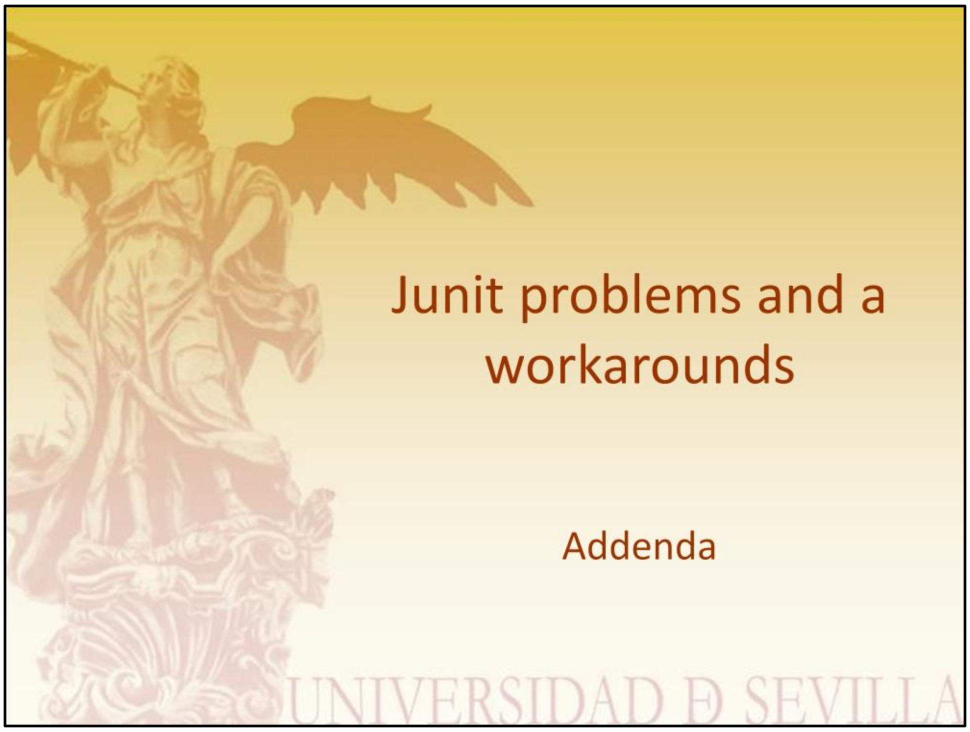


Junit problems and a workarounds

Addenda

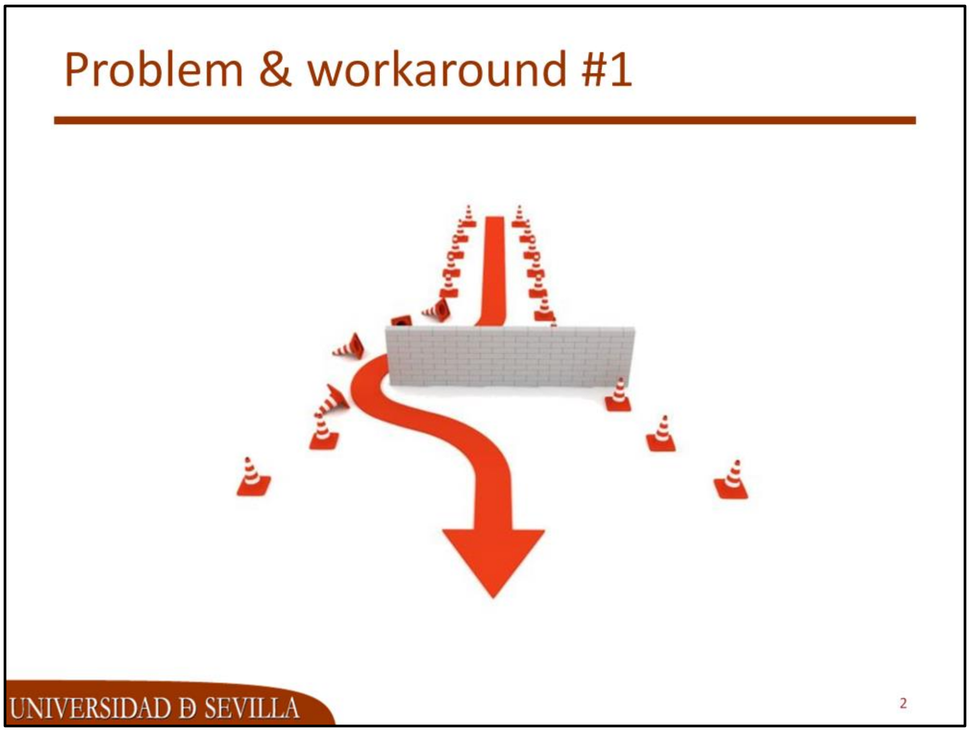
UNIVERSIDAD Ð SEVILLA

This addenda documents a couple of problems with JUnit and their corresponding workarounds.

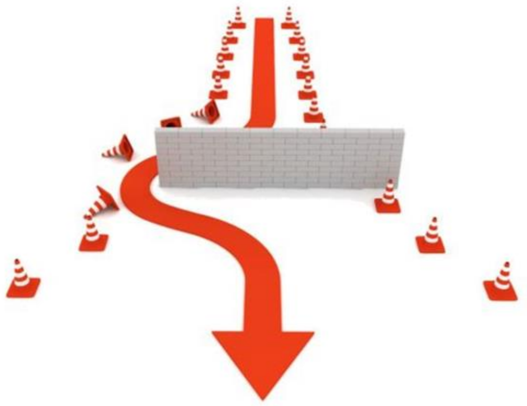
--

Copyright (C) 2018 Universidad de Sevilla

The use of these slides is hereby constrained to the conditions
of the TDG Licence, a copy of which you may download from
http://www.tdg-seville.info/License.html



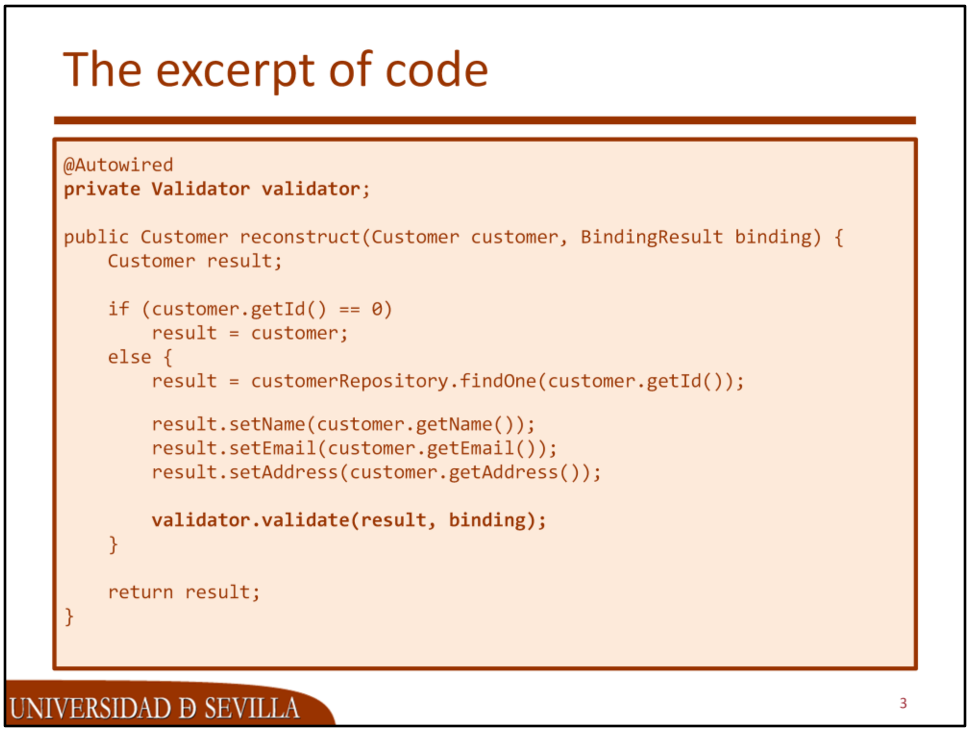
Problem & workaround #1



UNIVERSIDAD Ð SEVILLA

2

Let’s analyse the first problem and its workaround.



The excerpt of code

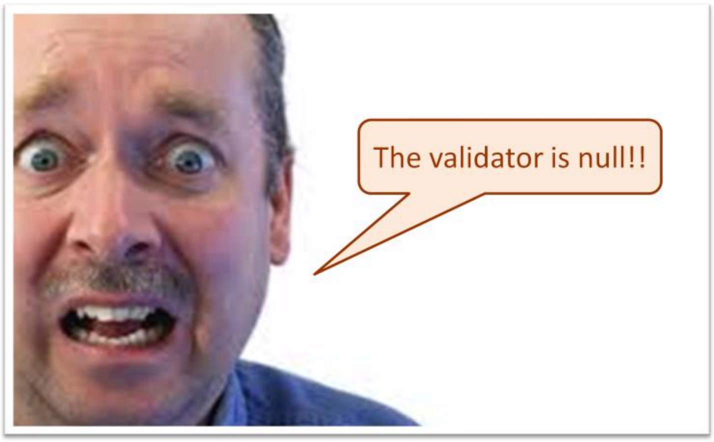
```
@Autowired
private Validator validator;

public Customer reconstruct(Customer customer, BindingResult binding) {
    Customer result;

    if (customer.getId() == 0)
        result = customer;
    else {
        result = customerRepository.findOne(customer.getId());

        result.setName(customer.getName());
        result.setEmail(customer.getEmail());
        result.setAddress(customer.getAddress());

        validator.validate(result, binding);
    }

    return result;
}
```

UNIVERSIDAD Ð SEVILLA

3

In Lesson L08, we discussed post hacking. We introduced form objects and pruned objects as two simple techniques to deal with such hacking. Both methods require the domain objects to be reconstructed and validated in order to be persisted, which led us to create the reconstruct method that we show in this slide. In this method, validation is explicitly invoked by means of an object called "validator", which is of type "org.springframework.validation.Validator", which provides a validate method. Recall that we need not provide an explicit implementation for the validator object since this attribute is autowired; in other words, Spring is responsible for finding the implementation and injecting it where appropriate.

The problem

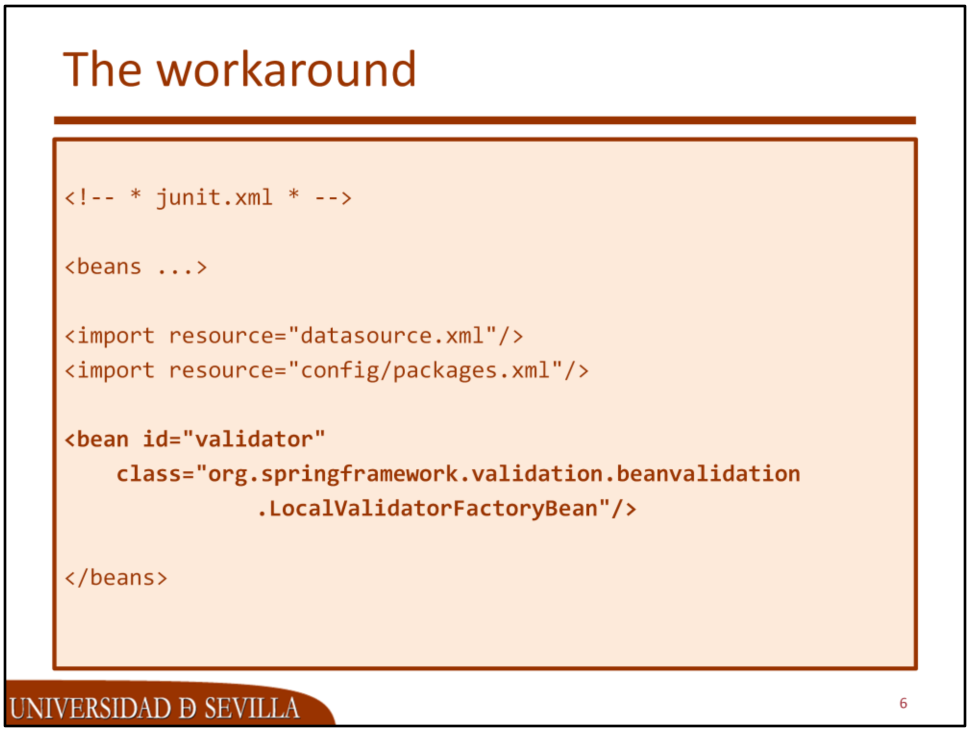


UNIVERSIDAD Ð SEVILLA 4

Unfortunately, there seem to be cases in which attribute “validator” is null when the reconstruct method is invoked from within a test case.

The explanation

We honestly don’t know why this happens. It may be a bug in the integration of JUnit and Spring or it may be the case that we made a mistake in the project template. Unfortunately, we have not been able to trace the bug down.

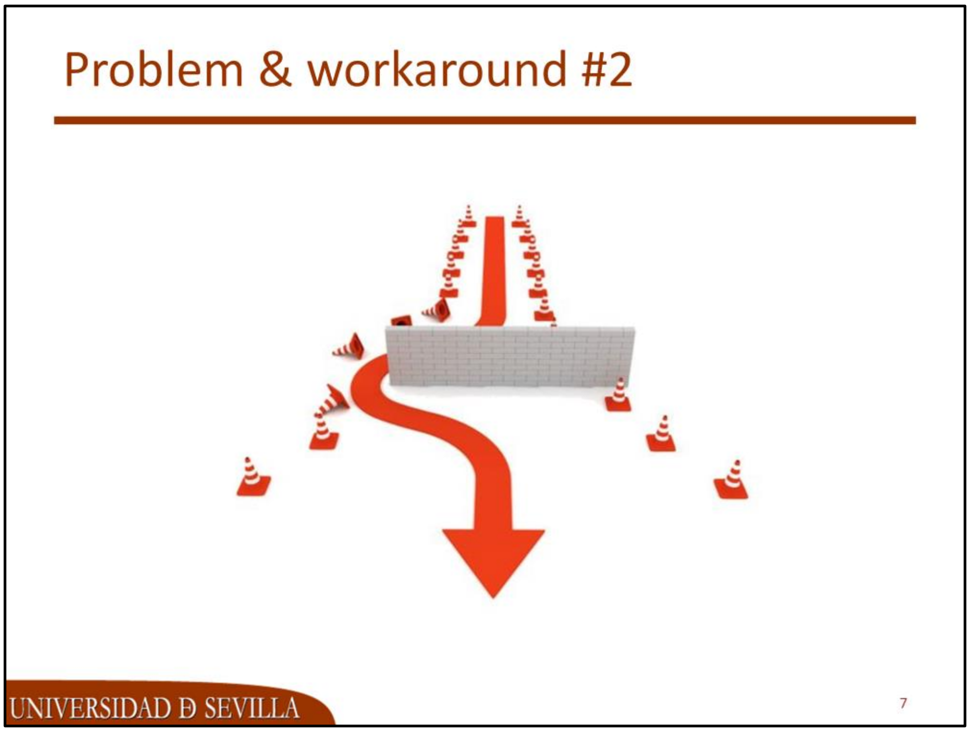


The workaround

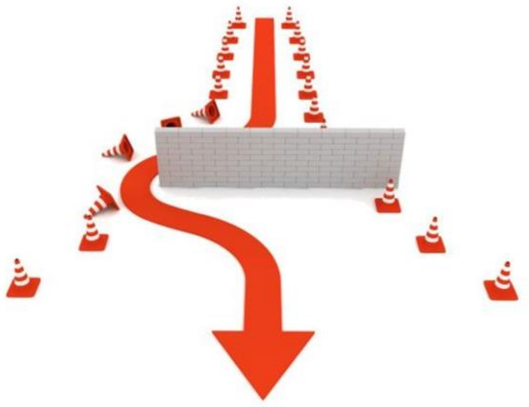
```
<!-- * junit.xml * -->

<beans ...>

<import resource="datasource.xml"/>
<import resource="config/packages.xml"/>

<bean id="validator"
    class="org.springframework.validation.beanvalidation
           .LocalValidatorFactoryBean"/>

</beans>
```

UNIVERSIDAD Ð SEVILLA 6

But we have a workaround if you find this problem: edit configuration file "junit.xml" and add the line in boldface in this slide. This line creates a validator explicitly so that Spring can locate it and inject it in your autowired attributes. Please, note that you must add this line only if you find the problem that was described in the previous slides; otherwise, don't add this line compulsively since it might result in two validators interfering with each other: the one that Spring creates automatically and the one that you create in this line.



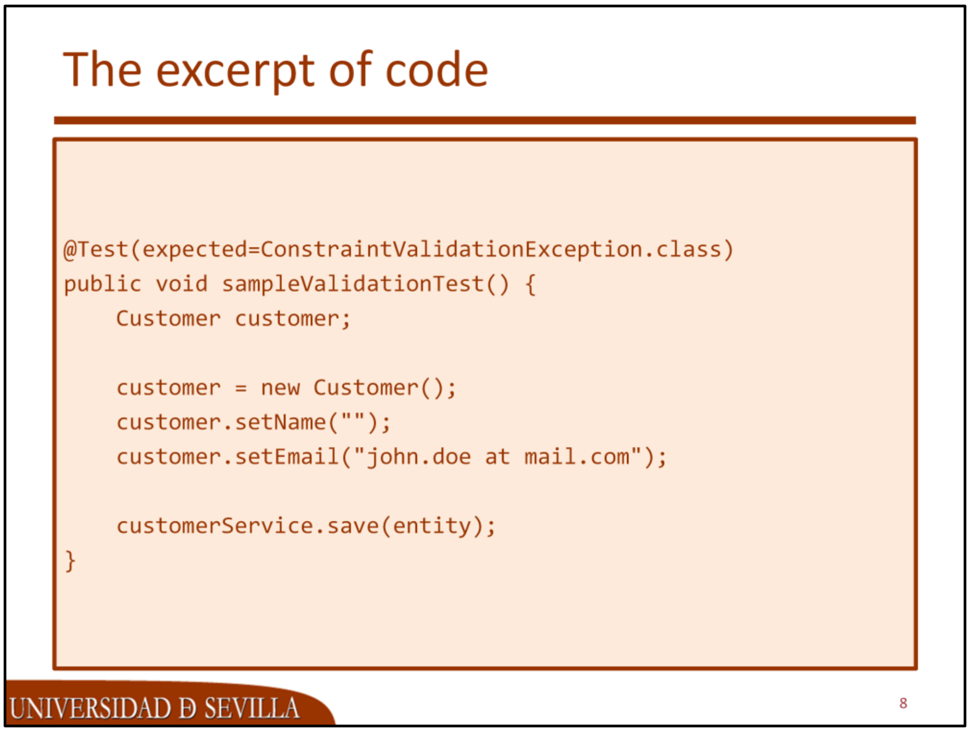
Problem & workaround #2



UNIVERSIDAD D SEVILLA

7

Let’s move on to the second problem and its workaround.



The excerpt of code

```
@Test(expected=ConstraintValidationException.class)
public void sampleValidationTest() {
    Customer customer;

    customer = new Customer();
    customer.setName("");
    customer.setEmail("john.doe at mail.com");

    customerService.save(entity);
}
```

UNIVERSIDAD Ð SEVILLA

8

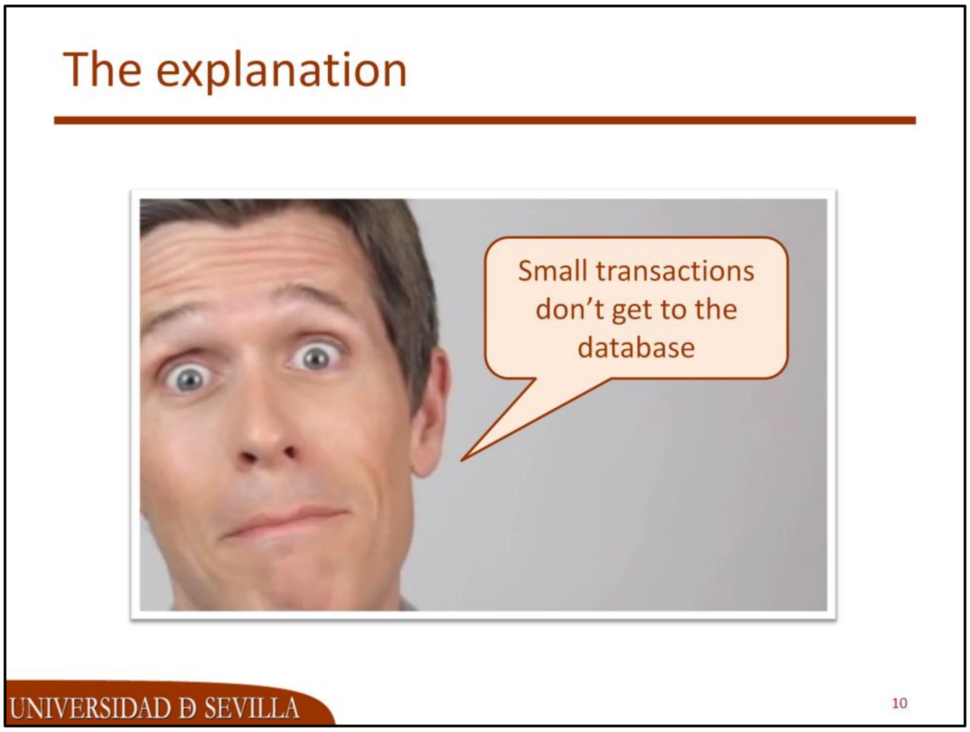
This slide shows a simple test that doesn’t involve any business rules, but validation constraints only.   Note that a customer entity is created and that the name is set to an empty string and the email to a string that is not a valid email address. Assuming that attribute “name” has a “@NotBlank” constraint and attribute “email” has an “@EMail“ constraint, the attempt to save this entity to the database should result in a constraint validation exception.

The problem

UNIVERSIDAD Ð SEVILLA

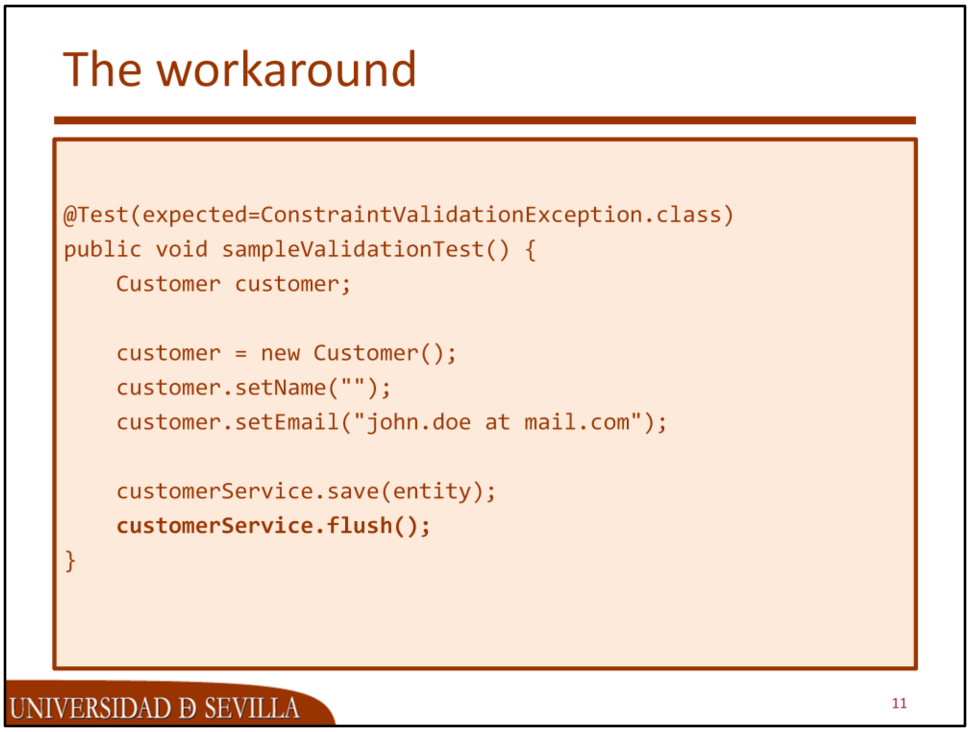
9

Unfortunately, there are cases in which the test fails because no exception is thrown.



The explanation

It's a problem with small transactions that involve very few objects and don't break any business rules, but validation constraints only. In such transactions, the objects may be kept in the cache, instead of being persisted to the database. When the test finishes, the transaction is automatically rolled back, which means that the objects never reach the database and are never actually validated. Note that it's an efficient approach that works pretty well in a production system, but, unfortunately, it doesn't help us to test our code properly.



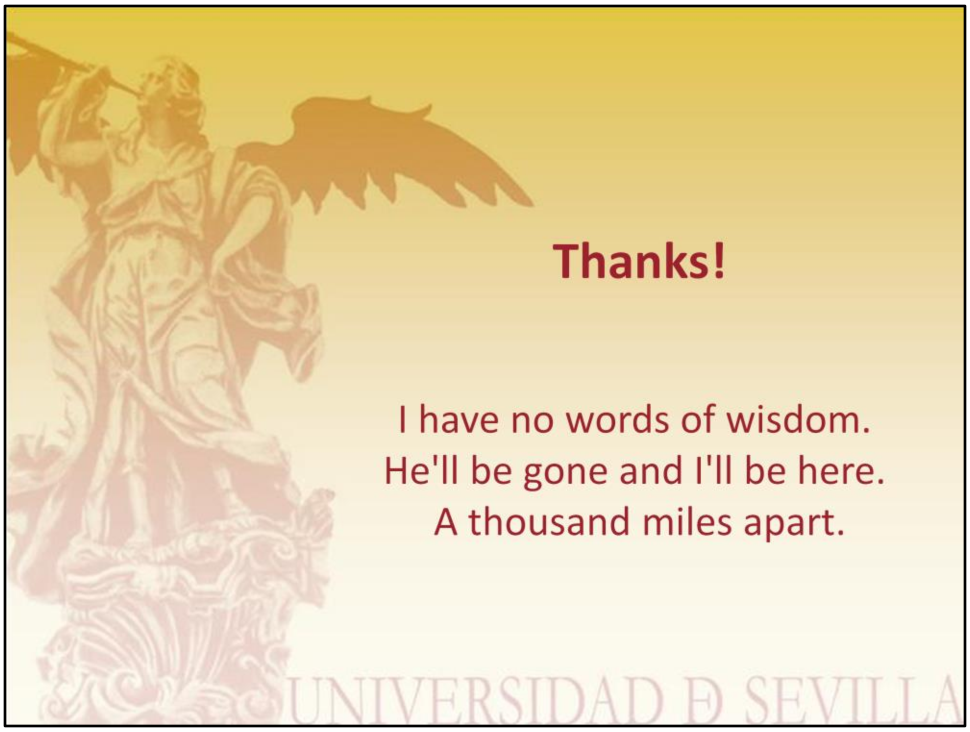
The workaround

```
@Test(expected=ConstraintValidationException.class)
public void sampleValidationTest() {
    Customer customer;

    customer = new Customer();
    customer.setName("");
    customer.setEmail("john.doe at mail.com");

    customerService.save(entity);
    customerService.flush();
}
```

UNIVERSIDAD D SEVILLA

11

The workaround is simple: add a method to flush your repositories and invoke it in your tests. This forces the cache to be saved to the database, which then forces the data to be validated.



Thanks!

I have no words of wisdom.
He'll be gone and I'll be here.
A thousand miles apart.

UNIVERSIDAD Ð SEVILLA

Thanks for reading this addenda!