trainitek

Code Once, Use Everywhere: Building Shared Libraries for Multiple Projects

Vadzim Prudnikau

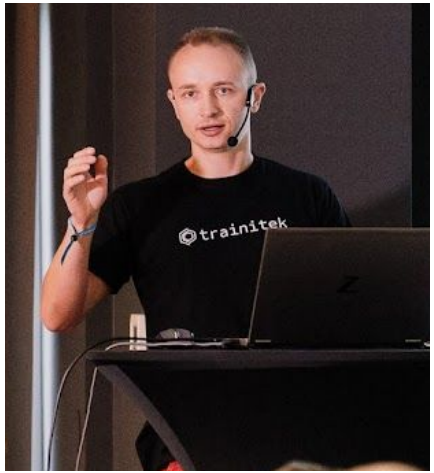
What you won’t learn today

- How to implement a common library in YOUR team
- However, you will get some useful insights if you need to do that

trainitek

Shortly about me

I run training in Spring Boot, Architecture, DDD, TDD, etc.



Co-owner and instructor at trainitek

Hands-on architect at APOTEK 1

Vadzim Prudnikau

vadim-prudnikov

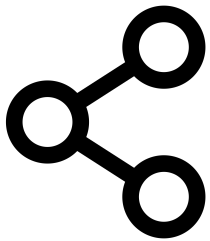
My motivation

- I developed a common library for Apotek 1 that several projects (5) have used for 3 years 💪
- I saw different approaches at work and conferences, and I felt I could do it a bit better 💡
- I would like to share my own experience 💖

What will we address today?

- **When** we need **to share** our code
- **What not** to **put** into the shared code
- **Structure** of the solution **and** its **integration** with **Spring Boot** ⭐
- Documentation
- Making changes
- Forcing vs advertising

When we need to share our code



When we need to share our code

Multiple projects solve the same technical problems

&&

You are tired of copy-pasting the same solutions across projects

&&

&&

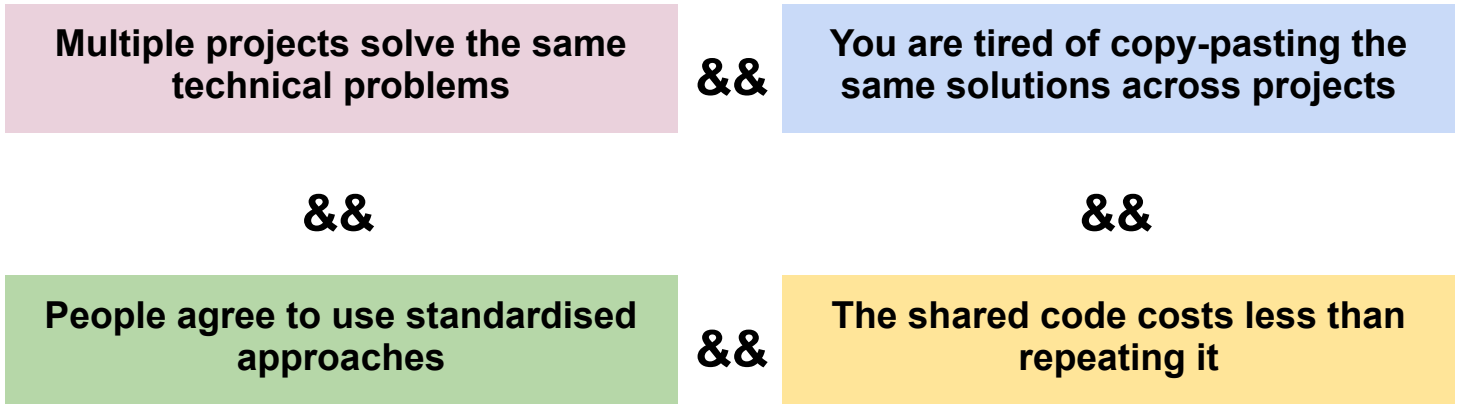
People agree to use standardised approaches

&&

The shared code costs less than repeating it

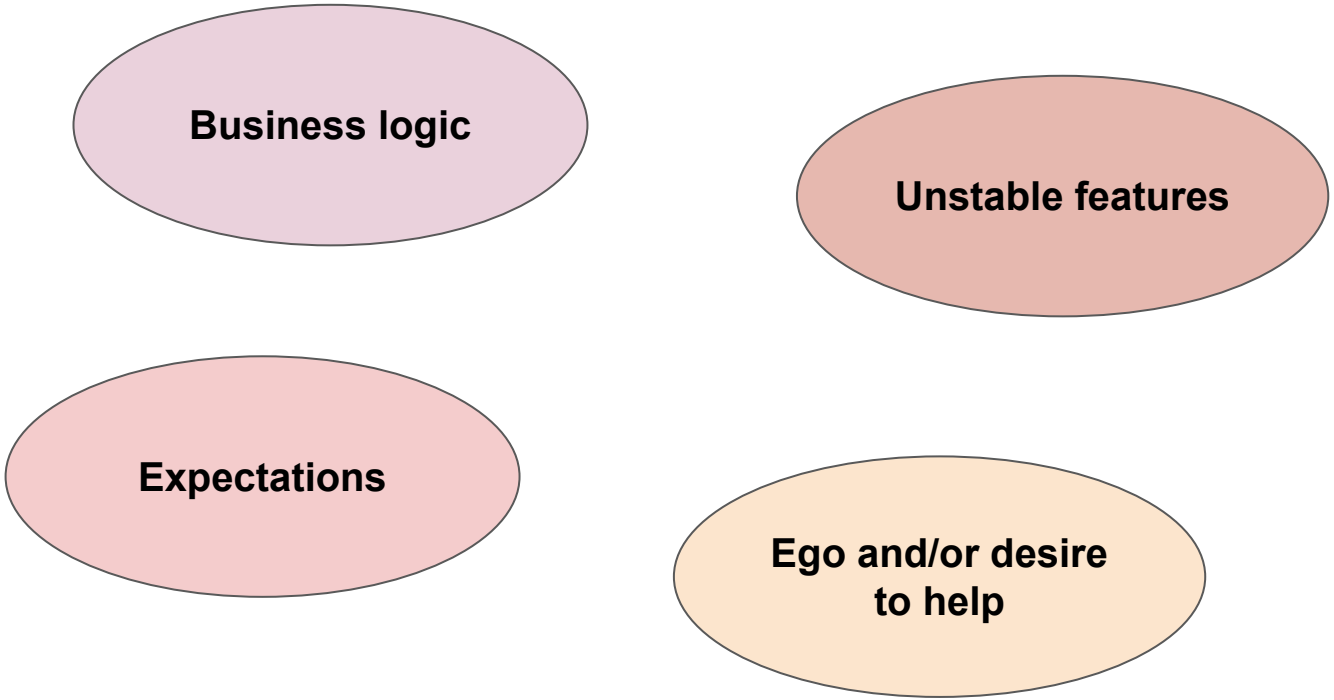
It’s not that easy

- I made mistakes myself regarding these points. It’s quite hard to follow it.

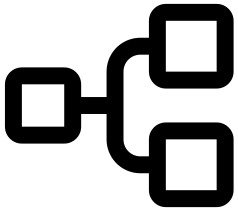


What **not** to put into the shared code

What **not** to put into the shared code



Structure of the library and its integration with Spring Boot



Rationale 1/5

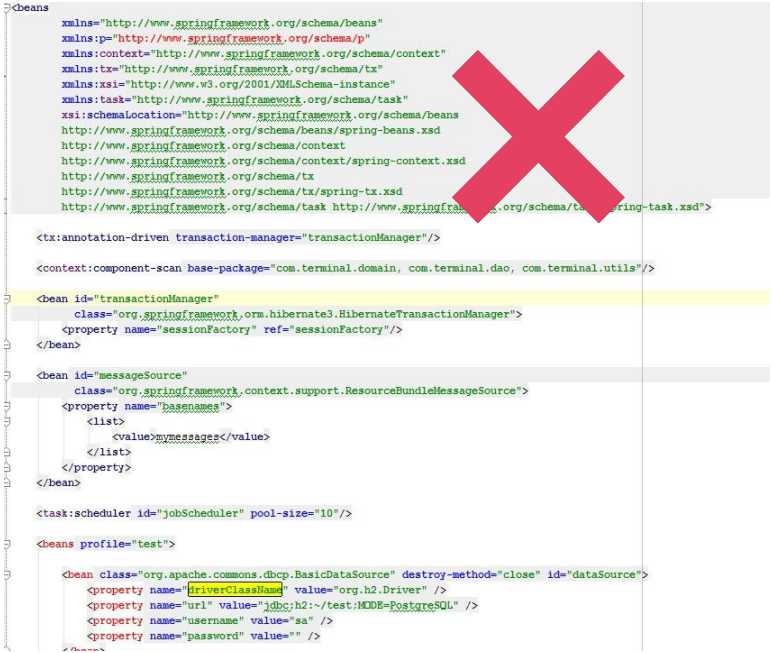
- The library should be easy to include, ideally by adding a Maven dependency

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

trainitek

Rationale 2/5

- Developers should **not** be bothered by the configuration of the library



https://stackoverflow.com/questions/30578950/configure-spring-xml-configuration-namespaces-to-avoid-idea-warnings

Rationale 3/5

- It should be possible to use only specific modules

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Rationale 4/5

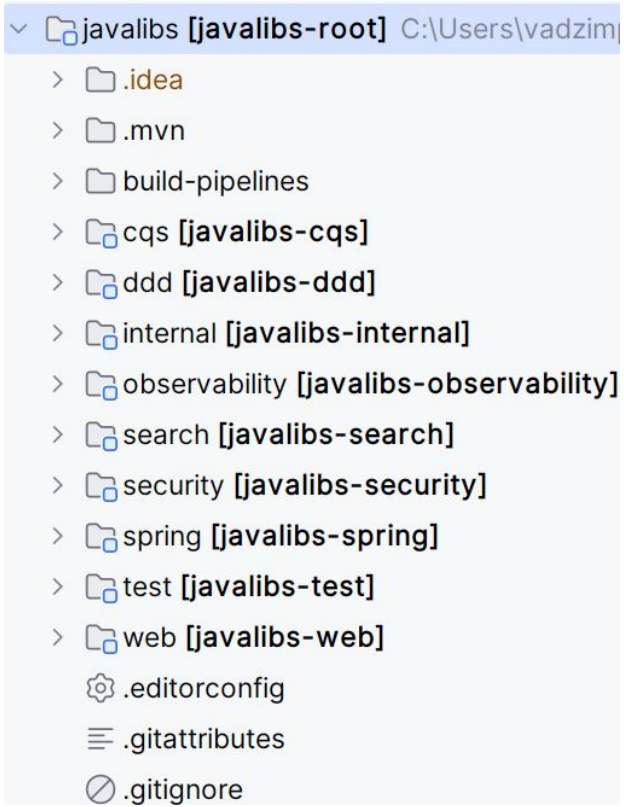
- The library structure should follow and provide some guidelines

Rationale 5/5

- Our code doesn’t always depend on the Spring Framework

trainitek

Root level



trainitek

Root pom.xml

```
<groupId>no.apotek1.javalibs</groupId>
<artifactId>javalibs-root</artifactId>
<version>3.7-SNAPSHOT</version>
<packaging>pom</packaging>

<properties...>

<modules...>

<build>
    <pluginManagement...>

    <plugins...>
</build>

<dependencyManagement...>

<dependencies...>
```

Shared properties: JDK version, dependency versions, etc.

Default plugins settings: checkstyle, surefire, etc.

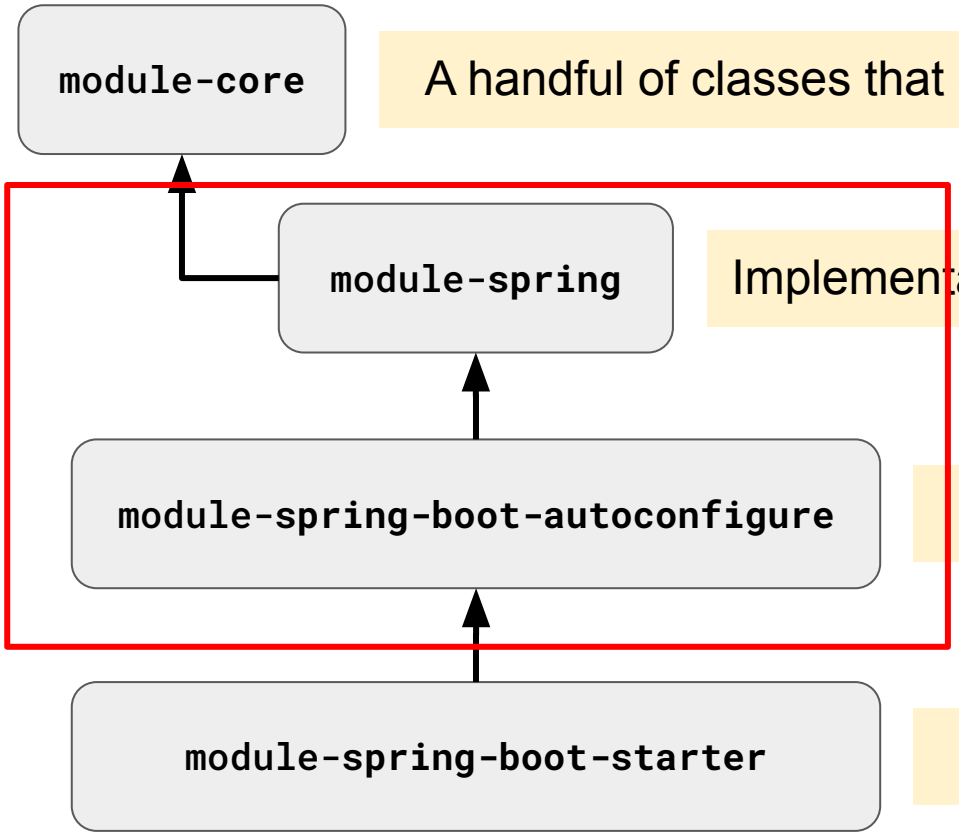
Default plugins to run: gmavenplus and checkstyle

Dependencies with their versions

Default dependencies like logging

```
<properties>
  <!-- default environment properties -->
  <gmavenplus-plugin.version>4.1.1</gmavenplus-plugin.version>
  <checkstyle.version>10.21.4</checkstyle.version>
  <include>**/*Test.java</include>
    <version>${springdoc-openapi.version}</version>
```

Module level: concept



A handful of classes that don’t depend on any specific framework

Implementations with the help of Spring Framework

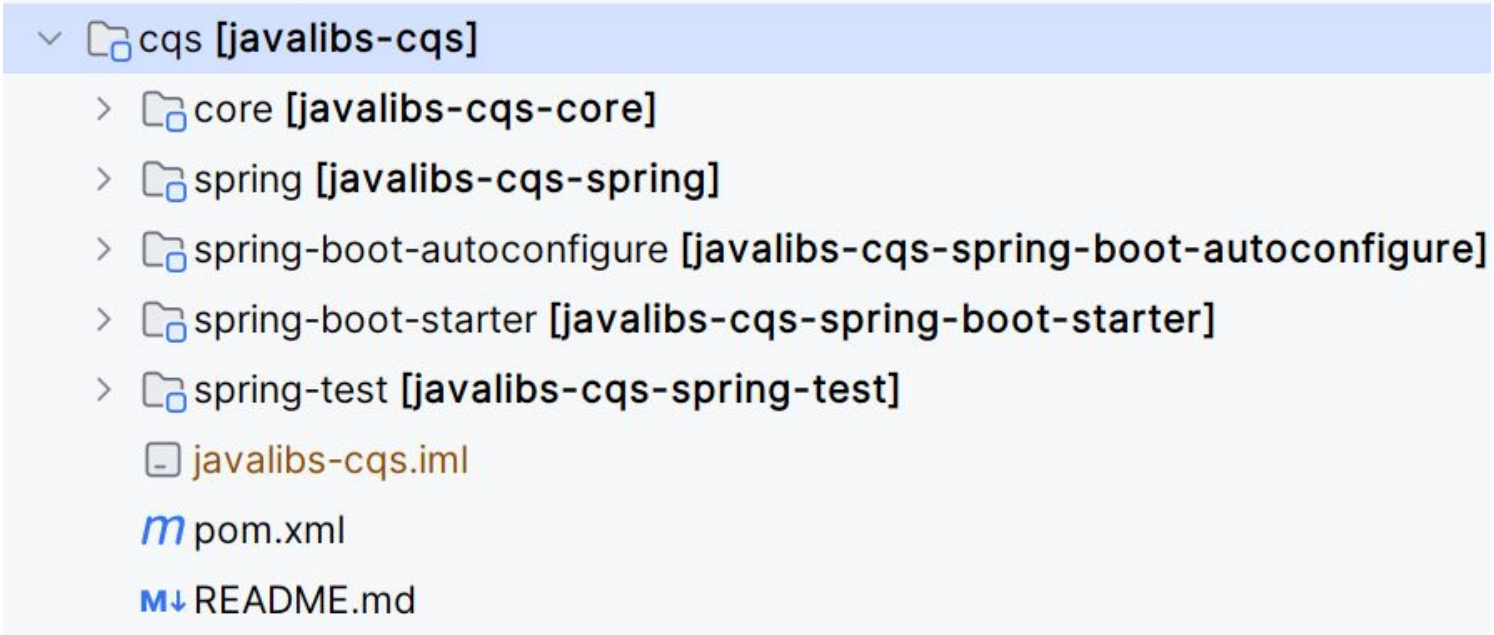
This can be one module

Autoconfigurations and properties

Default set of dependencies

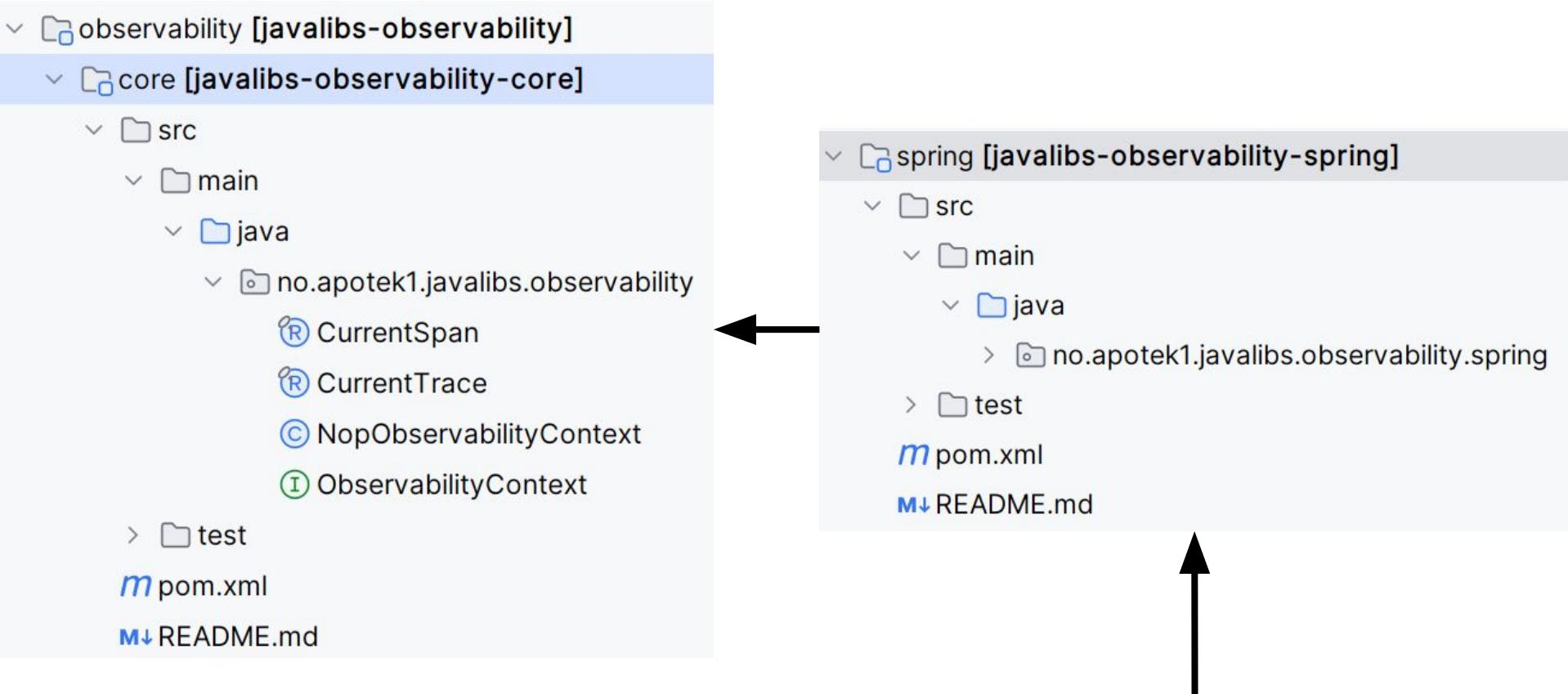
trainitek

Module level: example



trainitek

-core and -spring

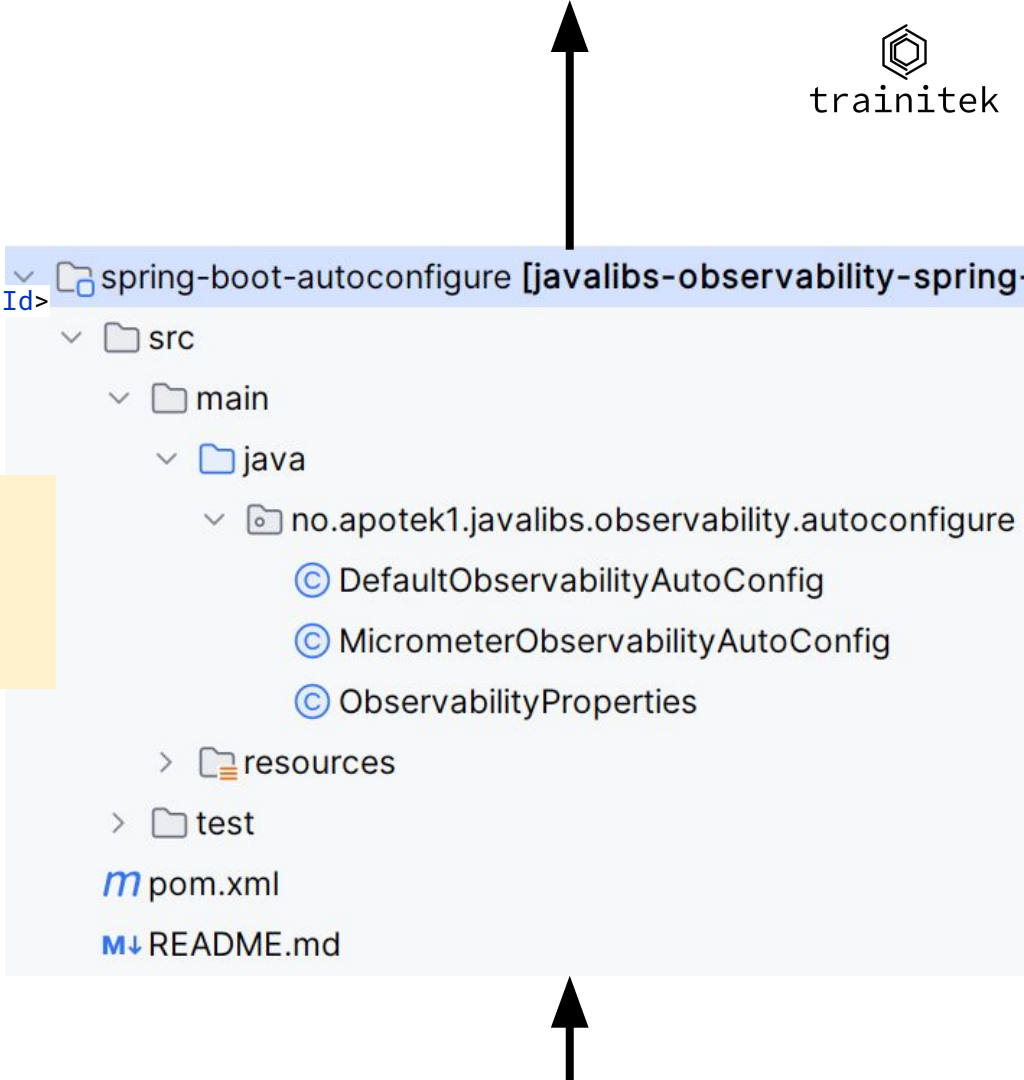


trainitek

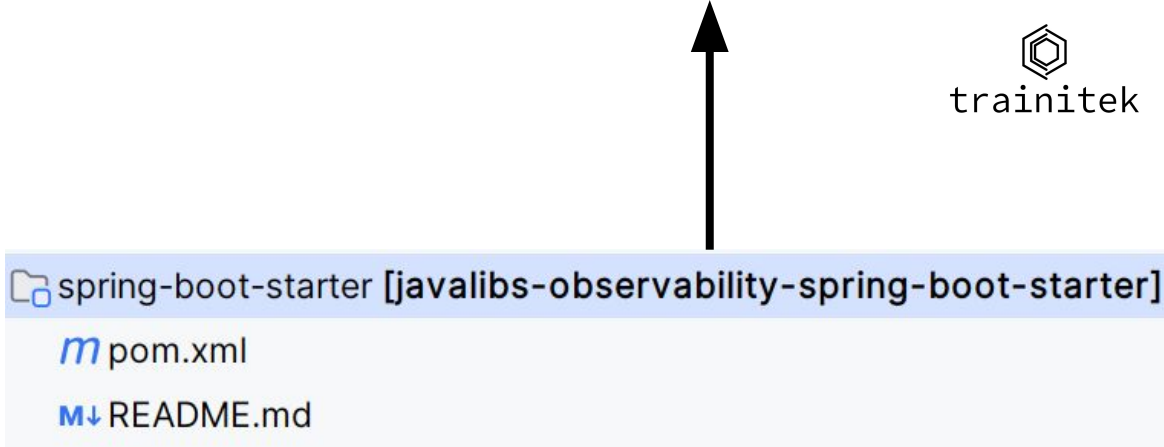
-spring-boot-autoconfigure

```
<dependency>
    <groupId>no.apotek1.javalibs</groupId>
    <artifactId>javalibs-observability-spring</artifactId>
    <version>${project.version}</version>
</dependency>
<dependency>
    <groupId>no.apotek1.javalibs</groupId>
    <artifactId>javalibs-test-core</artifactId>
    <v
    <s
</depe
<depen
    <g
    <a
    <v
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-tracing</artifactId>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-autoconfigure</artifactId>
</dependency>
```

Optional dependencies prevent clients from depending on unnecessary implementations!



-spring-boot-starter




```xml
<dependency>
   <groupId>no.apotek1.javalibs</groupId>
   <artifactId>javalibs-observability-spring-boot-autoconfigure</artifactId>
   <version>${project.version}</version>
</dependency>
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
   <groupId>io.micrometer</groupId>
   <artifactId>micrometer-tracing</artifactId>
</dependency>
<dependency>
   <groupId>io.micrometer</groupId>
   <artifactId>micrometer-tracing-bridge-otel</artifactId>
</dependency>
```

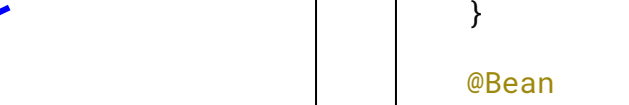
Auto Configurations

```
@AutoConfiguration
@ConditionalOnClass(Tracer.class)
@EnableConfigurationProperties(ObservabilityProperties.class)
public class MicrometerObservabilityAutoConfig {

    @Bean
    @ConditionalOnMissingBean
    ObservabilityContext micrometerObservabilityContext(Tracer tracer) {
        return new MicrometerObservabilityContext(tracer);
    }
}
```

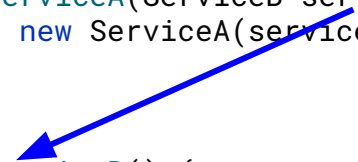
trainitek

@Configuration and @AutoConfiguration

```java
@Configuration
public class MyConfig {

    @Bean
    ServiceA serviceA() {
        return new ServiceA(serviceB());
    }

    @Bean
    ServiceB serviceB() {
        return new ServiceB();
    }
}
```



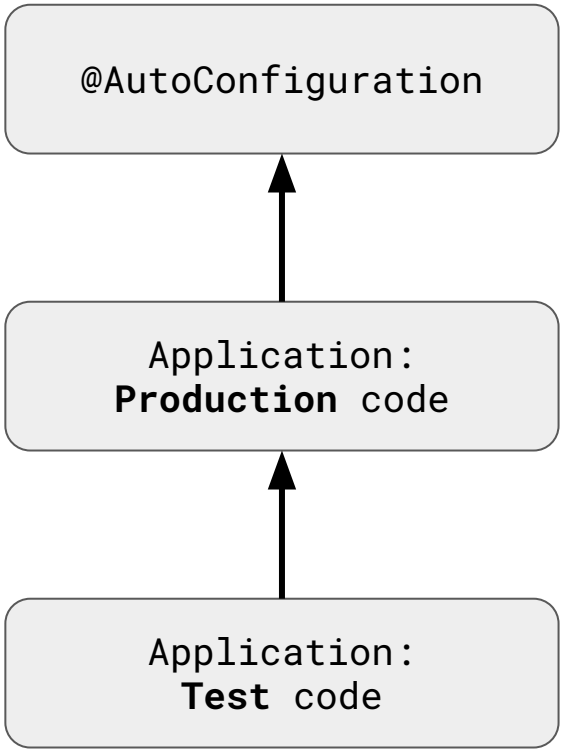
```java
@AutoConfiguration
public class MyAutoConfig {

    @Bean
    ServiceA serviceA(ServiceB serviceB) {
        return new ServiceA(serviceB);
    }

    @Bean
    ServiceB serviceB() {
        return new ServiceB();
    }
}
```



Auto Configurations

```
@AutoConfiguration
@ConditionalOnClass(Tracer.class)
@EnableConfigurationProperties(ObservabilityProperties.class)
public class MicrometerObservabilityAutoConfig {

    @Bean
    @ConditionalOnMissingBean
    ObservabilityContext micrometerObservabilityContext(Tracer tracer) {
        return new MicrometerObservabilityContext(tracer);
    }
}
```

Overriding beans



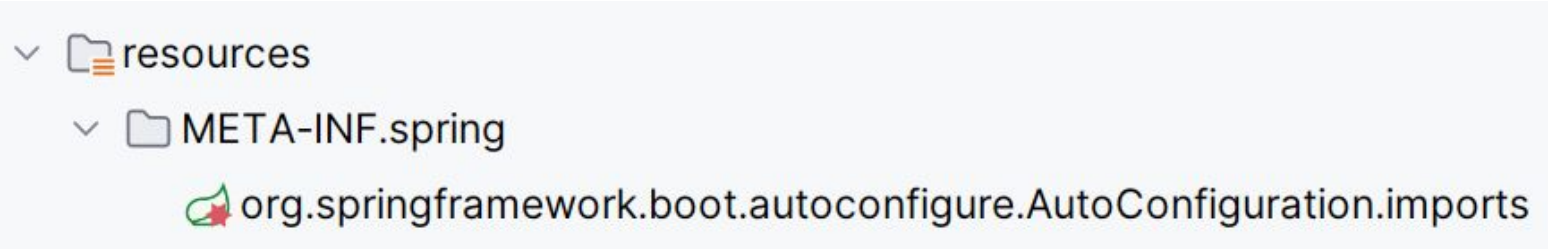
Default bean instances

Custom bean instances.
Just define them.

Testing bean instances.
Use `@Primary` annotation.

Auto Configurations

- Make sure your auto configuration runs **after** ordinary configurations!



@Fallback ?

```
@Configuration
public class MovieConfiguration {

    @Bean
    public MovieCatalog firstMovieCatalog() { ... }

    @Bean
    @Fallback
    public MovieCatalog secondMovieCatalog() { ... }
```

- `@Fallback` should be used by your application e.g. when you have a multi-module setup. It's idiomatically incorrect to use it in auto configurations.

@TestBean ?

```
class OverrideBeanTests {
    @TestBean
    CustomService customService;

    // test case body...

    static CustomService customService() {
        return new MyFakeCustomService();
    }
}
```

- @TestBean is useful when you need to redefine a bean for a particular test class.

Code quality

- To reduce maintenance costs, we need to delivery code of good quality. Ideally, automatically verified.

- At Apotek 1, we use the following tools:
  - **EditorConfig** - for code formatting
  - **Checkstyle** and **Error Prone** - to adhere to coding standards
  - **ArchUnit** - to spot barely visible issues and for knowledge sharing
  - **Snyk** - for automated vulnerability scans

trainitek

ArchUnit example: shared library

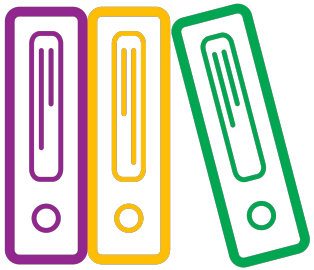
```
@ArchTest
 @ArchTest
public static final ArchRule autoconfiguredBeansAreAnnotatedWithAnyConditionalAnnotation =
 public static final ArchRule controllersAreAnnotatedWithOpenAPIAnnotations =
        methods().that().areAnnotatedWith(Bean.class)
         controllers().should().beAnnotatedWith(Tag.class)
            and().areDeclaredInClassesThat().areMetaAnnotatedWith(AutoConfiguration.class)
         .as("Controllers should be annotated with Open API annotations");
           .should().beMetaAnnotatedWith(Conditional.class)

           .as("Auto-configuration beans should be annotated with any                age) {
    return slic @Conditional annotation")+ ".(**)")

           .because("Spring Framework should not create those beans

                    if an application already declared beans of the same type");
            .as("Code is free of package cycles")
            .because("package cycles is an indicator of problems with

                      code design related to modularity and dependency direction.

                      Read more at https://our_wiki/…");

 }
```

trainitek

ArchUnit example: client project

```
class ArchitectureTest {

   @ArchTest
   public static final ArchTests standardTests = ArchTests.in(StandardArchTests.class);

   @ArchTest
   public static final ArchRule codeIsFreeOfCycles =
               codeIsFreeOfPackageCyclesRule(ArchitectureTest.class.getPackageName());

   @ArchTest
   public static final ArchTests springTests = ArchTests.in(SpringFrameworkArchTests.class);

   @ArchTest
   public static final ArchTests dddTests = ArchTests.in(DDDArchTests.class);

   @ArchTest
   public static final ArchTests cqsTests = ArchTests.in(CQSArchTests.class);

   @ArchTest
   public static final ArchTests webTests = ArchTests.in(WebArchTests.class);
}
```

Documentation



Make sure people can find it

- README files
- Confluence
- etc.

DRY?

- The DRY principle doesn’t work that well with documentation
- Sometimes, it’s worth having duplicated information, cross-links, etc. in multiple places
- This is a price to pay for your convenience

trainitek

README.md

- Make sure people can navigate through multiple README files by using **links**
- Imagine that your documentation is read by **junior** developers
- Supply your documentation with **examples** and **guidelines**

Modules

> Check `README.md` files in each submodule for more information.

test

Common dependencies and classes that help to write tests

security

Security related classes for authentication and authorization

web

Components that help to write REST API and test it

Test

This module contains code that helps to simplify writing tests.

Usage

Maven Dependencies

In most cases, you just need to include the javalibs-test-spring-boot-starter module to a list of your dependencies.

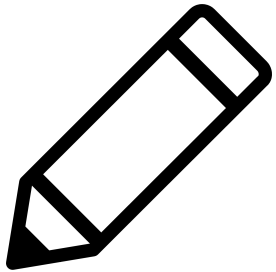
Database Support

Transaction in Tests

If you are writing a test that needs a transaction then you can use the `TransactionSupport` trait:

```
class MySpec extends Specification implements TransactionSupport {
    ...

    def "test"() {
        ...
        def result = tx { doSomethingInTransaction() }
        ...
    }
}
```

Making changes



General tips

- Making changes should be straightforward for contributors
- Avoid long and complicated routines
- Fix the guidelines if people can’t follow them
- If it happens, it’s your mistake, not theirs

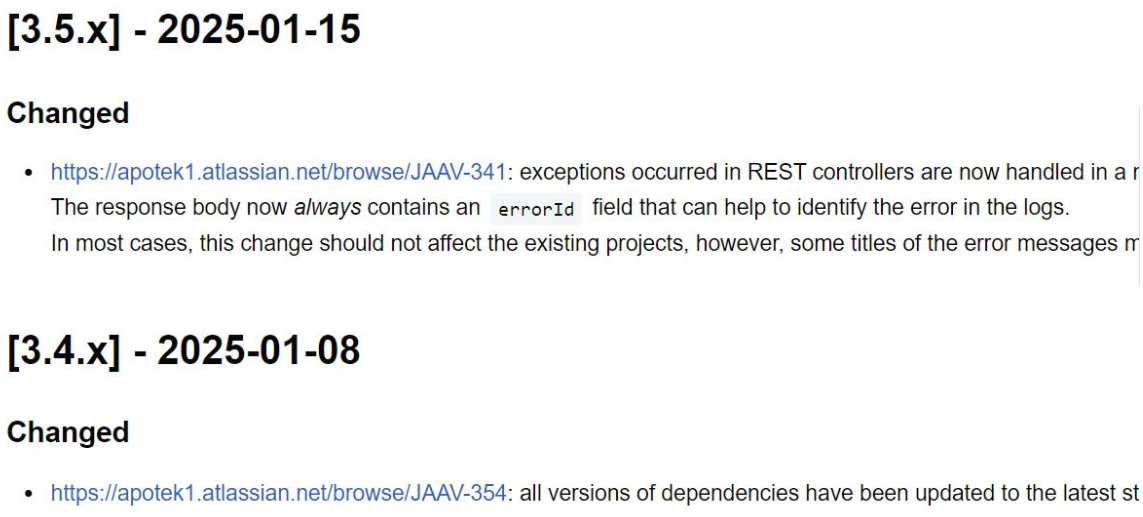
Versioning (3.7.10893)

- **Major** - for incompatible changes like:
  - JDK 17 -> JDK 21
  - Upgrading major versions of Spring Framework
  - etc.
- **Minor** - ALSO for changes where some migration is required.
  - We chose this because we don’t want to have too high major version (e.g. 34).
  - However, it doesn’t mean that you should do the same.
- **Patch** - this version is automatically updated by the CI/CD pipeline and is set to the build ID

trainitek

Documenting the changes

- We decided to use a **CHANGELOG** file - something simple that works for us
- https://keepachangelog.com/en/1.0.0/



[3.5.x] - 2025-01-15

Changed

- https://apotek1.atlassian.net/browse/JAAV-341: exceptions occurred in REST controllers are now handled in a r
  The response body now *always* contains an `errorId` field that can help to identify the error in the logs.
  In most cases, this change should not affect the existing projects, however, some titles of the error messages m

[3.4.x] - 2025-01-08

Changed

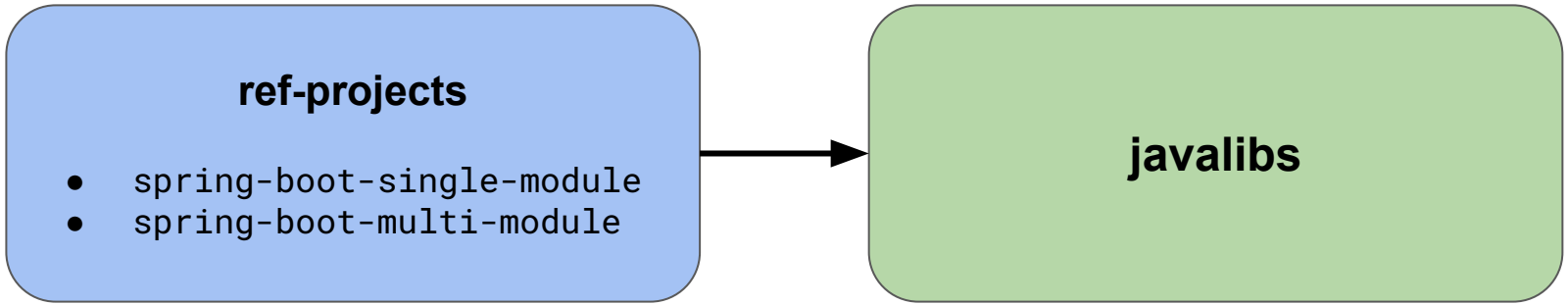
- https://apotek1.atlassian.net/browse/JAAV-354: all versions of dependencies have been updated to the latest st

Supporting old versions

- Our shared library doesn’t support multiple major version 🤔
- Instead, we chose to constantly upgrade our systems and depend on the latest version of the library 💡
- **Result**: simplicity at the given capacity 🚀

The change is done! Just ship it?

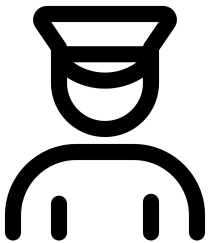
- A successful build is not enough
- It’s smart to create an example of a client project that can help verify the changes
- In our case, we call it “reference projects”

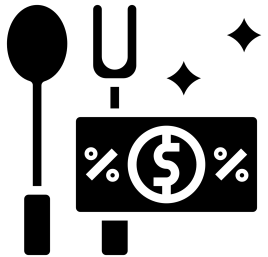


Reproducible builds?

```
<properties>
    <javalibs.version>[3.7,3.8)</javalibs.version>
</properties>

  …

<dependency>
    <groupId>no.apotek1.javalibs</groupId>
    <artifactId>javalibs-cqs-spring-boot-starter</artifactId>
    <version>${javalibs.version}</version>
</dependency>
```

Forcing vs Advertising





Forcing vs Advertising

- Prefer to advertise common libraries rather than forcing people to use them ✅
- This is the way to verify the quality and demand for the shared code 🟡
- Otherwise, there is a chance of living inside the illusion of a good job done 🧠

Conclusions

- (1/6) Make sure you need a shared library
- (2/6) Don't include anything unnecessary
- (3/6) Create a stable and understandable structure
- (4/6) Help people navigate and contribute via well-written documentation
- (5/6) Choose the easiest versioning strategy
- (6/6) If people want to use it, you have done a good job

trainitek

Thank You!

Presentation Slides

vadim-prudnikov