THEORY FOR EXERCISE #1 - GO UNLEACHED

- Java 8 Lambda Expressions
- Blocking I/O vs. non-blocking I/O
  - Using Spring MVC
- Configuration
  - Using Spring Boot
- Some details…

JAVA 8 LAMBDA EXPRESSIONS

- Using an *anonymous inner class* for a callback:

```
asyncHttpClient.execute(url,

    new AsyncCompletionHandler<Response>() {
        @Override
        public Response onCompleted
            (Response response) {

            // TODO: Handle the response...
        }
    }
);
```

JAVA 8 LAMBDA EXPRESSIONS

- Using an *anonymous inner class* for a callback:

```
asyncHttpClient.execute(url,

    new AsyncCompletionHandler<Response>() {
        @Override
        public Response onCompleted
                (Response response) {

            // TODO: Handle the response...
        }
    }
);
```

This is just overhead!!

JAVA 8 LAMBDA EXPRESSIONS

- Anonymous inner class

```
asyncHttpClient.execute(url,

    new AsyncCompletionHandler<Response>() {
        @Override
        public Response onCompleted
                (Response response) {

            // TODO: Handle the response...
        }
    }
);
```

Gone using Lambdas!!!

- Lambda expression

```
asyncHttpClient.execute(url,

    (Response response) ->  {

        // TODO: Handle the response...
    }
);
```

JAVA 8 LAMBDA EXPRESSIONS

- Type inference

```
asyncHttpClient.execute(url,

  response ->  {

    // TODO: Handle the response...
  }
);
```

- Local variables

```
final DeferredResult<String> dr =
  new DeferredResult<>();

asyncHttpClient.execute(url,

  response ->  {

    // TODO: Handle the response...
    dr.setResult(response.getResponseBody());
  }
);
```

BLOCKING I/O WITH SPRING MVC

```
@RestController
public class RouterController {

    @Value("${serviceProvider.url}”) private String url;

    @Autowired private RestTemplate restTemplate;

    @Autowired private UtilBlocking util;

    @RequestMapping("/router")
    public ResponseEntity<String> router(@RequestParam String qry) {

        try {
            return restTemplate.getForEntity(url + "/service?qry=" + qry, String.class);

        } catch (RuntimeException ex) {
            return util.handleException(ex, url);
        }
    }
}
```

NON-BLOCKING I/O WITH SPRING MVC

```
@RestController
public class RouterController {

    @Value("${serviceProvider.url}") private String url;

    @Autowired private AsyncHttpClientCallback asyncHttpClient;

    @Autowired private UtilCallback util;
```

NON-BLOCKING I/O WITH SPRING MVC

```
@RequestMapping("/router")
public DeferredResult<ResponseEntity<String>> router(@RequestParam String qry) {

    final DeferredResult<ResponseEntity<String>> dr = new DeferredResult<>();

    asyncHttpClient.execute(url + "/service?qry=" + qry,

        throwable -> {
                util.handleException(throwable, url, dr);
        },

        response -> {
            dr.setResult(util.createResponse(response));
        }
    );

    // Return to let go of the precious thread we are holding on to...
    return dr;
}
```

CONFIGURATION WITH SPRING BOOT

```
@ComponentScan()
@EnableAutoConfiguration
public class Application {

    @Value("${serviceProvider.connectionTimeoutMs}")
    private int serviceProviderConnectionTimeoutMs;

    @Value("${serviceProvider.requestTimeoutMs}")
    private int serviceProviderRequestTimeoutMs;

    @Bean
    public AsyncHttpClient getAsyncHttpClient() {

        AsyncHttpClientConfig config = new AsyncHttpClientConfig.Builder().
            setConnectionTimeoutInMs(serviceProviderConnectionTimeoutMs).
            setRequestTimeoutInMs(serviceProviderRequestTimeoutMs).
            build();

        return new AsyncHttpClient(config);
    }
```

THE DEVIL IS IN THE DETAILS…

- Logging
  - We use LogBack instead of Log4J to avoid scalability bottlenecks in Log4J
- Error handling
  - Communication, service and timeout errors
  - Handled by the Utility-classes
- Left out capabilities (to reduce code complexity)
  - Handling of HTTP headers (e.g. accept headers and custom headers)
  - Handling of Correlation Id for logging
  - Replacement of using ThreadLocal
  - Automatic testing of non-blocking services (See Callista blog)