Cambridge
Technology Partners

Java EE testing revisited with Arquillian

Thomas Hug
Bartosz Majsak

Cambridge
Technology Partners

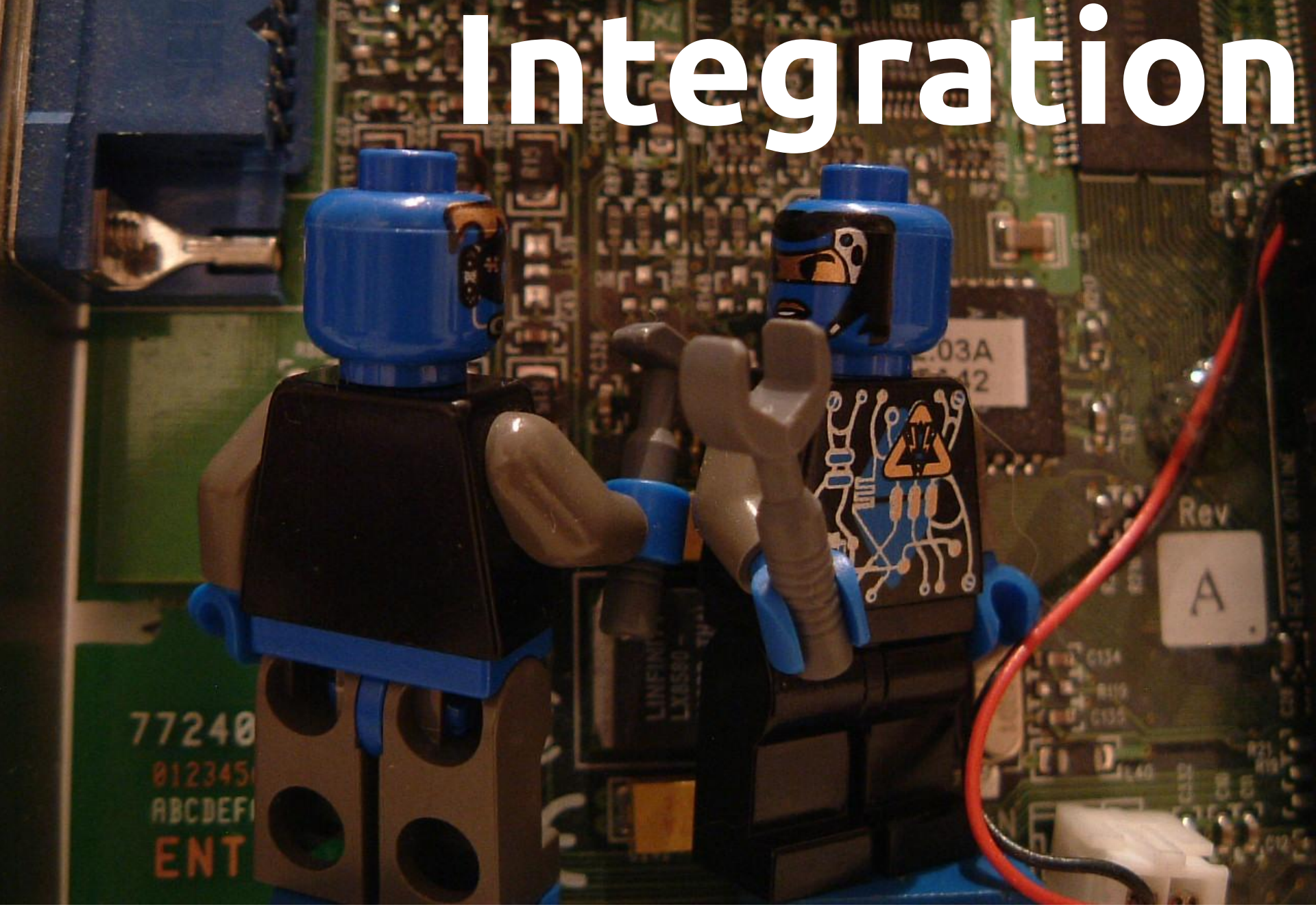
/ch/open

Agenda

- Who we are?
- Smooth introduction to Arquillian Universe
- Acceptance Test Driven Development
- Testing Java Persistence done right!
- Driving your browser
- Extending Arquillian


LEGO

Integration




Containers

Deployment

Alt+Shift+X, T
Alt+Shift+X, N

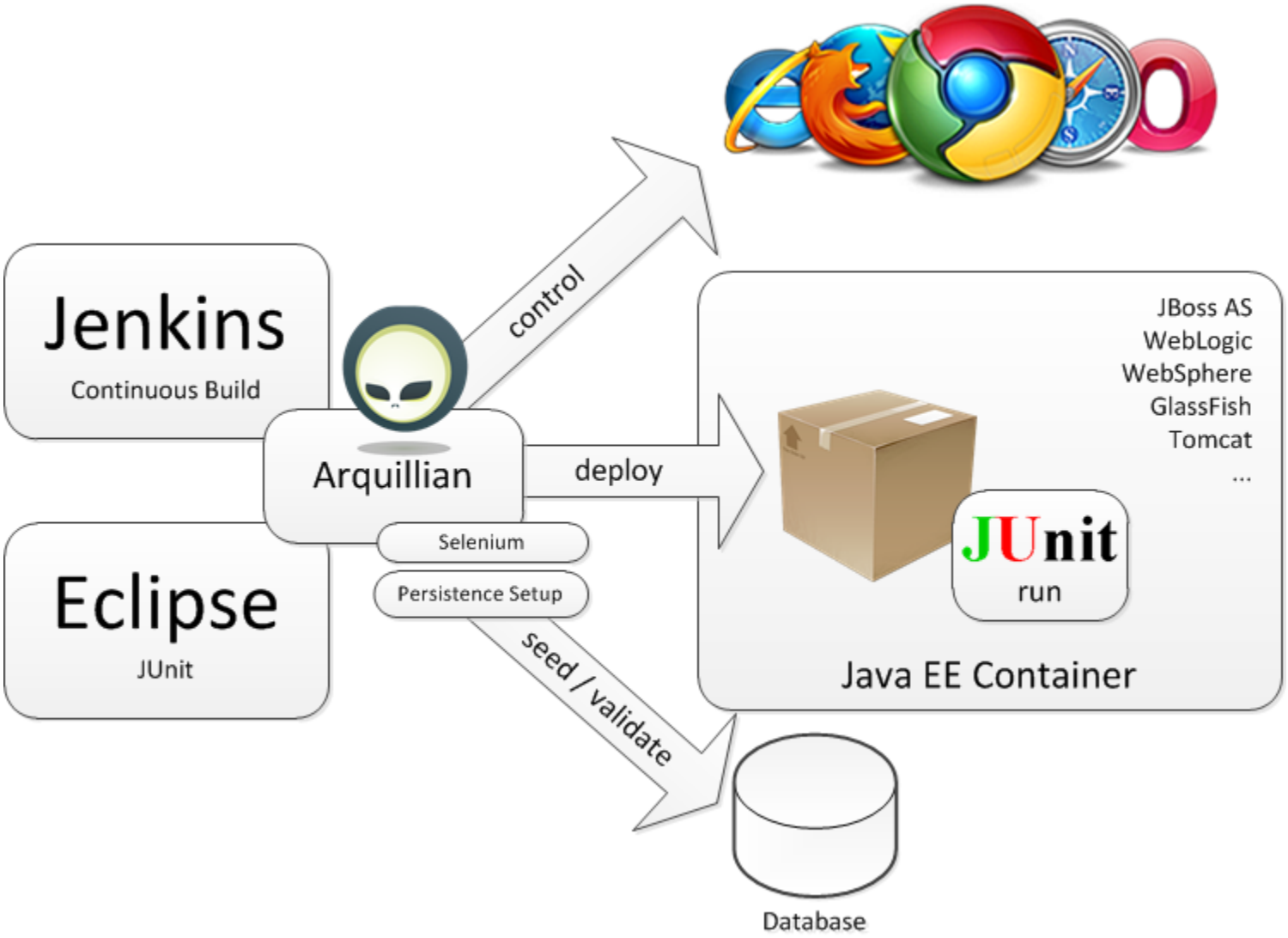
ShrinkWrap

Cambridge
Technology Partners

/ch/open

```
public class FluidOunceConverterTestCase {

    @EJB
    FluidOunceConverter converter;

    @Test
    public void shouldConvertFluidOuncesToMillilitres() {
          // given
          double ouncesToConvert = 8d;
          double expectedMillilitres = 236.588237d;

          // when
          double ouncesInMl = converter.convertToMillilitres(ouncesToConvert);

          // then
          assertThat(ouncesInMl).isEqualTo(expectedMillilitres);
    }

}
```

```java
@RunWith(Arquillian.class)
public class FluidOunceConverterTestCase {

    @Deployment
    public static JavaArchive createDeployment() {
        return ShrinkWrap.create(JavaArchive.class, "test.jar")
                         .addClasses(FluidOunceConverter.class,
                         FluidOunceConverterBean.class);
    }

    @EJB
    FluidOunceConverter converter;

    @Test
    public void shouldConvertFluidOuncesToMillilitres() {
          // given
          double ouncesToConvert = 8d;
          double expectedMillilitres = 236.588237d;

          // when
          double ouncesInMl = converter.convertToMillilitres(ouncesToConvert);

          // then
          assertThat(ouncesInMl).isEqualTo(expectedMillilitres);
    }

}
```


Jenkins
Continuous Build
Eclipse
JUnit
Arquillian
control
deploy
Selenium
Persistence Setup
seed / validate
JBoss AS
WebLogic
WebSphere
GlassFish
Tomcat
...
JUnit
run
Java EE Container
Database

RAD command line tool

- Standards focused
- Java technology agnostic
- Evolved from the seam-gen tool
- Interactive shell

http://forge.jboss.org

Extendable

- Various Plugins available
- Rich API, CDI based

Jump-starting projects and frameworks

- As we will see for Arquillian

Labs setup

VirtualBox image

Resources: 4 GB RAM, 2 cores (adjust if your machine can't take it)

username: arq , password: letmein

Labs

`~/dev/workspace/chopen-workshop-arquillian`

```
> git pull origin master
```

Eclipse IDE

`~/dev/eclipse`

Applications servers

`~/dev/as`, JBoss AS 7.1.1.Final is already added in Eclipse

*

Lab 01 - Getting ready with Arquillian

During this lab you will learn:

- How to create a project with Forge.
- Add Arquillian to a project using Forge.
- Create an Arquillian Test.

Lab 02 - Getting to know Ike

During this lab you will learn:

- The core functionalities of Arquillian.
- How to define test deployment.
- How to write JUnit tests powered by Arquillian.
- How to run the test in GlassFish Embedded both using a build tool and directly from the IDE.

Lab 02 - Task

- Fix failing test **`FluidOuncesConverterTest`** by implementing **`FluidOuncesConverter`** interface with the logic for converting fluid ounces to milliliters.
- Make it an **`@EJB`** component and test in Glassfish 3.1 Embedded (already pre-configured in pom.xml).
- This will require following steps:
  - Enhance our test to be Arquillian powered.
  - Define deployment package programmatically using ShrinkWrap.
    - add Fest-Assert in your test deployment

      ```
      .addPackages(true, "org.fest");
      ```

  - Run it directly from the IDE.
  - Run the test from command line using mvn clean package.

Lab 02 - Summary

- Arquillian brings your test to the runtime!
- Your application and tests can share the same programming model, regardless of technology stack.
- You can define your micro deployments using ShrinkWrap programmatic approach.
- To make JUnit test Arquillian-powered just by using `@RunWith`.
- Your tests can be run both from IDE or during the build.

Lab 03 - Into the world of containers

Arquillian supports three different types of containers:

- Embedded
- Managed
- Remote

In this lab you will learn how to use them.

Lab 03 - Summary

- Embedded container is entirely controlled by Arquillian and run under the same JVM. Arquillian is able to:
  - Start the container,
  - Deploy and execute your tests,
  - Undeploy test package,
  - Stop the container.
- Arquillian has exactly the same control over Managed Containers, but they are run in the separated JVM.
- Remote container needs to be up and running so Arquillian can deploy, execute and undeploy your tests.

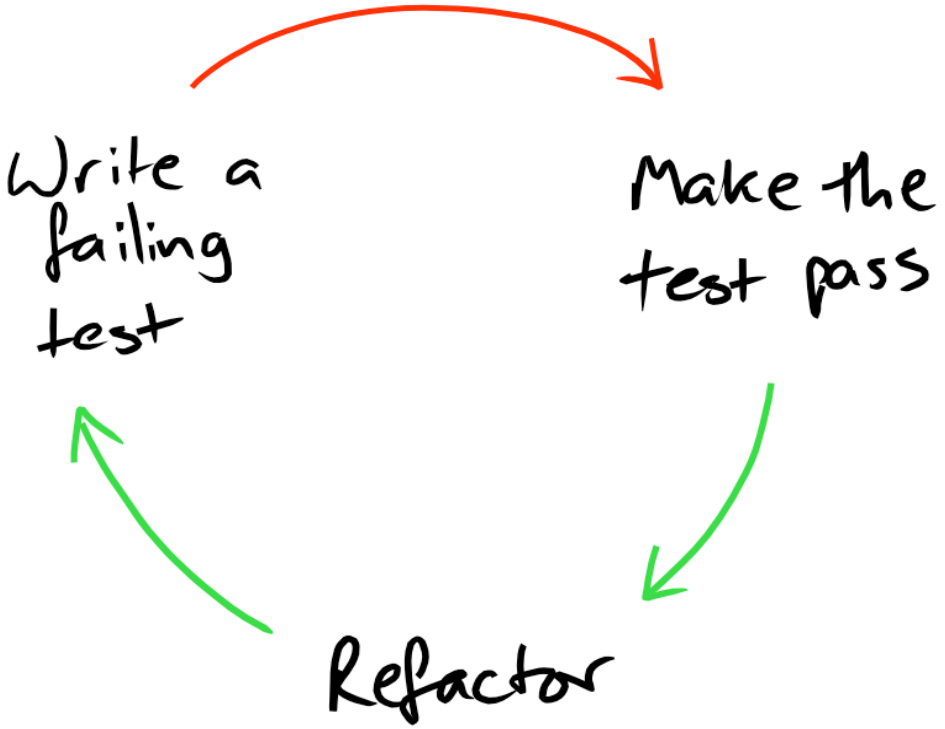
Lab 04 - Walking Skeleton

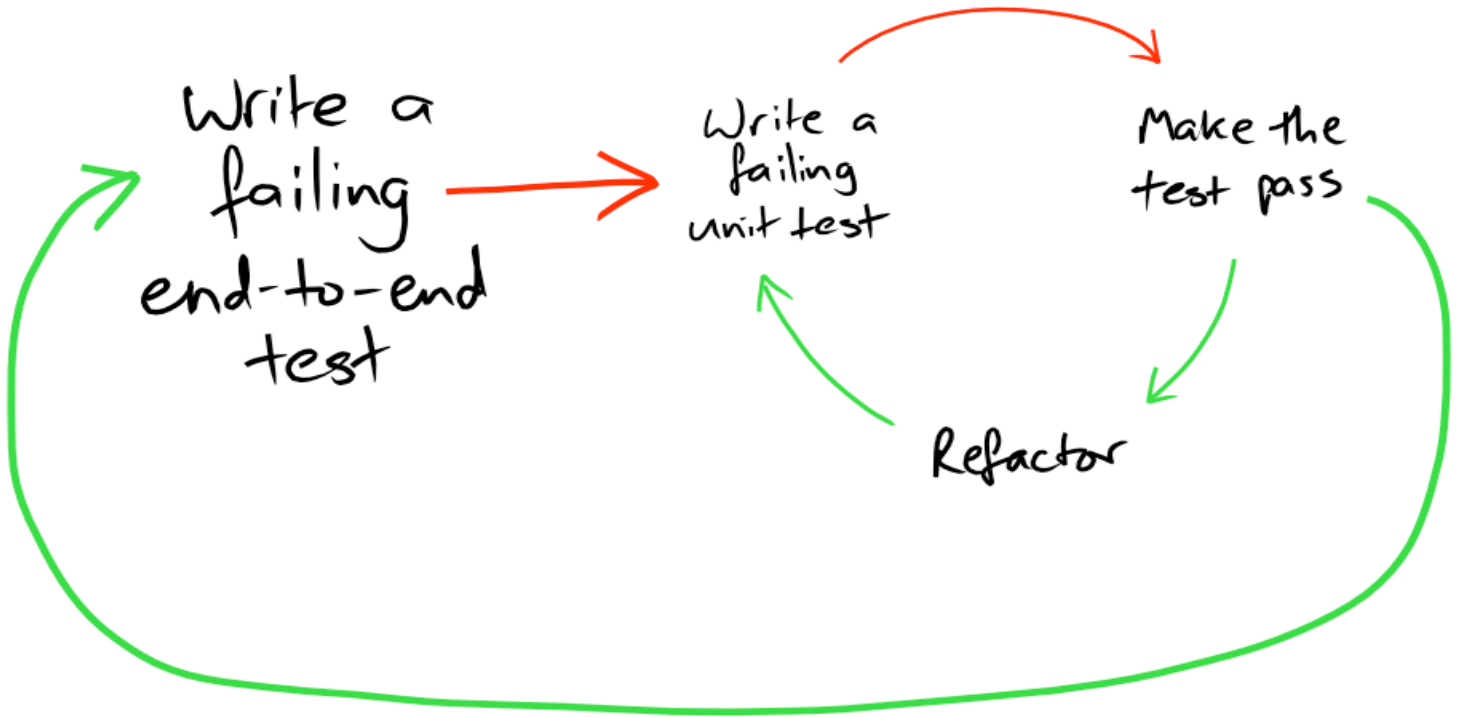
During this lab you will learn:

- What is BDD? How it can help you in writing better tests?
- How you can test your application using ATDD approach?
- How to test JPA effectively.

Scenario “Finding all beers”

Given “I'm on the main page”
When “I enter 'all' as search criteria”
Then “I should see list of all beers”


Write a failing test
Make the test pass
Refactor


Write a failing end-to-end test
Write a failing unit test
Make the test pass
Refactor

Lab 04 - Task

- Replace stubbed **`ch.open.arquillian.lab04.BeerService`** and implement JPA repository which will be used instead.
- Make **`BeerRepositoryBasicJpaTest`** to pass.
- Run acceptance test to verify that application is still providing required functionality.
- Rewrite **`BeerRepositoryBasicJpaTest`** using Arquillian Persistence Extension and use data sets to populate database for your tests.
  - `@UsingDataSet` - beers.yml skeleton is already in `src/test/resources/datasets`
  - `@ShouldMatchDataSet` (if you want to compare the content after)
- Persistence Extension docs :
  - https://docs.jboss.org/author/display/ARQ/Persistence

Lab 04 - Summary

- **Behavior Driven Development** let you think about tests in terms of features your application needs to deliver. BDD frameworks such as Spock provide DSL (based on Gherkin) which allows you structurize the tests as user stories / specifications.
- **Walking Skeleton** is the simplest deployable unit of the application providing minimalistic implementation of the given feature. Arquillian with container adapters and ShrinkWrap (with Maven Resolver) for building your package fits perfectly in this approach.
- **Arquillian Persistence Extension** simplifies in-container JPA testing by leveraging concept of DataSets from DBUnit which you can use to seed your database or verify content of the tables after test.

Lab 05 - Going into the details

During this lab you will learn:

- What are the **Page Objects** and why they are helpful.
- How to test your web application using **Drone Extension** to drive the browser and Arquillian to deploy your web application.

Lab 05 - Task

- Implement new Page Object **`BeerDetailsPage`** which will encapsulate web elements from the web site and expose Beer details as simple POJO.
- Extend **`BeerAdvisorPage`** by implementing `detailsOf` method which will allow to
  - Click on the link with the name of the beer (taken as parameter)

    hint: you can use XPath expression to find the matching link
  - Instantiate **`BeerDetailsPage`** which fill fetch web elements from the opened page with details of the selected beer
  - Extract this information and create Beer POJO containing them

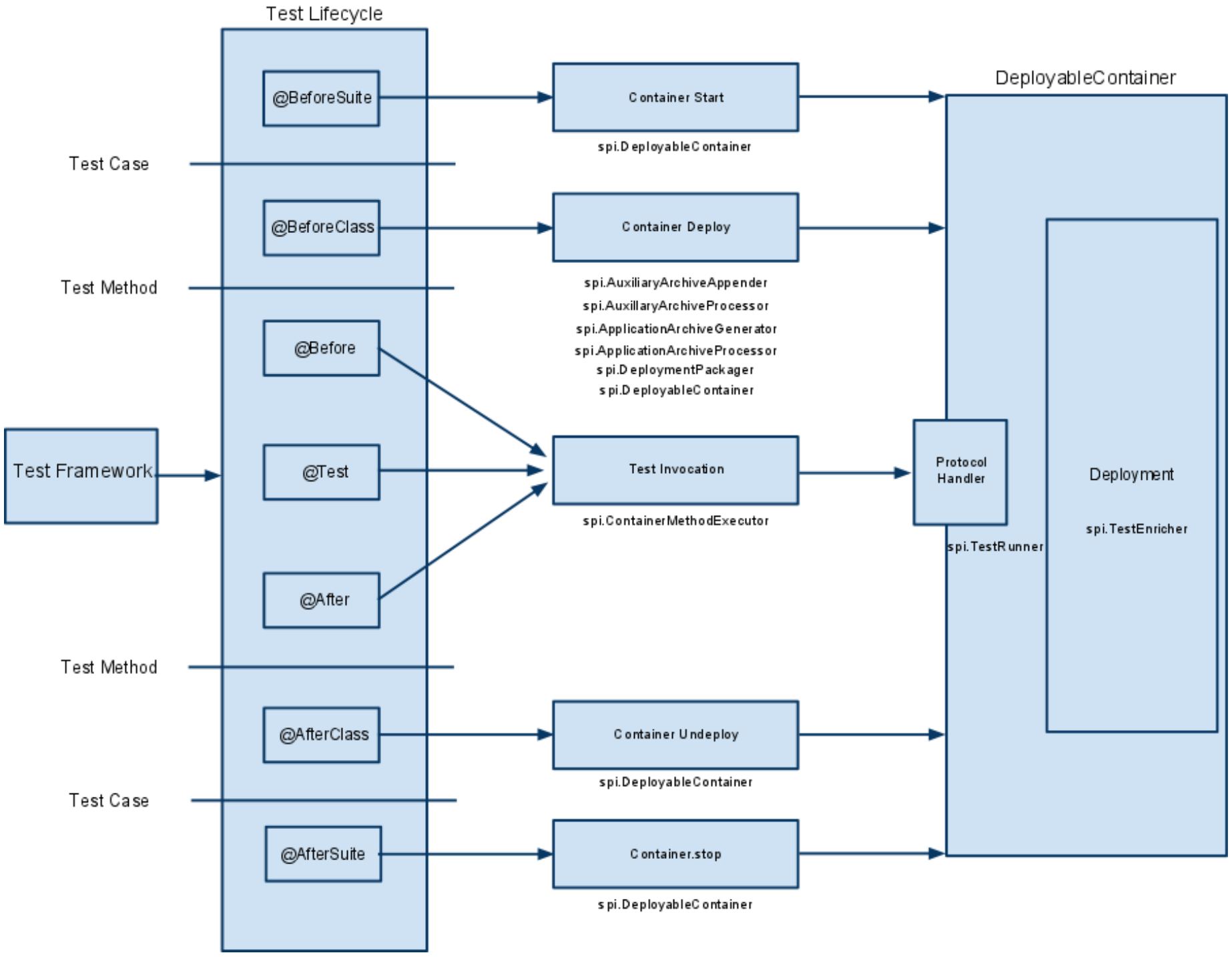
Lab 05 - Summary

- **Page Objects** let you encapsulate re-usable fragment on your web application.
- **Arquillian Drone extension** simplifies usage of Selenium.
- **Spock** allows you to structure your tests in BDD manner.
- Arquillian can be also used for black-box testing (as client). In this mode it only takes care of your deployment, but the test itself is run from outside the container.

Lab 06 - Beyond the limits

The goal of this lab is to explore and understand:

- Available SPIs.
- Event model.
- Prepare your extension package on the client side.
- Client and remote part of the extension.
- Register your own interceptor which can execute logic before and after test execution.


Test Lifecycle
@BeforeSuite
Container Start
spi.DeployableContainer
DeployableContainer
Test Case
@BeforeClass
Container Deploy
Test Method
spi.AuxiliaryArchiveAppender
spi.AuxillaryArchiveProcessor
spi.ApplicationArchiveGenerator
spi.ApplicationArchiveProcessor
spi.DeploymentPackager
spi.DeployableContainer
@Before
Test Framework
@Test
Test Invocation
spi.ContainerMethodExecutor
Protocol Handler
spi.TestRunner
Deployment
spi.TestEnricher
@After
Test Method
@AfterClass
Container Undeploy
spi.DeployableContainer
Test Case
@AfterSuite
Container.stop
spi.DeployableContainer

Lab 06 - Task

- Implement observer (**`@Observes`**) which will extract information from the test method (**`@Prompt`** annotation) and log it.
- Register it as a part of **`PromptRemoteExtension`**.
- Prepare extension deployment by adding all classes which are required in runtime.
- *hint: use ShrinkWrap API for registering* **`PromptRemoteExtension`** *as Service Provider*

```
.addAsServiceProvider(RemoteLoadableExtension.class,
                      PromptRemoteExtension.class);
```

Lab 06 - Summary

- **`org.jboss.arquillian.core.spi.LoadableExtension`** it's an entry point to register your own extension
- **`org.jboss.arquillian.container.test.spi.client.deployment.AuxiliaryArchiveAppender`** is responsible for creating extension jar which will be deployed together with the test
- **`org.jboss.arquillian.container.test.spi.RemoteLoadableExtension`** is used to register extension points such as observers in container
- Arquillian events model is very similar to CDI - **`@Observers`** can be used to hook into the test lifecycle

References

- http://arquillian.org/
- https://github.com/arquillian
- https://docs.jboss.org/author/display/ARQ/

- http://ctpjava.blogspot.ch/
- https://github.com/ctpconsulting/