =
socket.accept();
                        new Thread(() ->
handleRequest(request)
                .start();
                }
        }
```

Thread pooling

- Collectie van long-living threads
- Threads zijn herbruikbaar

```
public class WebServer {

        public static void main(String[] args) {
                ExecutorService executor =
Executors.newFixedThreadPool(50);
                ServerSocket socket = new ServerSocket(80);
                while (true) {
                        Socket request = socket.accept();
                        executor.submit( () ->
handleRequest(request)).start();
                }
```

AXXES
IT CONSULTANCY

ExecutorService

- Abstractie van thread pools
- API voor asynchrone tasks
- Uitgebreide execution policies:
  - Max concurrent tasks
  - Volgorde
  - Queue size
  - Task hooks

AXXES
IT CONSULTANCY

Executors

- Factory methods voor ExecutorServices

```
ExecutorService executor = Executors.newSingleThreadExecutor();
ExecutorService executor = Executors.newFixedThreadPool(4);
ExecutorService executor = Executors.newCachedThreadPool();
ScheduledExecutorService executor = Executors.newScheduledThreadPool(4);
```

AXXES
IT CONSULTANCY

Runnable vs Callable

- Allebei abstractie van een asynchrone taak
- Callable geeft een resultaat terug

```
ExecutorService ex = Executor.newSingleThreadExecutor();
Runnable runnable = () -> doSomething();
Future future = ex.submit(runnable);
```

```
ExecutorService ex = Executor.newSingleThreadExecutor();
Callable<String> callable = () -> return “Hello world!”;
Future<String> future = ex.submit(callable);
```

AXXES IT CONSULTANCY

CompletionService

- ExecutorService + BlockingQueue
- Resultaat van de taak wordt op queue gezet

```
ExecutorService executor = Executors.newSingleThreadExecutor();
CompletionService cs = new ExecutorCompletionService(executor);
for (int i = 0; i < 100; i++) {
        Callable<Image> callable = () -> loadImage(i);
        cs.submit(callable);
}
List<Image> images = new ArrayList<>();
for (int i = 0; i < 100; i++) {
    Image image = cs.take();
    images.add(image);
}
```

AXXES
IT CONSULTANCY

ThreadPoolExecutor

- Standaard implementatie van ExecutorService
- Zelf instantiëren -> volledige controle over configuratie
- RejectedExecutionPolicy: AbortPolicy (default), DiscardPolicy, CallerRunsPolicy,...
- Hooks: beforeExecute(), afterExecute(), terminated()

IT IS ABOUT PEOPLE

Oefeningen

IT IS ABOUT PEOPLE

CompletableFuture API

AXXES
IT CONSULTANCY

CompletableFuture API

- Java 8
- Verbetering op Future interface
- Volledig framework voor asynchrone operaties te beheren

AXXES
IT CONSULTANCY

CompletableFuture - Callbacks

```
Future<String> future = new FutureTask(() -> “Hello!”);
future.run();
String result = future.get();
System.out.println(result);
```

```
CompletableFuture.supplyAsync(() -> “Hello!”)
                                 .thenAccept(result ->
System.out.println(result));
```

AXXES
IT CONSULTANCY

CompletableFuture - Chaining

```
Future<String> future = new FutureTask(() -> “Hello”);
future.run();
future = new FutureTask(() -> future.get() + “ world!”);
future.run();
System.out.println(future.get());
```

```
CompletableFuture.supplyAsync(() -> “Hello!”)
                        .thenCompose(result -> CompletableFuture.supplyAsync(() -> result + “
world!”))
                        .thenAccept(result -> System.out.println(result));
```

AXXES
IT CONSULTANCY

CompletableFuture - Combining

```
FutureTask<String> f1 = new FutureTask(() -> “Hello Alice!”);
FutureTask<String> f2 = new FutureTask(() -> “Hello Bob!);
f1.run();
f2.run();
System.out.println(f1.get());
System.out.println(f2.get());


CompletableFuture<Void> future = CompletableFuture.allOf(
                CompletableFuture.supplyAsync(() -> “Hello Alice!”),
                CompletableFuture.supplyAsync(() -> “Hello Bob!”)
);
```

AXXES
IT CONSULTANCY

CompletableFuture - Combining

```
List<CompletableFuture<String>> futures = Arrays.asList(
        CompletableFuture.supplyAsync(() -> “Hello Alice!”),
        CompletableFuture.supplyAsync(() -> “Hello Bob!”)
);
CompletableFuture.allOf(futures.toArray())
                .thenAccept(v -> {
                        futures.stream()
                .map(Future::get)
                .forEach(result -> System.out.println(result))
                });
```

AXXES
IT CONSULTANCY

CompletableFuture - Exception Handling

```
Future<String> future = new FutureTask(() -> {
        int i = new Random().nextInt();
        if (i % 2 == 0) {
                throw new RuntimeException("Error"));
        } else {
                return "'Success";
        }
}
try {
                future.get()
} catch (Exception ex) {
                // exception handling goes here
}
```

AXXES
IT CONSULTANCY

CompletableFuture - Exception Handling

```
CompletableFuture.supplyAsync(() -> potentiallyFailingMethod())
                        .exceptionally(ex -> {
                    System.out.println(ex.getMessage();
                    return “default value”);
                        })
                        .thenAccept(result -> System.out.println(result);
```

AXXES
IT CONSULTANCY

Feedback

https://forms.office.com/Pages/DesignPage.aspx?fragment=FormId%3DtsPR7Ye-u0OS1HqRHFzuF9nICjxPwLpIlxQKqEeGerhUQjVXRUMzVkZPUktNVFlNSFAwRzdLUlJMTy4u%26Token%3D2441c34e14a04015a02c69fef119db60