

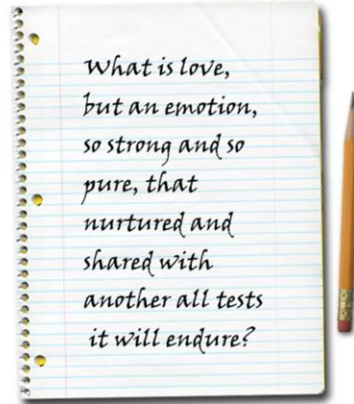
Welcome to this lesson! Our goal's to keep practicing regarding repositories and services.

--

Copyright (C) 2017 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>

The problem: a notebook



UNIVERSIDAD DE SEVILLA

We'll stick with quite a simple project: a notebook to which people can post notes. It's very similar to the bulletin board, the difference being that notes are private. That is, a customer should be able to create notes and list, update, or delete them, but he or she shouldn't be allowed to manipulate the notes that were created by other customers.

Ready to start riding?



UNIVERSIDAD DE SEVILLA

It's like riding this bike because we've provided you with a lot of support. The materials that accompany this lecture provide a project called "Notebook" that is almost complete; you only need to provide a couple of repositories and services so that it works. Please, note that we won't provide a detailed description of what you have to do to load the project and create the corresponding database; we assume that you already command these basic steps.

You're constrained!



UNIVERSIDAD DE SEVILLA

Please, note that you're totally constrained. Your only duty in this project is to implement the repositories and services that we describe later; you aren't allowed to change anything else. Please, take this very seriously: the other parts of the project were implemented by other people; you can't change the artefacts they've produced and hope this doesn't mess things up!

Your first day as a software engineer

*So, how was your
first day on the job?*



UNIVERSIDAD DE SEVILLA

5

Please, realise that this problem is very similar to the problems that you'll have to solve on your first day as a software engineer: your boss will task you with a very simple task like creating a repository or a service building on a specification that will be very similar to ours; very likely, you'll also have to implement a few business rules.

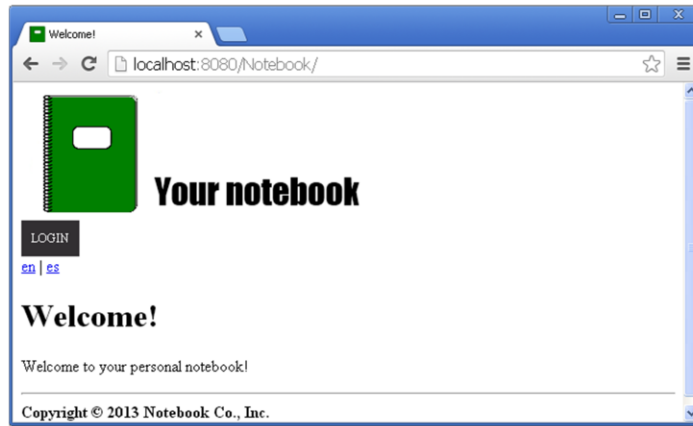


The roadmap is very similar to the previous lecture: a user's tour, a description of the UML domain model, and then a description of the repositories and services that you have to implement; finally, we'll report on the checks that you have to implement.



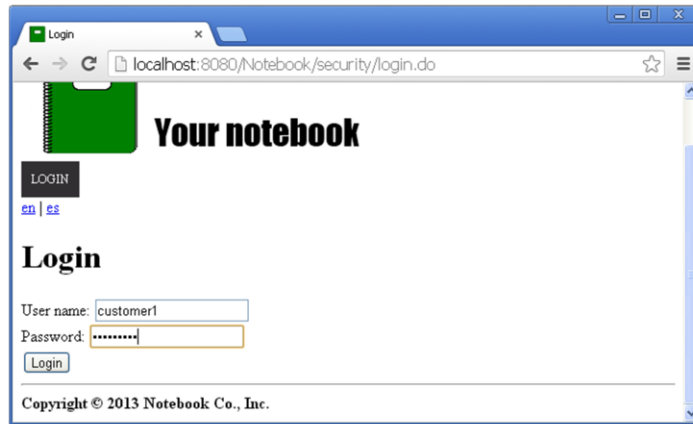
Let's start with the user's tour.

The welcome screen



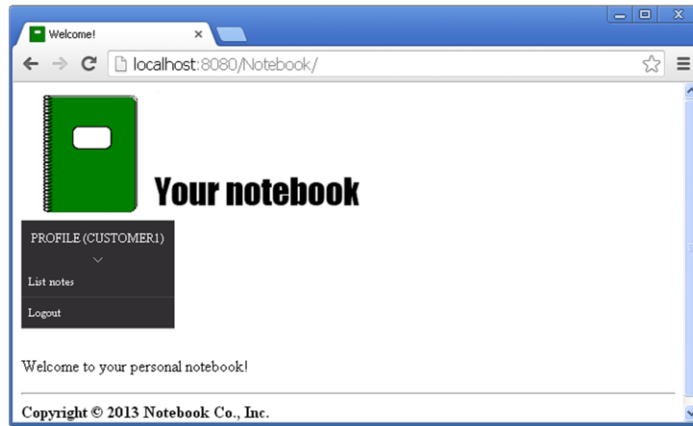
This is the welcome screen. By now, you should be very familiar with it. It has a logo, a simple menu that just displays “Login”, a language bar, and a welcome message.

Logging in as a customer



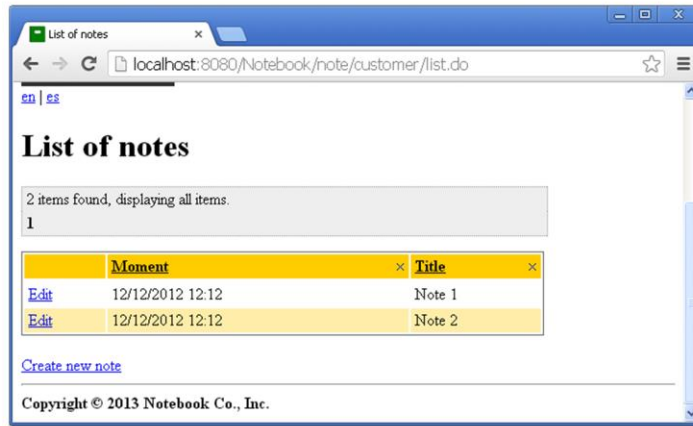
You may log in as a customer using two pre-defined user accounts:
“customer1/customer1” and “customer2/customer2”.

The main menu



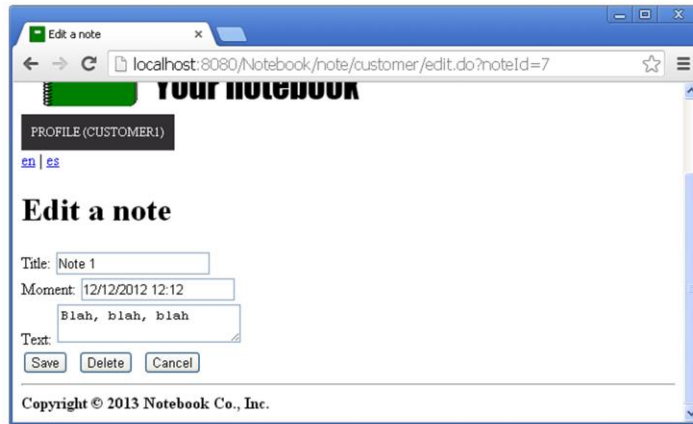
Once you're logged in, the menu will change and will show a couple of options, namely: "List notes" and "Logout".

Listing notes



This is what the application should look like whenever you select option "list notes". It shows a couple of notes that were written by "customer1". In the bulleting board, we showed every bulletin to every customer; in this application, we must show only the notes that the customer who is logged in has written. Note that, as usual, the listing provides links to edit the notes and a link to create a new note.

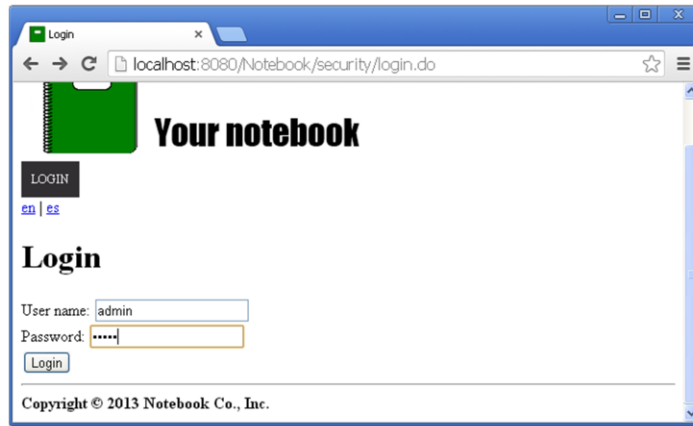
Editing a note



The screenshot shows a web browser window titled 'Edit a note' with the address bar displaying 'localhost:8080/Notebook/note/customer/edit.do?noteId=7'. The page header includes 'YOUR NOTEBOOK' and 'PROFILE (CUSTOMER1)' with links for 'en' and 'es'. The main heading is 'Edit a note'. The form contains the following fields: 'Title' with the value 'Note 1', 'Moment' with the value '12/12/2012 12:12', and 'Text' with the value 'Blah, blah, blah'. At the bottom of the form are three buttons: 'Save', 'Delete', and 'Cancel'. The footer of the page reads 'Copyright © 2013 Notebook Co., Inc.'.

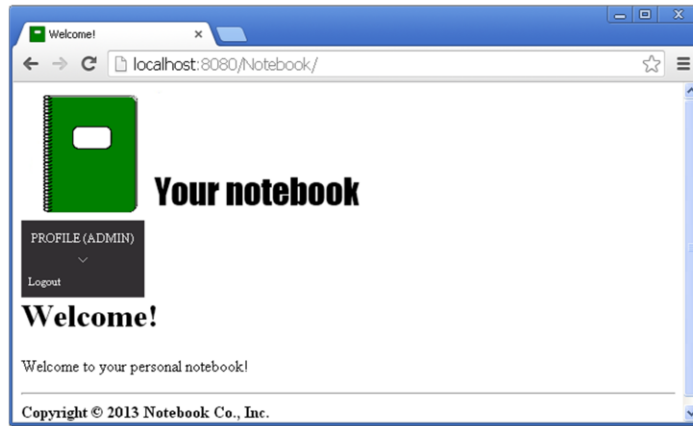
When the user clicks on an edit link or a create link, the edition form should look like in this slide. When he or she presses the “Save” button, the system must attempt to save the note to the database and display error messages if there are problems; when he or she presses the “Delete” button, the system must delete the note or show an error message if it’s not possible; when he or she presses “Cancel”, it must return to the listing.

Logging in as an administrator



You can also log in as an administrator.

The main menu

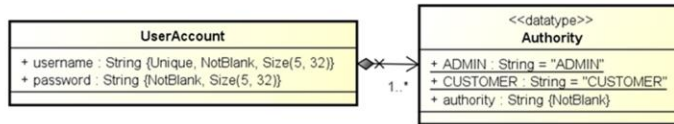


In this case, the main menu doesn't offer any options, except for logout. In other words, we won't implement any administrator's functionalities.



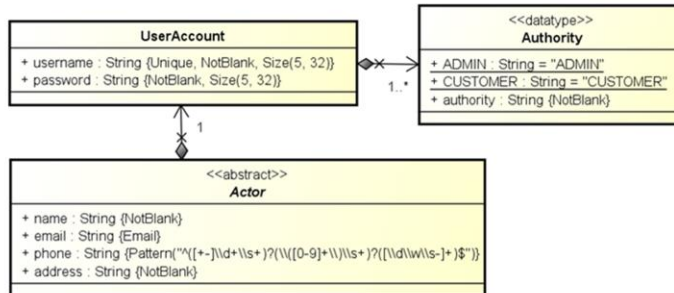
Let's now present the UML domain model.

The model (I)



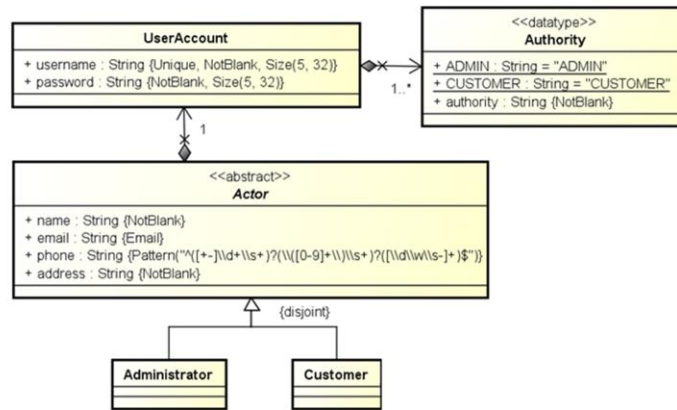
We'll introduce it in several slides. Classes "UserAccount" and "Authority" should be very familiar to you, so we won't provide any additional details on them.

The model (II)



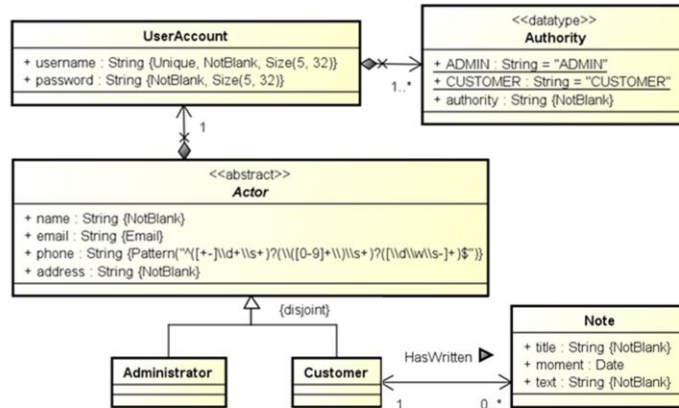
We'll use a class called "Actor" to represent the actors of the system, each of which will be characterised by means of a name, an email, a phone number (with international prefixes if necessary), and an address.

The model (III)



We'll specialise class "Actor" into two subclasses: "Administrator" and "Customer".

The model (IV)



The administrators don't have any other features, but every customer may write an arbitrary number of notes, each of which is written by a single customer. Regarding notes, the system stores information about their title, the moment when they were written or modified for the last time, and a piece of text. Note that the moment should be set automatically, when a customer creates or edits a note.



The UML domain model was simple, wasn't it? Let's now provide a little more information on the repositories that you have to implement.

Repository #1

- Name
 - CustomerRepository
- Methods
 - Customer findByUserAccountId(int id)

The first repository must be called “CustomerRepository”, and it must provide a single method with the signature in this slide. This method is intended to return the customer whose account identifier is passed as a parameter, if any. It’s quite a simple query!

Repository #2

- Name
 - NoteRepository
- Methods
 - `Collection<Note> findById(int id)`

The name of the second repository must be “NoteRepository”, and it must provide a single method with the signature in this slide. This method is intended to return the collection of notes that were written by the customer whose identifier is passed as a parameter. Neither is it too difficult!



Let's now provide a few details on the services that you have to implement.

Service #1

- Name
 - CustomerService
- Methods
 - Customer findByPrincipal()
 - Customer findByUserAccount(
 UserAccount userAccount)

UNIVERSIDAD DE SEVILLA

The name of the first service must be “CustomerService” and it must implement the following methods:

- “findByPrincipal”: it returns the customer who is currently logged in to the system.
- “findByUserAccount”: it returns the customer who has the user account that is passed as a parameter.

Service #2

- Name
 - NoteService
- Methods
 - Note create(Customer customer)
 - Note findOneToEdit(int id)
 - void save(Note note)
 - void delete(Note note)
 - Collection<Note> findByPrincipal()
 - void checkPrincipal(Note note)

UNIVERSIDAD DE SEVILLA

The name of the second service must be “NoteService”, and it must implement the following methods:

- “create”: it returns a new note whose author’s the customer who is passed as a parameter.
- “findOneToEdit”: it returns the note whose id is passed as a parameter, but raises an exception if the principal isn’t the customer who wrote it.
- “save”: it saves “note” to the database.
- “delete”: it deletes “note” from the database.
- “findByPrincipal”: it returns the collection of notes authored by the principal.
- “checkPrincipal”: it checks that the principal may have access to “note”, i.e., that “note” was written by the principal.



Let's now delve into the checks that you're requested to implement.

Service #1

- Name
 - CustomerServiceTest
- Methods
 - testFindByPrincipal

First, you have to implement the following checks regarding the customer service:

- “testFindByPrincipal”: write a check that authenticates as “customer1”, then retrieves the principal, checks that it is not null, checks that the user name of the principal is “customer1”, and, finally, logs out.

Service #2

- Name
 - NoteServiceTest
- Methods
 - testCreate
 - testSave
 - testDelete
 - testFindOneToEditPositive
 - testFindOneToEditNegative

UNIVERSIDAD D SEVILLA

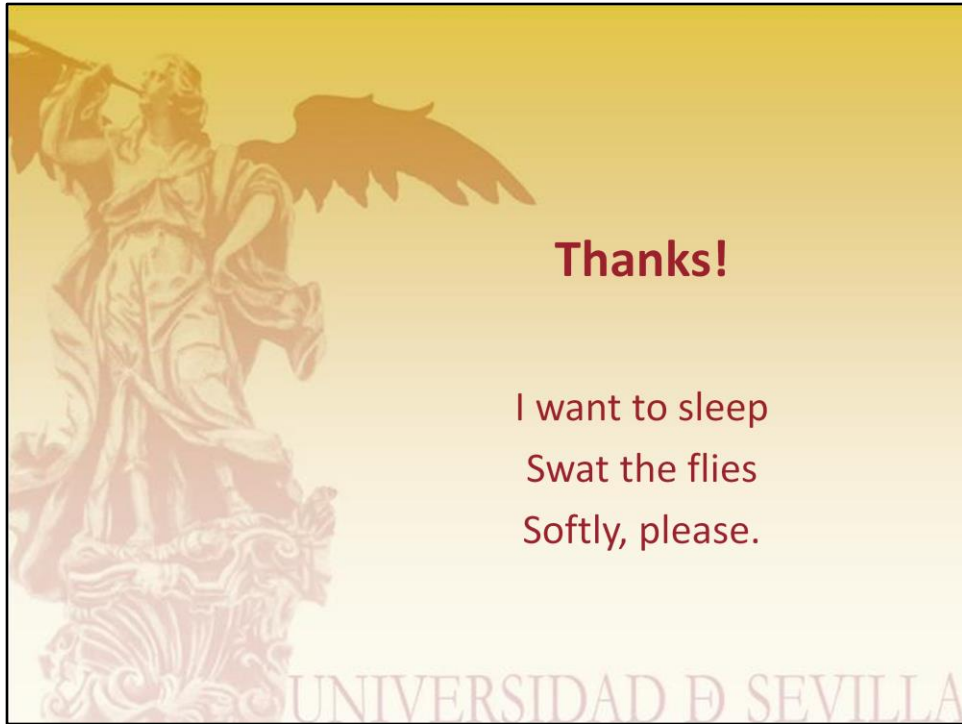
First, you have to implement the following checks regarding the notes service:

- “testCreate”: authenticate as “customer1”, retrieve the principal, create a note for the principal, and then check that the customer that is associated with the note is the principal, that the moment is not null, but the title and the text are null.
- “testSave”: authenticate as “customer1”, retrieve the principal, create a note for the principal, set the title and the text, save the note, then retrieve the list of notes written by the principal and check that it includes the previous note.
- “testDelete”: authenticate as “customer1”, retrieve the principal, create a note for the principal, save it, delete it, retrieve the notes of the principal and check that it does not include the note that was created at the beginning of this check.
- “testFindOneToEditPositive”: authenticate as “customer1”, retrieve the principal, create a note and save it, invoke method “findOneToEdit” to retrieve that note, and check that the saved note and the note returned by “findOneToEdit” are equal.
- “testFindOneToEditNegative”: authenticate as “customer1” and retrieve any of the notes that he or she’s written; log out, authenticate as “customer2” and invoke the “findOneToEdit” method to retrieve the note of “customer1”. Note that you must enclose the statements that correspond to the actions performed by “customer2” in a try catch block.

Ready to work?



Are you ready to work? C'mon!



Thanks for attending this lecture! See you next day!