Enterprise Programmering 1

Lesson 05: EJB

Prof. Andrea Arcuri

About these slides

- These slides are just high level overviews of the topics covered in class
- The details are directly in the code comments on the Git repository

EJB Types

- 3 types, using @ annotations on the class
- *@Stateless*
- *@Stateful*
- *@Singleton*

@Stateless

- A EJB which is not supposed to have own state, ie fields
  - eg, "*private int x = 0;*"
  - can still inject objects, like *EntityManager*
- Technically, you can declare field variables, but there is no guarantee call on proxyed EJB is always on same instance
- For a given EJB, Container can have a pool of instances, and each time you use an injected proxy it can call method on different instance

@Stateful

- Can have state, ie local variables
- *@Stateful* EJB are linked to users (to sessions, to be more precise)
- If you have many requests (eg web page visits) from different users, need to have a EJB instance for each of them
  - eg, 50,000 different users asking for a page using a *@Stateful* EJB? Then you need to keep 50,000 instances in memory
- JEE Container can automatically store EJB instances to disk when running out of space (and resume when those are needed)

@Singleton

- A EJB that can have state
- Only one instance exists in the whole Container
- Every time you inject a *@Singleton*, is always the same instance
- As the same singleton can be used by different threads (eg handling concurrent web page requests), each method invocation is automatically *synchronized* in the proxy class to avoid concurrency issues

Injection

- You can inject a EJB inside another EJB by declaring a variable annotated with *@EJB*
  - eg, *"@EJB private A a;"*
  - recall that you cannot instantiate a EJB directly with "new"
- Note: you can also use *@Inject*, but that is part of the **CDI** (Contexts and Dependency Injection) specs, which is a more general framework for injection, not just EJB
  - Note: we will not see the details of CDI specs in this course

@PostConstruct

- The Container, before doing dependency injection, needs to create an instance of the EJB with “new”
- This means that the code of the *constructor* is called BEFORE dependency injection (DI) is done
- If you need to access an injected variable in the constructor, you will hence get a null pointer exception
- A method marked with *@PostConstruct* will be executed AFTER the constructor and DI
  - so, useful when you need initializing code relying on injected variables

Container Deployment

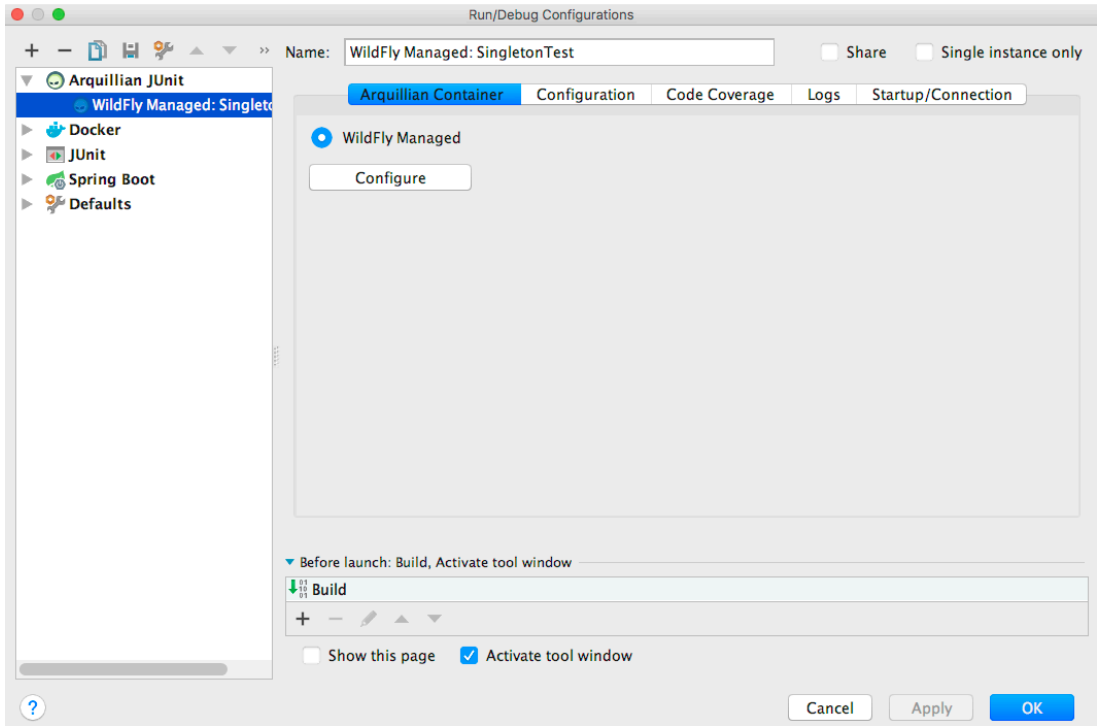
- To use EJBs, we need to run them in a JEE Container
  - WildFly, GlassFish, Payara, etc.
- We would need to package the JAR/WAR with our code, install it on a running container
- But before that, we would need to download, install and start a container
- But how to test the methods of EJBs directly from a JUnit test?

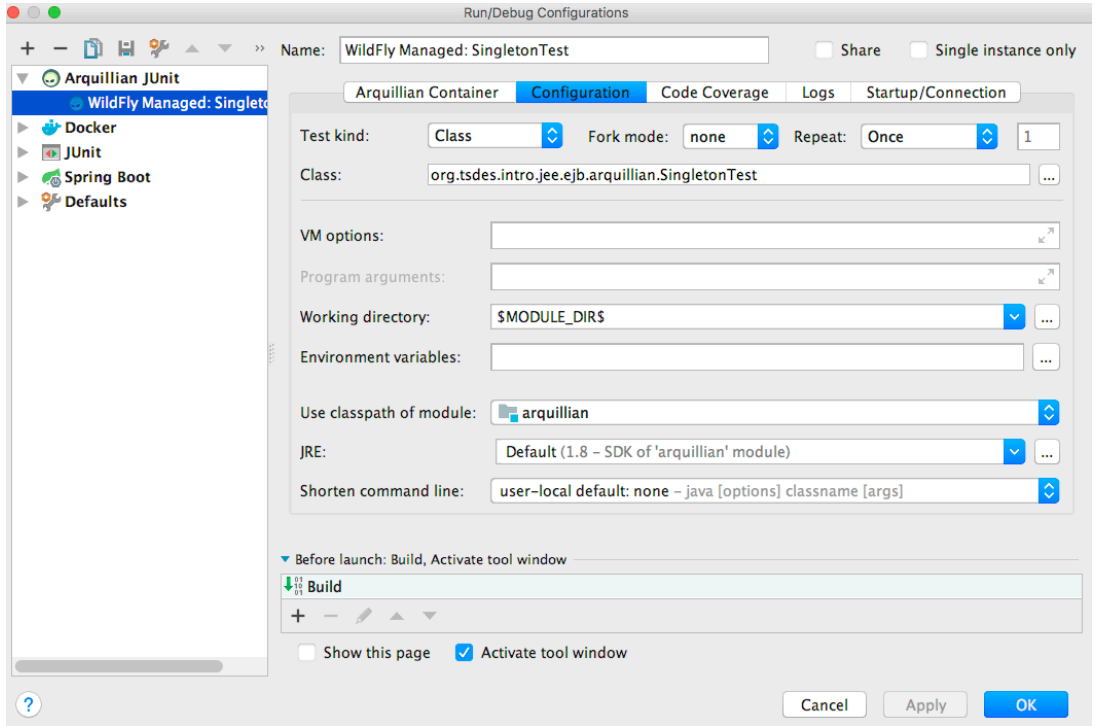
Arquillian

- A library extending JUnit that allows you to package JAR/WAR files directly from tests and deploy them on a container
- The tests themselves are run in the container, so can use dependency injection *@EJB*
- Configuration in special resource file called *arquillian.xml*
- Limitations: cannot just right-click in IDE to run tests, need some manual settings first…
- … plus, you still need to download and install a JEE Container

Test Configuration

- Arquillian “WildFly Managed”
- “Working directory” -> “$MODULE_DIR$”





Download/Install WildFly

- We do it with Maven plugin, as part of the build
- WildFly installed under the "*target*" folder
  - So it would be deleted when running "mvn clean"
- Need to run "*mvn test*" at least once to download/install WildFly BEFORE you can run tests in IntelliJ

Git Repository Modules

- *NOTE: most of the explanations will be directly in the code as comments, and not here in the slides*
- **intro/jee/ejb/singleton**
- **intro/jee/ejb/arquillian**
- **intro/jee/ejb/multithreading**
- **intro/jee/ejb/stateful**
- **intro/jee/ejb/callback**
- Exercises for Lesson 05 (see documentation)