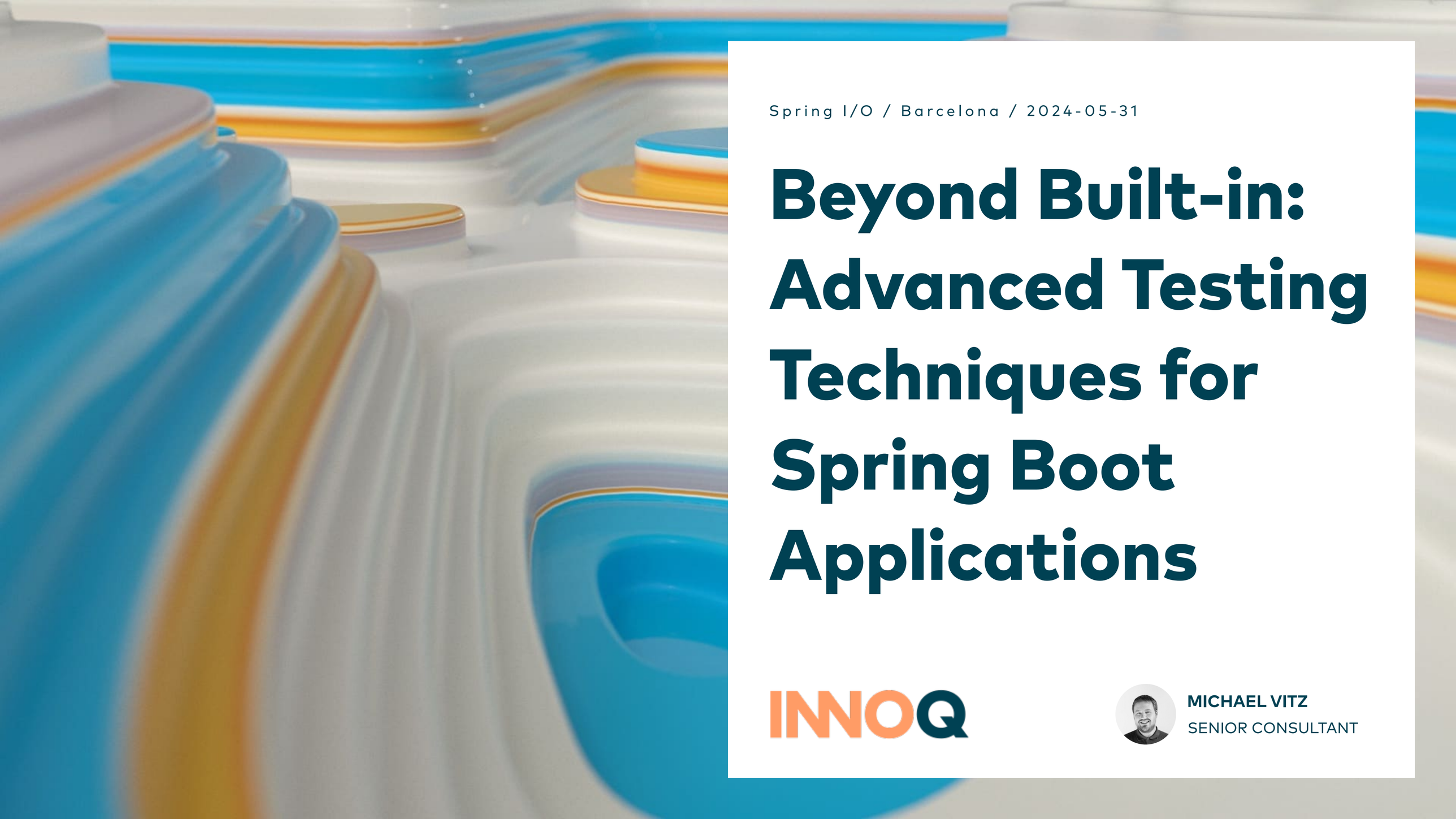


Spring I/O / Barcelona / 2024-05-31

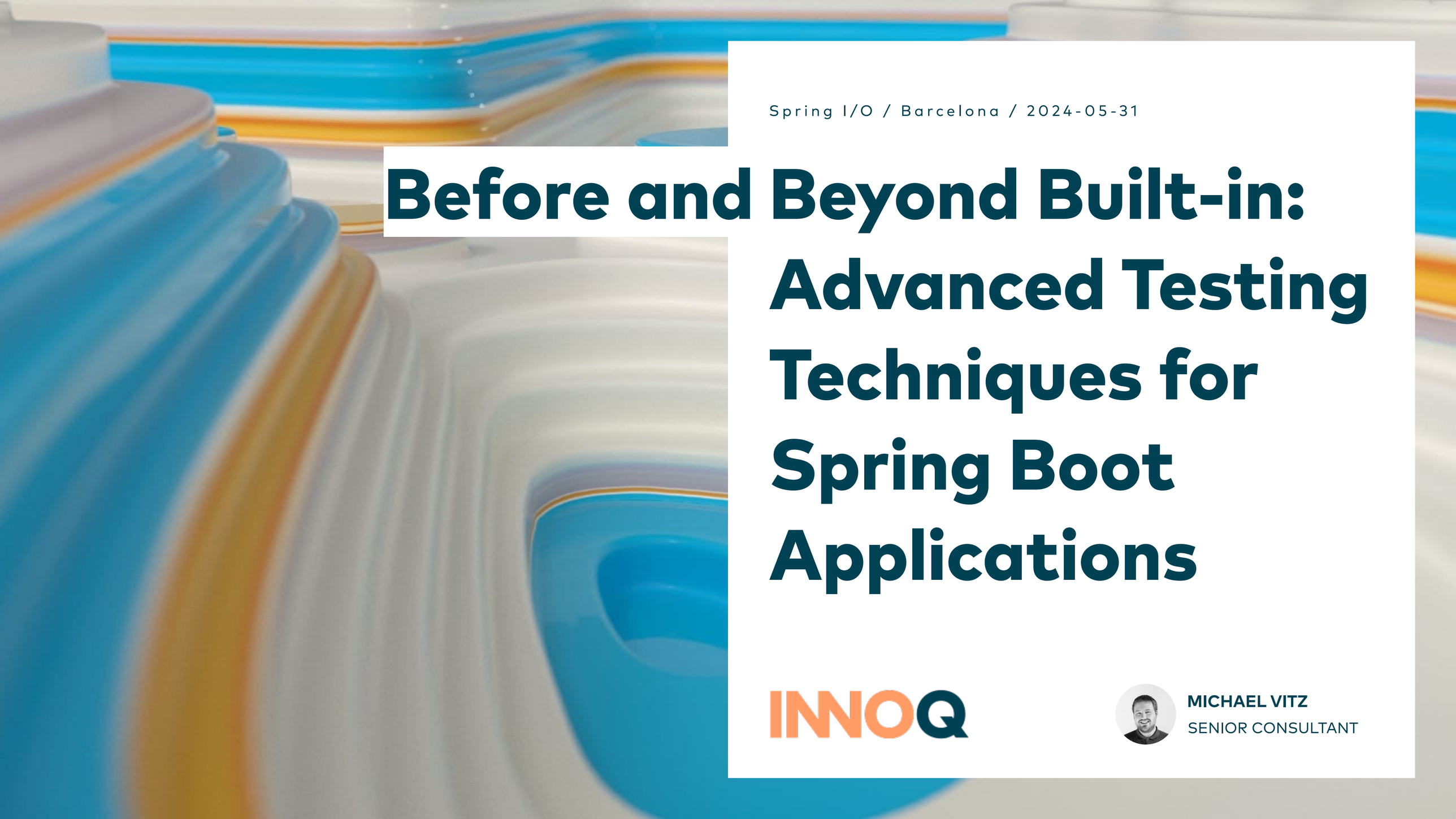
Beyond Built-in: Advanced Testing Techniques for Spring Boot Applications

INNOQ

MICHAEL VITZ
SENIOR CONSULTANT

MICHAEL VITZ

Java Champion
Senior Consultant at INNOQ

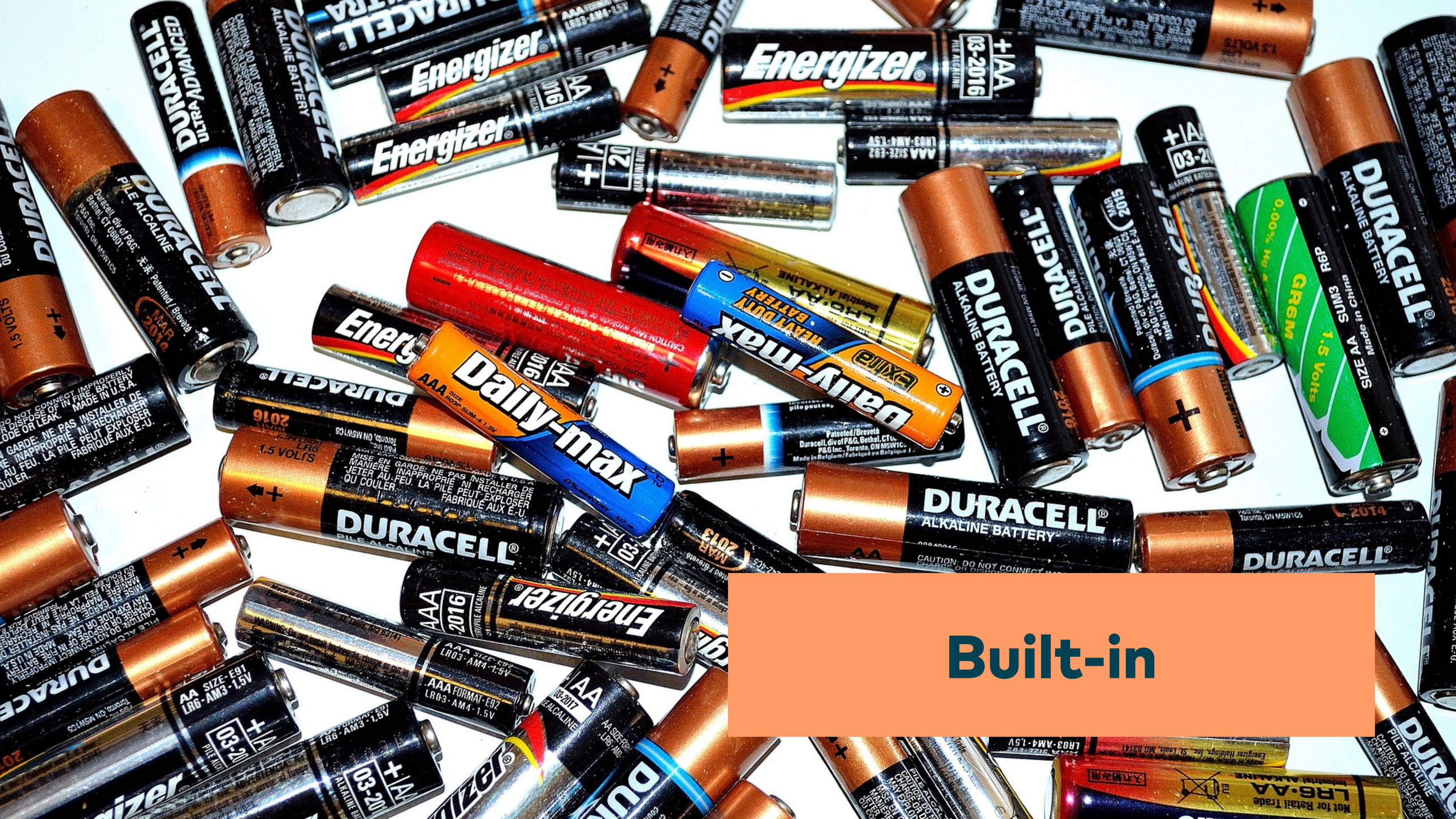


Spring I/O / Barcelona / 2024-05-31

Before and Beyond Built-in: Advanced Testing Techniques for Spring Boot Applications

INNOQ

MICHAEL VITZ
SENIOR CONSULTANT


Built-in

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

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
</dependency>

<dependency>
    <groupId>org.assertj</groupId>
    <artifactId>assertj-core</artifactId>
</dependency>
<dependency>
    <groupId>org.hamcrest</groupId>
    <artifactId>hamcrest</artifactId>
</dependency>

<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
</dependency>

<dependency>
    <groupId>org.awaitility</groupId>
    <artifactId>awaitility</artifactId>
</dependency>
```

```
<dependency>
    <groupId>com.jayway.jsonpath</groupId>
    <artifactId>json-path</artifactId>
</dependency>
<dependency>
    <groupId>org.skyscreamer</groupId>
    <artifactId>jsonassert</artifactId>
</dependency>

<dependency>
    <groupId>org.xmlunit</groupId>
    <artifactId>xmlunit-core</artifactId>
</dependency>
```

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

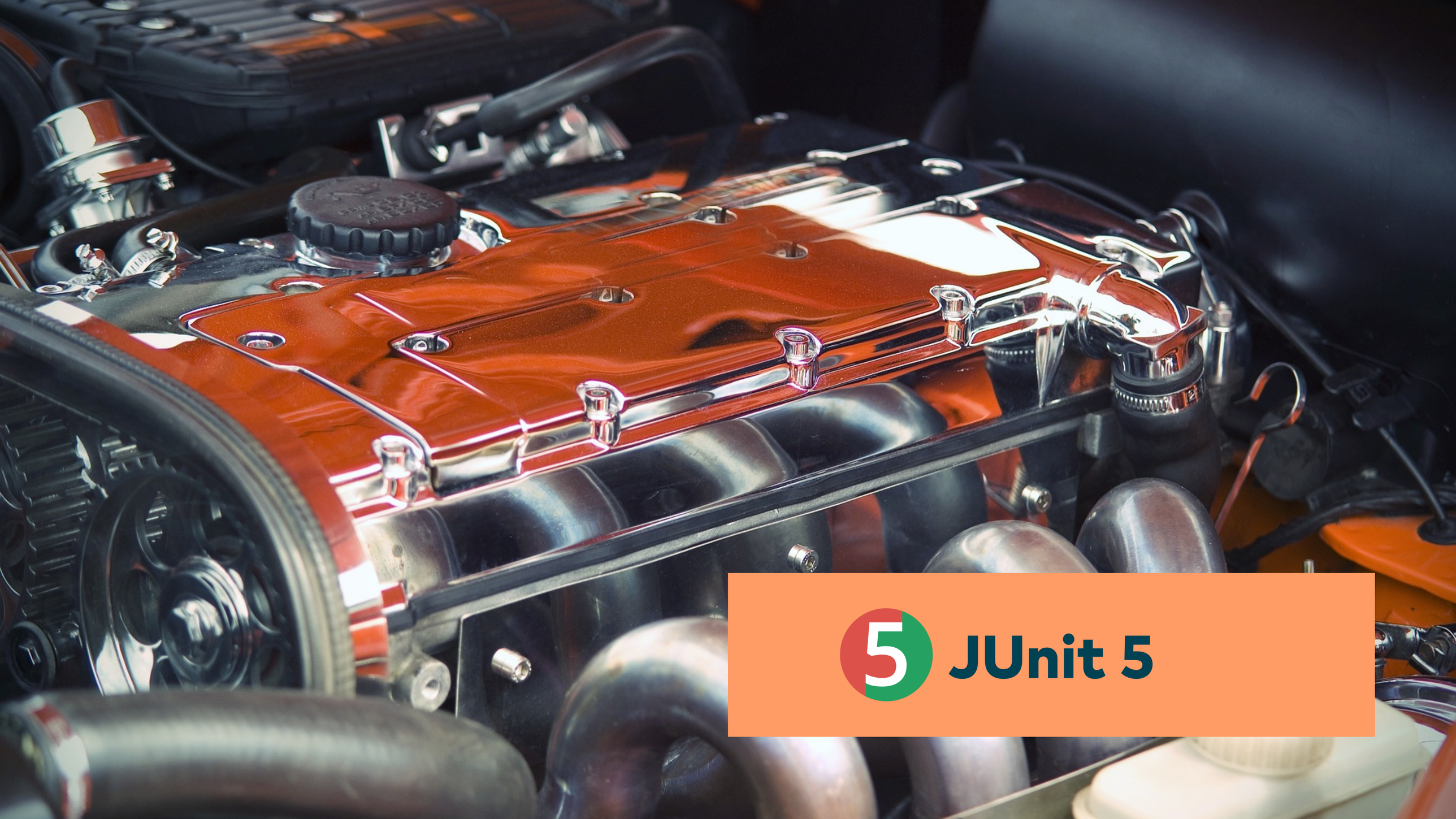
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
</dependency>
```

Read the reference documentation!

Framework: https://docs.spring.io/spring-framework/reference/testing.html

Boot: https://docs.spring.io/spring-boot/reference/testing/

Before



JUnit 5

Instance Per Test

```
class SomeTest {

    Person adultPerson;

    @BeforeEach
    void setUp() {
        adultPerson = new Person(42);
    }

    // ...
}
```

Instance Per Test

```
class SomeTest {

    Person adultPerson = new Person(42);




    // ...
}
```

Learn the basics!

JUnit 5: https://junit.org/junit5/docs/current/user-guide/

Parameterized Tests

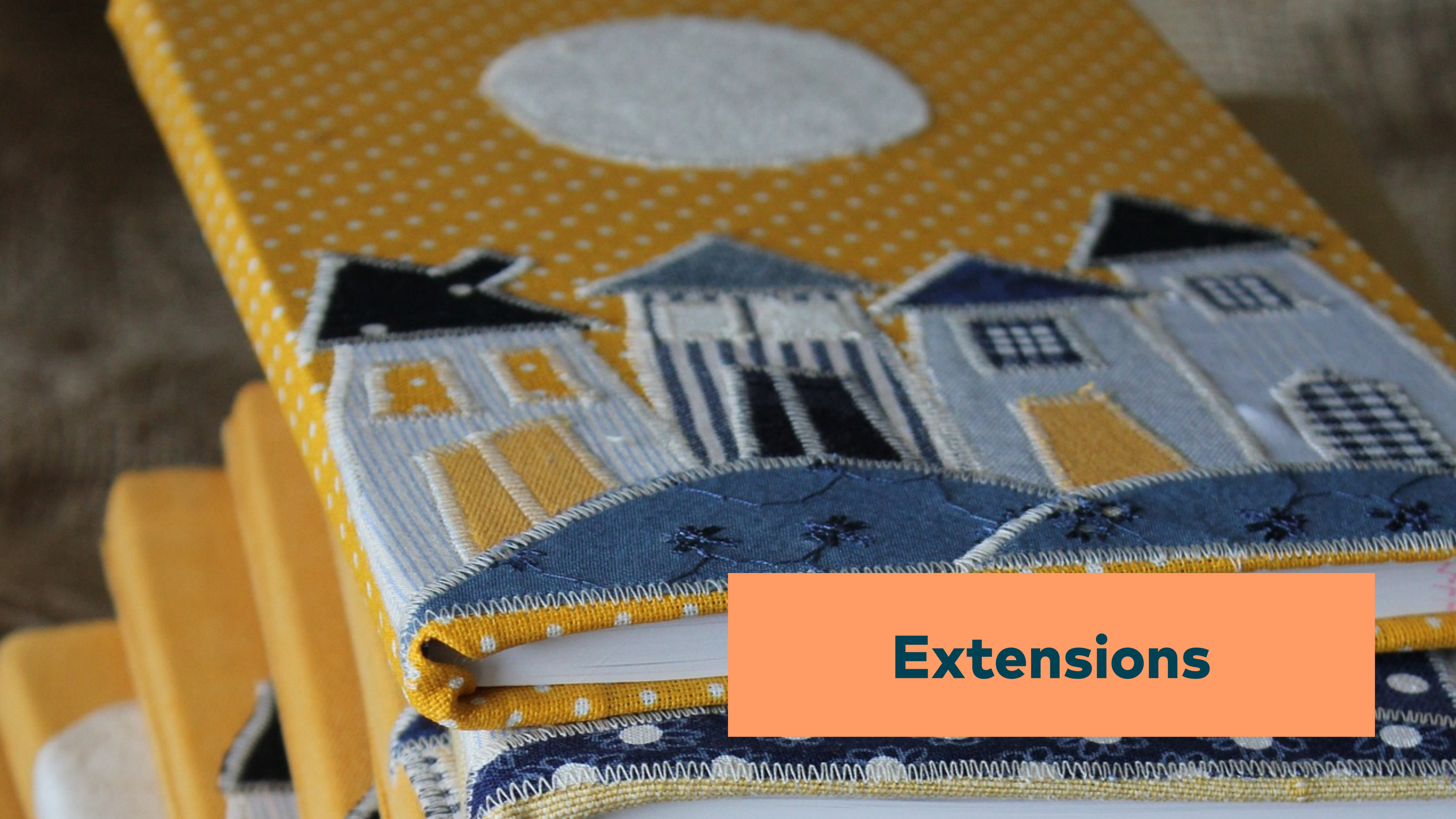
```
@ParameterizedTest
@ValueSource(ints = { 18, 21, 42, 77, 99, 122 })
void isAdult_shouldReturnTrue_whenPersonIsOverEighteen(int age) {
    // given
    var person = new Person();
    person.setAge(age);

    // when
    var isAdult = person.isAdult();

    // then
    assertTrue(isAdult);
}
```

https://junit.org/junit5/docs/current/user-guide/#writing-tests-parameterized-tests

Parameterized Tests

```
@ParameterizedTest
@MethodSource("isAdultWithCountryExamples")
void isAdult_shouldWork_whenPersonIsAdultInGivenCountry(String country, int age, boolean shouldBeAdult) {
    // given
    var person = new Person();
    person.setCountry(country);
    person.setAge(age);

    // when
    var isAdult = person.isAdult();

    // then
    assertEquals(shouldBeAdult, isAdult);
}

static Stream<Arguments> isAdultWithCountryExamples() {
    return Stream.of(
        arguments("US", 18, false),
        arguments("US", 21, true),
        arguments("DE", 18, true));
}
```

https://junit.org/junit5/docs/current/user-guide/#writing-tests-parameterized-tests



Extensions

Extensions

```
class SomeService {

    private static final Logger LOGGER = LoggerFactory.getLogger(SomeService.class);

    void doSomething(int tries) {
        try {
            // ...
        } catch (Exception e) {
            if (tries > 2) {
                // ...
                LOGGER.error("Giving up");
            } else {
                // ...
                LOGGER.warn("Failure, trying again later");
            }
        }
    }
}
```

Extensions

```
class SomeServiceTest {

    @Test
    void doSomething_shouldLogError_whenMoreThanTwoTries(
            @Logging LoggingEvents events) {
        // when
        someService.doSomething(3);

        // then
        assertThat(events.all())
            .isNotEmpty()
            .extracting(ILoggingEvent::getFormattedMessage)
            .containsExactly("Giving up");
    }
}
```

Extensions

```
@Target({PARAMETER, TYPE})
@Retention(RUNTIME)
@ExtendWith(LoggingExtension.class)
@Inherited
public @interface Logging {
}
```

```
public final class LoggingEvents {

    private final ListAppender<ILoggingEvent> appender;

    LoggingEvents(ListAppender<ILoggingEvent> appender) {
        this.appender = appender;
    }

    public boolean isEmpty() {
        return all().isEmpty();
    }

    public List<ILoggingEvent> all() {
        return matching(event -> true);
    }

    public List<ILoggingEvent> withLevel(Level level) {
        return matching(event -> event.getLevel().equals(level));
    }

    private List<ILoggingEvent> matching(Predicate<ILoggingEvent>
        return new ArrayList<>(appender.list).stream()
            .filter(predicate)
            .collect(toList());
    }
```

Extensions

```
class LoggingExtension implements ParameterResolver,                                    {


    @Override
    public boolean supportsParameter(ParameterContext parameterContext, ExtensionContext extensionContext) {
        return parameterContext.getParameter().getType().isAssignableFrom(LoggingEvents.class);
    }

    @Override
    public Object resolveParameter(ParameterContext parameterContext, ExtensionContext extensionContext) {
        final Store store = getStore(extensionContext);
        final ListAppender<ILoggingEvent> appender = getAppender(store);
        return new LoggingEvents(appender);
    }

    private static Store getStore(ExtensionContext context) {
        return context.getStore(Namespace.create(LoggingExtension.class, context.getRequiredTestMethod()));
    }

    private static ListAppender<ILoggingEvent> getAppender(Store store) {
        return store.get(APPENDER, CloseableAppender.class).appender;
    }
```

Extensions

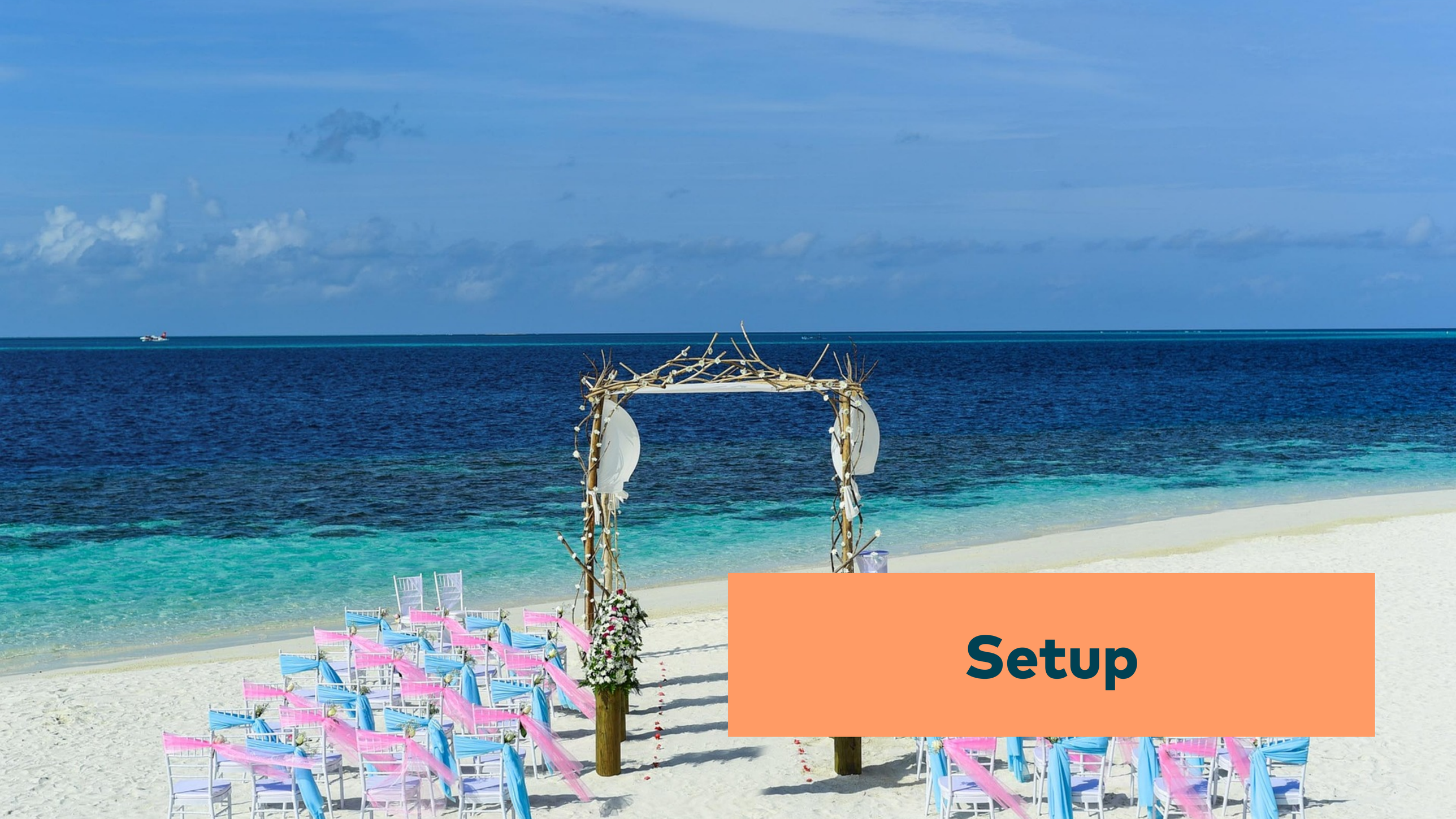
```
class LoggingExtension implements ParameterResolver, BeforeTestExecutionCallback {

    // ...

    @Override
    public void beforeTestExecution(ExtensionContext extensionContext) {
        final Store store = getStore(extensionContext);

        final ListAppender<ILoggingEvent> appender = new ListAppender<>();

        storeLogger(store, CloseableLogger.from(
            LoggerFactory.getLogger(ROOT_LOGGER_NAME), org.slf4j.event.Level.INFO, appender));

        storeAppender(store, appender);
    }

    private static void storeLogger(Store store, CloseableLogger logger) {
        store.put(logger.getName(), logger);
    }

    private static void storeAppender(Store store, ListAppender<ILoggingEvent> appender) {
        store.put(APPENDER, new CloseableAppender(appender));
    }
```

Extensions

- junit5-logging-extension: https://github.com/innoq/junit5-logging-extension
- Log Collectors: https://github.com/haasted/TestLogCollectors
- SpringExtension: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/test/context/junit/jupiter/SpringExtension.html
- OutputCaptureExtension: https://docs.spring.io/spring-boot/api/java/org/springframework/boot/test/system/OutputCaptureExtension.html
- ...

https://junit.org/junit5/docs/current/user-guide/#extensions

Know some advanced features!



Setup

Setup

```
@Test
void total_shouldIncludeDiscount_whenPersonIsAdult() {
    // given
    var adult = new Person();
    adult.setAge(18);


    // when
    // ...

    // then
    // ...
}
```

Setup

```
@Test
void total_shouldIncludeDiscount_whenPersonIsAdult() {
    // given
    var adult = new Person("NameDoesNotMatter", 18);



    // when
    // ...

    // then
    // ...
}
```

Setup

```
@Test
void total_shouldIncludeDiscount_whenPersonIsAdult() {
    // given
    var adult = Persons.MICHAEL; // or Persons.michael();



    // when
    // ...

    // then
    // ...
}
```

Setup

```
@Test
void total_shouldIncludeDiscount_whenPersonIsAdult() {
    // given
    var adult = Persons.aPerson()
        .withAge(18)
        .build();

    // when
    // ...

    // then
    // ...
}
```

Setup

```
@Test
void total_shouldIncludeDiscount_whenPersonIsAdult() {
    // given
    var adult = Persons.aPerson()
        .olderThan(18)
        .build();

    // when
    // ...

    // then
    // ...
}
```

Setup

```
@Test
void total_shouldIncludeDiscount_whenPersonIsAdult() {
    // given
    var adult = Persons.aPerson()
        .thatsAnAdult()
        .build();

    // when
    // ...

    // then
    // ...
}
```

Setup

```
class Persons {


    static Persons.Builder aPerson() {
        return new Builder();
    }

    static class Builder {

        Person person = new Person(
            RandomStringUtils.randomAlphabetic(2, 20),
            RandomUtils.nextInt(0, 120));


        public Builder withAge(int age) {
            person.setAge(age);
            return this;
        }

        // ...
```

commons-lang3: https://commons.apache.org/proper/commons-lang/

Setup

```
class Persons {

    static Faker FAKER = new Faker(Locale.of("es"), new Random(42));

    static Persons.Builder aPerson() {
        return new Builder();
    }

    static class Builder {

        Person person = new Person(
            FAKER.name().fullName(),
            FAKER.number().numberBetween(0, 120));


        public Builder withAge(int age) {
            person.setAge(age);
            return this;
        }

        // ...
```

data-faker: https://github.com/datafaker-net/datafaker

Setup

```
class Persons {


    static Persons.Builder aPerson() {
        return new Builder();
    }

    static class Builder {

        Person person = Instancio.of(Person.class)
            .generate(Select.field(Person::getAge), gen -> gen.ints().min(0).max(120))
            .generate(Select.field(Person::getName), gen -> gen.string().alphaNumeric())
            .create();

        public Builder withAge(int age) {
            person.setAge(age);
            return this;
        }

        // ...
```

Instancio: https://github.com/instancio/instancio

Setup

```
class Persons {

    static final Property<Person, String> name = Property.newProperty();
    static final Property<Person, Integer> age = Property.newProperty();

    static final Instantiator<Person> Person = lookup -> new Person(
        lookup.valueOf(name, "Michael"),
        lookup.valueOf(age, 42));
}
```

make-it-easy: https://github.com/npryce/make-it-easy

Setup

```
@Test
void total_shouldIncludeDiscount_whenPersonIsAdult() {
    // given
    var adult = an(Person,
        with(42, age))
        .make();

    // when
    // ...

    // then
    // ...
}
```

Keep test code reasonable

Assertions

Assertions

```
@Test
void constructor_shouldSetNameAndAge() {
    // when
    var michael = new Person("Michael", 38);

    // then
    assertThat(michael.getName()).isEqualTo("Michael");
    assertThat(michael.getAge()).isEqualTo(38);
    assertThat(michael.isAdult()).isTrue();
}
```

Assertions

```
@Test
void constructor_shouldSetNameAndAge() {
    // when
    var michael = new Person("Michael", 38);

    // then
    assertThat(michael)
        .extracting(Person::getName, Person::getAge, Person::isAdult)
        .containsExactly("Michael", 38, true);
}
```

Assertions

```
@Test
void constructor_shouldSetNameAndAge() {
    // when
    var michael = new Person("Michael", 38);

    // then
    assertThat(michael)
        .hasName("Michael")
        .hasAge(38)
        .isAdult();
}
```

AssertJ: https://assertj.github.io/doc/#assertj-core-custom-assertions

Hamcrest: https://hamcrest.org/JavaHamcrest/tutorial#writing-custom-matchers

Keep test code readable

equals / hashCode

```
class Person {

    // ...

    @Override
    public int hashCode() {
        // ???
    }

    @Override
    public boolean equals(Object obj) {
        // ???
    }
}
```

Reflexive `x.equals(x) // -> true`

Symmetric `x.equals(y) // -> y.equals(x)`

Transitive `x.equals(y) && y.equals(z) // -> x.equals(z)`

Consistent `x.equals(y) // -> x.equals(y)`

equals / hashCode

```
@Test
 void equalsAndHashCode_shouldFulfillContract() {
    EqualsVerifier.forClass(Person.class).verify();
}
```

EqualsVerifier: https://jqno.nl/equalsverifier/

Keep test code maintainable

Beyond

Spring Boot Testing

Spring Boot Testing

- @SpringBootTest
- Loads everything into Application Context
- Can be “slow”, although contexts are cached
- Allows testing the whole application

https://docs.spring.io/spring-boot/reference/testing/spring-boot-applications.html

Maybe you don't need your whole application context

Test Slices

- @WebMvcTest, @DataJpaTest, ...
- Loads only relevant subset into Application Context
- Faster than @SpringBootTest
- Integration testing isolated parts

https://docs.spring.io/spring-boot/reference/testing/spring-boot-applications.html#testing.spring-boot-applications.autoconfigured-tests

Maybe you don't even need a slice of your application context

Extend

Extend

```
@RestController
public class TimeController {

    static DateTimeFormatter DATE_TIME_FORMAT =
        DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

    @GetMapping("/time")
    public String now() {
        LocalDateTime now = LocalDateTime.now();
        return now.format(DATE_TIME_FORMAT);
    }
}
```

Extend

```
@WebMvcTest

class TimeControllerTest {

    @Autowired
    MockMvc mvc;

    @Test
    void now_shouldRenderCurrentTime() throws Exception {
        mvc.perform(get("/time"))
            .andExpect(status().isOk())
            .andExpect(content().string(is(equalTo("2024-05-31 17:42:53"))));
    }









}
```

Extend

```
@RestController
public class TimeController {

    static DateTimeFormatter DATE_TIME_FORMAT =
        DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

    private final Clock clock;

    public TimeController(Clock clock) {
        this.clock = clock;
    }

    @GetMapping("/time")
    public String now() {
        LocalDateTime now = LocalDateTime.now(clock);
        return now.format(DATE_TIME_FORMAT);
    }
}
```

Extend

```
@WebMvcTest
@TestPropertySource(properties = "spring.main.allow-bean-definition-overriding=true")
class TimeControllerTest {

    @Autowired
    MockMvc mvc;

    @Test
    void now_shouldRenderCurrentTime() throws Exception {
        mvc.perform(get("/time"))
            .andExpect(status().isOk())
            .andExpect(content().string(is(equalTo("2024-05-31 17:42:53"))));
    }

    @TestConfiguration
    static class TimeControllerTestConfiguration {

        @Bean
        public Clock clock() {
            LocalDateTime localDateTime = LocalDateTime.of(2024, 5, 31, 17, 42, 53);
            return Clock.fixed(localDateTime.toInstant(UTC), UTC);
        }
    }
}
```

Extend

```
@WebMvcTest
@WithLocalDateTime(date = "2024-05-31", time = "17:42:53")
class TimeControllerTest {

    @Autowired
    MockMvc mvc;

    @Test
    void now_shouldRenderCurrentTime() throws Exception {
        mvc.perform(get("/time"))
            .andExpect(status().isOk())
            .andExpect(content().string(is(equalTo("2024-05-31 17:42:53"))));
    }









}
```

Extend

```
@Target(TYPE)
@Retention(RUNTIME)
@Documented
@Inherited

@TestPropertySource(properties = "spring.main.allow-bean-definition-overriding=true")
@ImportAutoConfiguration(WithLocalDateTime.ClockConfiguration.class)
public @interface WithLocalDateTime {

    String date();
    String time();

    @TestConfiguration
    class ClockConfiguration {

        @Bean
        public Clock clock() {
            return new DelegatingClock(Clock.systemUTC());
        }
    }
```

Extend

```
@Target(TYPE)
@Retention(RUNTIME)
@Documented
@Inherited
@ExtendWith(WithLocalDateTime.WithLocalDateTimeExtension.class)
@TestPropertySource(properties = "spring.main.allow-bean-definition-overriding=true")
@ImportAutoConfiguration(WithLocalDateTime.ClockConfiguration.class)
public @interface WithLocalDateTime {

    // …

    class WithLocalDateTimeExtension implements BeforeEachCallback, AfterEachCallback {

        @Override
        public void beforeEach(ExtensionContext extensionContext) {
            findAnnotation(extensionContext.getTestClass(), WithLocalDateTime.class)
                .ifPresent((withLocalDateTime) -> setClockTo(extensionContext, withLocalDateTime));
        }

        @Override
        public void afterEach(ExtensionContext extensionContext) {
            findAnnotation(extensionContext.getTestClass(), WithLocalDateTime.class)
                .ifPresent((withLocalDateTime) -> resetClockFrom(extensionContext));
        }
```

Extend

```
@Target(TYPE)
@Retention(RUNTIME)
@Documented
@Inherited
@ExtendWith(WithLocalDateTime.WithLocalDateTimeExtension.class)
@TestPropertySource(properties = "spring.main.allow-bean-definition-overriding=true")
@ImportAutoConfiguration(WithLocalDateTime.ClockConfiguration.class)
public @interface WithLocalDateTime {

    // ...

    class WithLocalDateTimeExtension implements BeforeEachCallback, AfterEachCallback {

        // ...

        private static DelegatingClock delegatingClockFrom(ExtensionContext context) {
            Clock clock = getApplicationContext(context).getBean(Clock.class);
            // ...
            return (DelegatingClock) clock;
        }

        private static Clock fixedClock(LocalDateTime localDateTime) {
            return Clock.fixed(localDateTime.atZone(UTC).toInstant(), UTC);
        }
```

Compose

Compose

```
ALTER TABLE person
    ADD COLUMN firstname VARCHAR,
    ADD COLUMN lastname VARCHAR;

UPDATE person p
SET
    firstname = split_part(p.name, ' ', 1),
    lastname = split_part(p.name, ' ', 2)
FROM person po
WHERE p.name = po.name;

ALTER TABLE person
    ALTER COLUMN firstname SET NOT NULL,
    ALTER COLUMN lastname SET NOT NULL,
    DROP COLUMN name;
```

Compose

```
@MigrationTest(fromVersion = 1, toVersion = 2)
class V2AddFirstAndLastnameToPersonTableTest {

    @Autowired
    JdbcTemplate jdbcTemplate;

    @Test
    void migration_shouldSplitNameIntoFirstAndLastname(MigrationTestTemplate template) {
        template.beforeMigration(() -> {
            jdbcTemplate.execute("TRUNCATE TABLE person");
            jdbcTemplate.execute("INSERT INTO person (name) VALUES ('Test Fixture')");
        });

        template.afterMigration(() -> {
            String person = jdbcTemplate.queryForObject(
                "SELECT * FROM person",
                (rs, rowNum) -> {
                    return rs.getString("lastname") + ", " + rs.getString("firstname");
                });
            assertThat(person)
                .isEqualTo("Fixture, Test");
        });
    }
}
```

Compose

```
@Target(TYPE)
@Retention(RUNTIME)
@Documented
@Inherited
@SpringBootTest
@ImportAutoConfiguration(exclude = FlywayAutoConfiguration.class)
@ExtendWith(FlywayMigrationTestExtension.class)
public @interface MigrationTest {

    int fromVersion();
    int toVersion();
}
```

Compose

```
class FlywayMigrationTestExtension implements BeforeEachCallback, AfterEachCallback, ParameterResolver {

    @Override
    public void beforeEach(ExtensionContext context) throws Exception {
        // drop all tables
    }

    @Override
    public void afterEach(ExtensionContext context) throws Exception {
        // drop all tables and reapply all migrations
    }
```

Compose

```
class FlywayMigrationTestExtension implements BeforeEachCallback, AfterEachCallback, ParameterResolver {

    @Override
    public boolean supportsParameter(ParameterContext parameterContext,
                                     ExtensionContext extensionContext) throws ParameterResolutionException {
        return MigrationTestTemplate.class.equals(parameterContext.getParameter().getType())
              && findAnnotation(extensionContext.getTestClass(), MigrationTest.class).isPresent();
    }

    @Override
    public Object resolveParameter(ParameterContext parameterContext,
                                   ExtensionContext extensionContext) throws ParameterResolutionException {
        return findAnnotation(extensionContext.getTestClass(), MigrationTest.class)
           .map((migrationTest) -> {
               DataSource dataSource = getApplicationContext(extensionContext).getBean(DataSource.class);
               int fromVersion = migrationTest.fromVersion();
               int toVersion = migrationTest.toVersion();

               return new MigrationTestTemplate(dataSource, fromVersion, toVersion);
           })
          .orElseThrow(() -> new IllegalStateException("..."));
    }
}
```

Compose

```
class MigrationTestTemplate {

    public void beforeMigration(Executable executable) {
        try {
            migrateUpTo(fromVersion);
            executable.execute();
        } catch (Throwable throwable) {
            throw new IllegalStateException("unable to execute pre-migration steps", throwable);
        }
    }

    public void afterMigration(Executable executable) {
        // same as above
    }

    private void migrateUpTo(int upToVersion) {
        Flyway.configure()
            .dataSource(dataSource)
            .locations("/db/migration")
            .target(valueOf(upToVersion))
            .load()
            .migrate();
    }
}
```

Create

Create

```
@PostgresRepositoryTest(GreetingTextRepository.class)
class GreetingTextRepositoryPostgresTest {

    @Autowired
    GreetingTextRepository greetingTextRepository;

    @Test
    void getDefaultGreeting_shouldReturnGreetingTextFromDatabase() {
        var greetingText = greetingTextRepository
            .getDefaultGreetingText();

        assertThat(greetingText)
            .isEqualTo("Hallo %s.");
    }
}
```

Create

```
@Target(TYPE)
@Retention(RUNTIME)
@Documented
@Inherited
```

```
public @interface PostgresRepositoryTest {

    @AliasFor("repositories")
    Class<?>[] value() default {};

    @AliasFor("value")
    Class<?>[] repositories() default {};
}
```

Create

```
@Target(TYPE)
@Retention(RUNTIME)
@Documented
@Inherited
@ExtendWith(SpringExtension.class)
```

```
public @interface PostgresRepositoryTest {

    @AliasFor("repositories")
    Class<?>[] value() default {};

    @AliasFor("value")
    Class<?>[] repositories() default {};
}
```

Create

```
@Target(TYPE)
@Retention(RUNTIME)
@Documented
@Inherited
@ExtendWith(SpringExtension.class)
@BootstrapWith(SpringBootTestContextBootstrapper.class)
@TypeExcludeFilters(PostgresRepositoryTypeExcludeFilter.class)
```

```
public @interface PostgresRepositoryTest {

    @AliasFor("repositories")
    Class<?>[] value() default {};

    @AliasFor("value")
    Class<?>[] repositories() default {};
}
```

Create

```
final class PostgresRepositoryTypeExcludeFilter extends
        StandardAnnotationCustomizableTypeExcludeFilter<PostgresRepositoryTest> {

    private static final Class<?>[] NO_REPOSITORIES = {};

    private final Class<?>[] repositories;

    PostgresRepositoryTypeExcludeFilter(Class<?> testClass) {
        super(testClass);
        this.repositories = getAnnotation()
            .getValue("repositories", Class[].class)
            .orElse(NO_REPOSITORIES);
    }

    @Override
    protected boolean isUseDefaultFilters() {
        return true;
    }
```

Create

```
final class PostgresRepositoryTypeExcludeFilter extends
        StandardAnnotationCustomizableTypeExcludeFilter<PostgresRepositoryTest> {

    private static final Set<Class<?>> DEFAULT_INCLUDES =
        Collections.emptySet();
    private static final Set<Class<?>> DEFAULT_INCLUDES_AND_REPOSITORY =
        Set.of(Repository.class);

    @Override
    protected Set<Class<?>> getDefaultIncludes() {
        if (ObjectUtils.isEmpty(this.repositories)) {
            return DEFAULT_INCLUDES_AND_REPOSITORY;
        }
        return DEFAULT_INCLUDES;
    }

    @Override
    protected Set<Class<?>> getComponentIncludes() {
        return Set.of(this.repositories);
    }
}
```

Create

```
@Target(TYPE)
@Retention(RUNTIME)
@Documented
@Inherited
@ExtendWith(SpringExtension.class)
@BootstrapWith(SpringBootTestContextBootstrapper.class)
@TypeExcludeFilters(PostgresRepositoryTypeExcludeFilter.class)
@ImportAutoConfiguration
@OverrideAutoConfiguration(enabled = false)


public @interface PostgresRepositoryTest {

    @AliasFor("repositories")
    Class<?>[] value() default {};

    @AliasFor("value")
    Class<?>[] repositories() default {};
}
```

Create

```
# RepositoryTest auto-configuration imports
org.springframework....flyway.FlywayAutoConfiguration
org.springframework....jdbc.DataSourceAutoConfiguration
org.springframework....jdbc.DataSourceTransactionManagerAutoConfiguration
org.springframework....jdbc.JdbcTemplateAutoConfiguration
org.springframework....transaction.TransactionAutoConfiguration
```

META-INF/spring/de.mvitz.spring.test.slices.PostgresRepositoryTests.imports

Create

```
@Target(TYPE)
@Retention(RUNTIME)
@Documented
@Inherited
@ExtendWith(SpringExtension.class)
@BootstrapWith(SpringBootTestContextBootstrapper.class)
@TypeExcludeFilters(PostgresRepositoryTypeExcludeFilter.class)
@ImportAutoConfiguration
@OverrideAutoConfiguration(enabled = false)
@ContextConfiguration(initializers = PostgresRepositoryTestInitializer.class)

public @interface PostgresRepositoryTest {

    @AliasFor("repositories")
    Class<?>[] value() default {};

    @AliasFor("value")
    Class<?>[] repositories() default {};
}
```

Create

```
final class PostgresRepositoryTestInitializer implements
        ApplicationContextInitializer<ConfigurableApplicationContext> {

    @Override
    public void initialize(ConfigurableApplicationContext context) {
        final var container = new PostgreSQLContainer<>("postgres:latest")
            .withUsername("test")
            .withPassword("test");

        context.addApplicationListener(
            (ApplicationListener<ContextClosedEvent>) event ->
                container.stop());

        container.start();

        TestPropertyValues.of(
                "spring.datasource.url=" + container.getJdbcUrl(),
                "spring.datasource.username=" + container.getUsername(),
                "spring.datasource.password=" + container.getPassword())
            .applyTo(context.getEnvironment());
    }
}
```

Create

```
@Target(TYPE)
@Retention(RUNTIME)
@Documented
@Inherited
@ExtendWith(SpringExtension.class)
@BootstrapWith(SpringBootTestContextBootstrapper.class)
@TypeExcludeFilters(PostgresRepositoryTypeExcludeFilter.class)
@ImportAutoConfiguration
@OverrideAutoConfiguration(enabled = false)
@ContextConfiguration(initializers = PostgresRepositoryTestInitializer.class)
@Transactional
public @interface PostgresRepositoryTest {

    @AliasFor("repositories")
    Class<?>[] value() default {};

    @AliasFor("value")
    Class<?>[] repositories() default {};
}
```

Only extend/compose/create as last resort

More

Approval Testing

```
@Test
void testList() {
  // given
  String[] names = {
    "Llewellyn",
    "James",
    "Dan",
    "Jason",
    "Katrina"
  };

  // when
  Arrays.sort(names);

  // then
  Approvals.verifyAll("", names);
}
```

```
[0] = Dan
[1] = James
[2] = Jason
[3] = Katrina
[4] = Llewellyn
```

SomeTest.testList.received.txt

https://github.com/approvals/ApprovalTests.Java

Property Based Testing

```
class FizzBuzzTests {

    @Property
    boolean every_third_element_starts_with_Fizz(@ForAll("divisibleBy3") int i) {
        return fizzBuzz().get(i - 1).startsWith("Fizz");
    }

    @Provide
    Arbitrary<Integer> divisibleBy3() {
        return Arbitraries.integers().between(1, 100).filter(i -> i % 3 == 0);
    }

    // ...
}
```

junit-quickcheck: https://github.com/pholser/junit-quickcheck

jqwik: https://jqwik.net/

More

- Testcontainers, of course
- To use or not to use Mocks
- Test Pyramid or Test Honeycomb or …
- “Testing on the Toilet”: https://testing.googleblog.com/2007/01/introducing-testing-on-toilet.html
- Enable Testcontainer Logs: https://maciejwalkowiak.com/blog/testcontainers-spring-boot-container-logs/
- …

Wrap Up

Conclusion

- Read the documentation
- Learn the basics
- Know some advanced stuff
- Keep your eyes open for helpful libraries/utilities/concepts
- Only extend included batteries when necessary

Do whatever works in your context!

And not what a random person on a stage tells you! ;-)

Thanks! Questions?

Michael Vitz

Mail	michael.vitz@innoq.com
X	@michaelvitz
Mastodon	@michaelvitz@innoq.social
LinkedIn	michaelvitz
Bluesky	@michaelvitz.bsky.social

https://www.innoq.com/en/talks/2024/05/beyond-built-in-advanced-testing-techniques-for-spring-boot-applications/
https://github.com/mvitz/beyond-spring-boot-testing
https://www.innoq.com/en/articles/2023/10/spring-boot-testing/

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim
+49 2173 3366-0

Ohlauer Str. 43
10999 Berlin

Ludwigstr. 180E
63067 Offenbach

Kreuzstr. 16
80331 München

Wendenstraße 130
20537 Hamburg

Spichernstraße 44
50672 Köln