cphbusiness

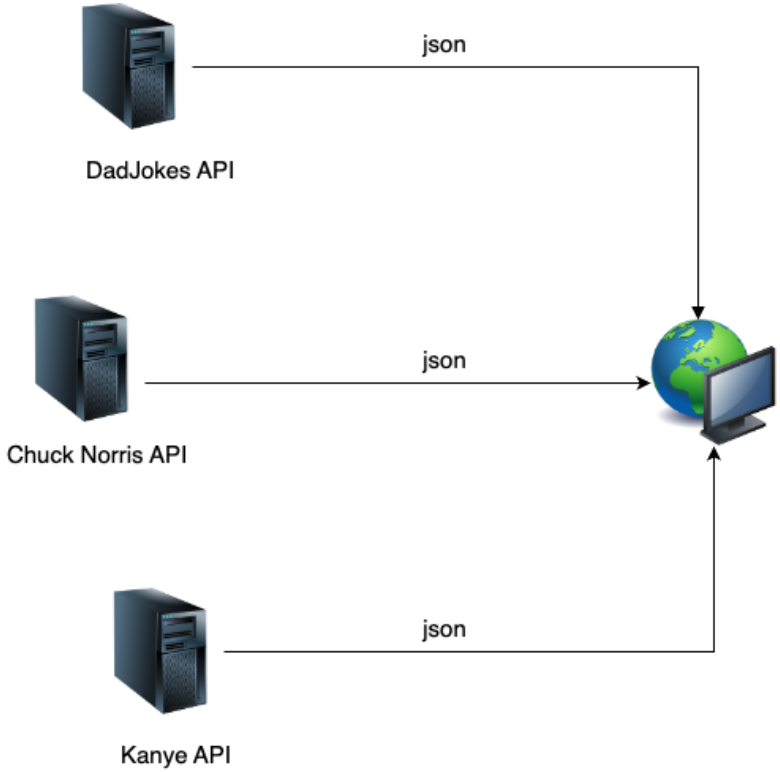
COPENHAGEN BUSINESS ACADEMY

Threads
Runnable
Executors
Futures and Callables

(Making remote HTTP-requests)

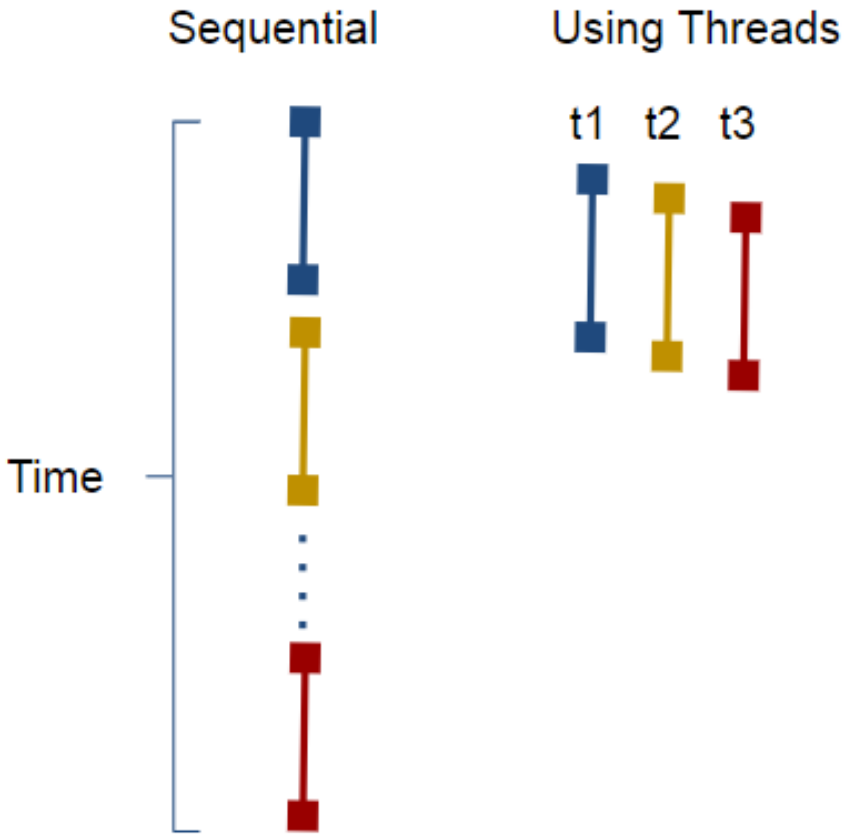
WHY Todays Topic's

cphbusiness



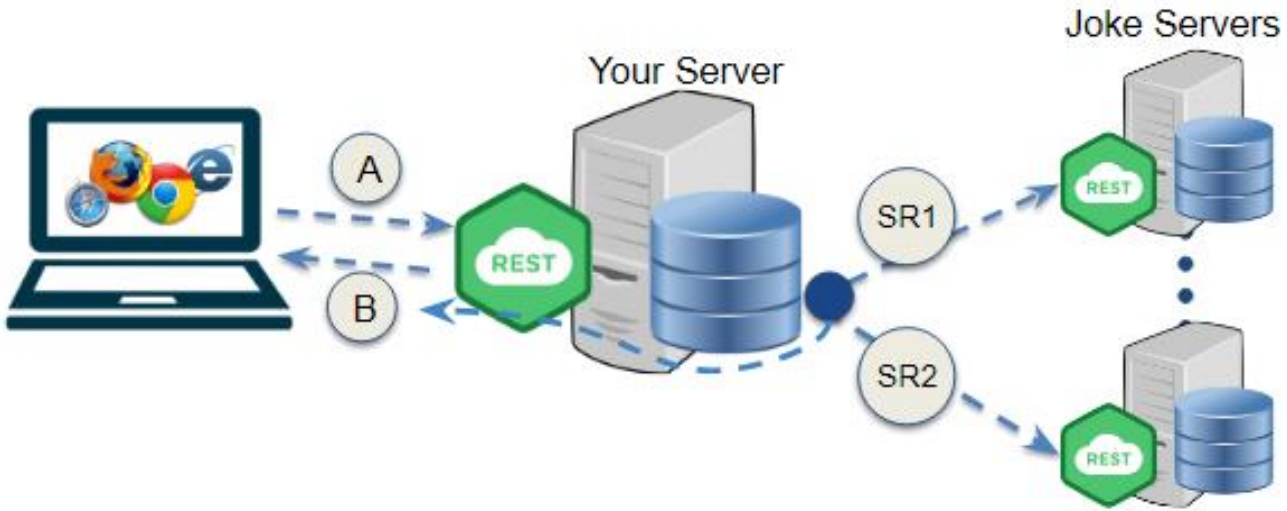
WHY

cphbusiness



The Goal: our own broker api

cphbusiness

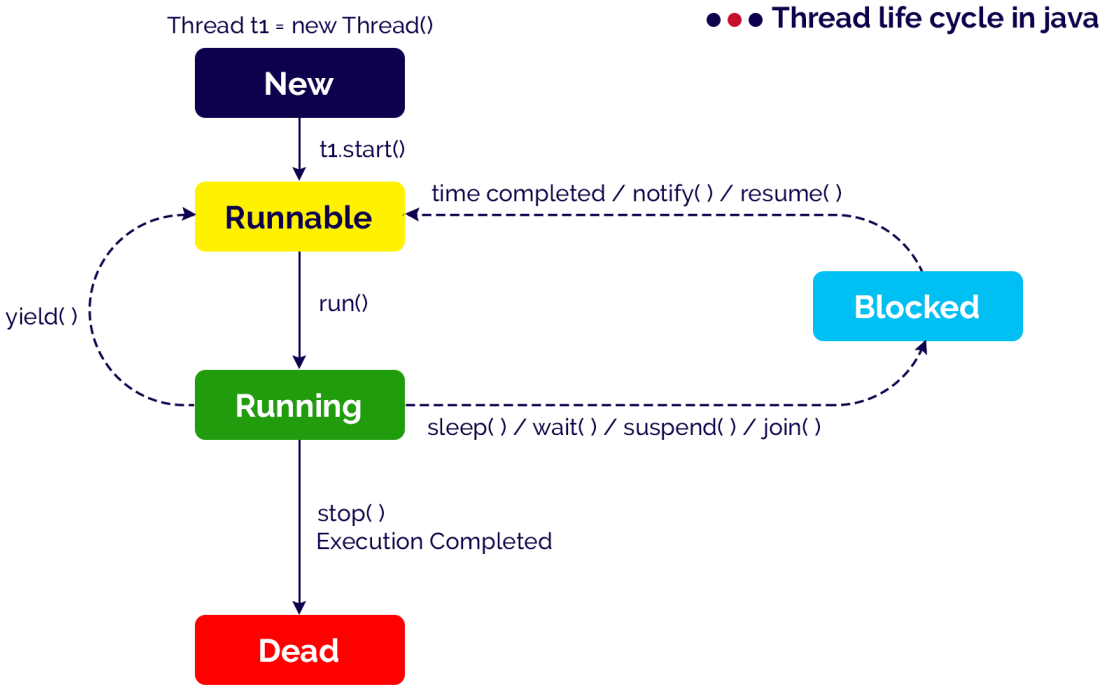


Short recap on threads

cphbusiness

Super Simplified Diagram of a Threads Life Cycle

cphbusiness



www.btechsmartclass.com Link

Immutability

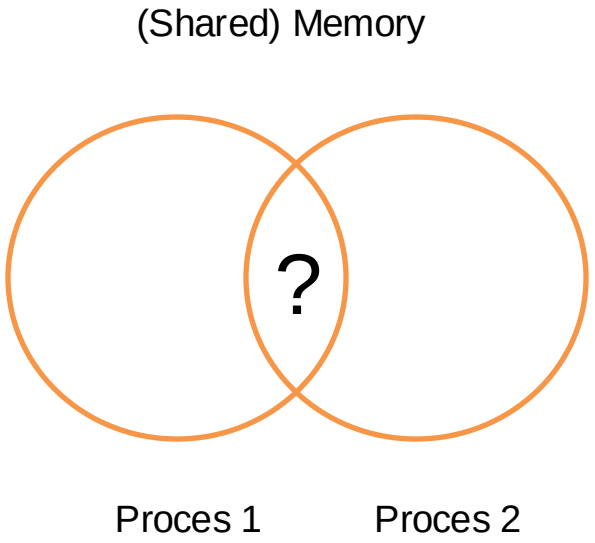
cphbusiness

Immutable = cannot be mutated
Same as final keyword in Java

Shared memory leads to race-conditions and starvation.
And Possibly deadlocking

Solution: Don't change values!

New problem: How do get data back from threads?



Runnable interface

cphbusiness

Runnable interface to normal Threads as well as executors

Has one abstract method `run()` that takes no arguments and returns void:

```
public class MyTask implements Runnable{
  @Override
  public void run() {
     // Your method here
  }
}
Thread t = new Thread(new MyTask());
t.run();
t.start();
```

"Never" create your threads like this. Use one of the thread-pools supplied by the `Executors` class

ExecutorService's

cphbusiness

- Threads have overhead (time and memory)
- Better to divide your work into tasks, and let **reusable** threads run them.
- "Tasks" is what we call Runnables (and Callables) in Java, when we call them through **executors**

	Thread	Task
Work	6581 ns	6612 ns
Create	1030 ns	77 ns
Create+start/(submit+cancel)	48929 ns	835 ns
Create+(start/submit)+complete	72759 ns	21226 ns

ExecutorService

cphbusiness

ExecutorService's are usually created via one of the factory methods in the **Executors** class as outlined below:

Executors with Runnables

```
ExecutorService es = Executors.newXXXThreadPool();
es.execute(task1);  //task1 must be a Runnable-instance
es.execute(task2);
…
es.shutdown();
es.awaitTermination();
```

ExecutorService: Pool types

cphbusiness

`NewCachedThreadPool()`
Creates a thread pool that creates new threads as needed, but will reuse previously constructed threads when they are available. If threads are idle for 60 seconds, they are terminated. Suitable for many short-lived tasks.

`newFixedThreadPool(int nThreads)`
Creates a thread pool that reuses a fixed number of threads operating off a shared unbounded queue.

`NewSingleThreadExecutor()`
Creates an Executor that uses a single worker thread operating off an unbounded queue. Ideal for scenarios where only one task should run at a time.

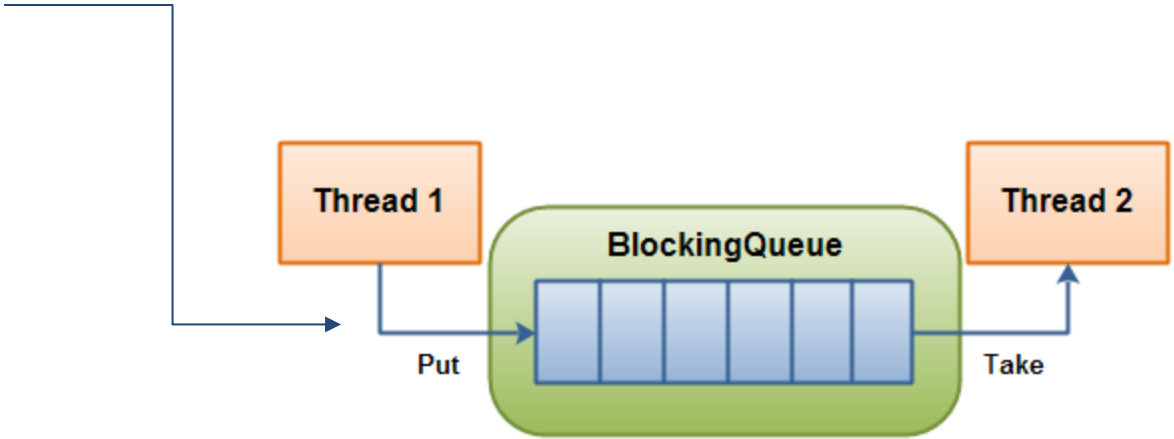
`newScheduledThreadPool()`
Executes periodically (eg. For database clean up) Useful for tasks that need to be executed on a timed schedule.

`WorkStealingPool() From Java8`
Creates a thread pool that maintains enough threads to support the given parallelism level, and may use multiple queues to reduce contention.

A Producer/Consumer Solution

cphbusiness

En “task” køres via `run()` metoden i en `Runnable`



Getting results back from Threads

cphbusiness

Shared memory leads to race-conditions and starvation. And Possibly deadlocking

Solution: Don’t change values!

New problem: How do get data back from threads?

Future and Callable

Callable interface

cphbusiness

Callable interface has one abstract method `call()` that takes no arguments
Generic type: `Object of <T>` where T can be any Class

Below is an example with T as String:

```
import java.util.concurrent.Callable;
public class MyTask implements Callable<String> {
  @Override
  public String call() throws Exception {
    // Your method that returns a String here
    return "hello";
  }
}
--------------------
MyTask t = () -> "Hello!";
```

Future interface

cphbusiness

Threads are *asynchronous*, so we generally do not know when we will get the result back. So, how do we extract the value from a Callable?

`Future<T>` comes to the rescue

```
Future<String> future = executor.submit(
  new Callable<String>(){
    @Override
    public String call(){
      //do stuff
    }
  }
);
```

Working with futures

cphbusiness

A Future represents work that will be done at some point in the "future", hence the name. A bit like *javascript promises*!

There are several ways to get the result from a Future:

```
//Get the result when it's ready. Blocks the thread until then.
future.get();
//If you want to escape from the blocking method
//no later than a given time, you can set that.
future.get(10, TimeUnit.MINUTES);
//Or you can ask if the task is done (returns a boolean).
future.isDone();
```

ExecutorService

cphbusiness

ExecutorService's are usually created via one of the factory methods in the **Executors** class as outlined below:

Executors with Callables (and Futures)

```
ExecutorService es = Executors.newXXXThreadPool();
//call1 must be a Callable object of type String. Observe how
//submit(..) returns a Future
Future<String> f1 = es.submit(call1)
...
//Get the result (when ready) from the future
System.out.println(f1.get());
...
```

Let’s create an example

cphbusiness

Jokes