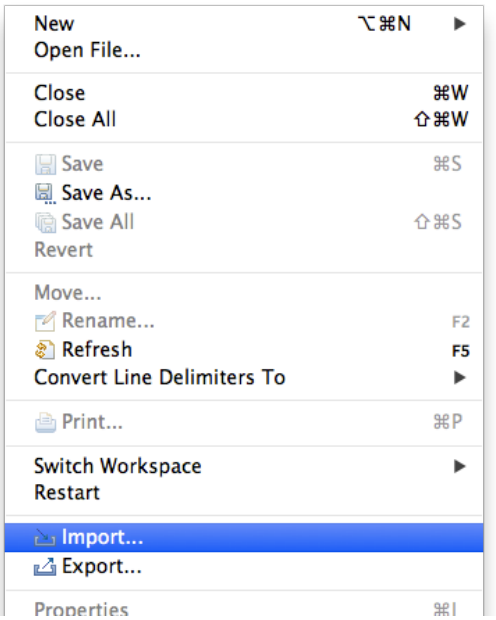
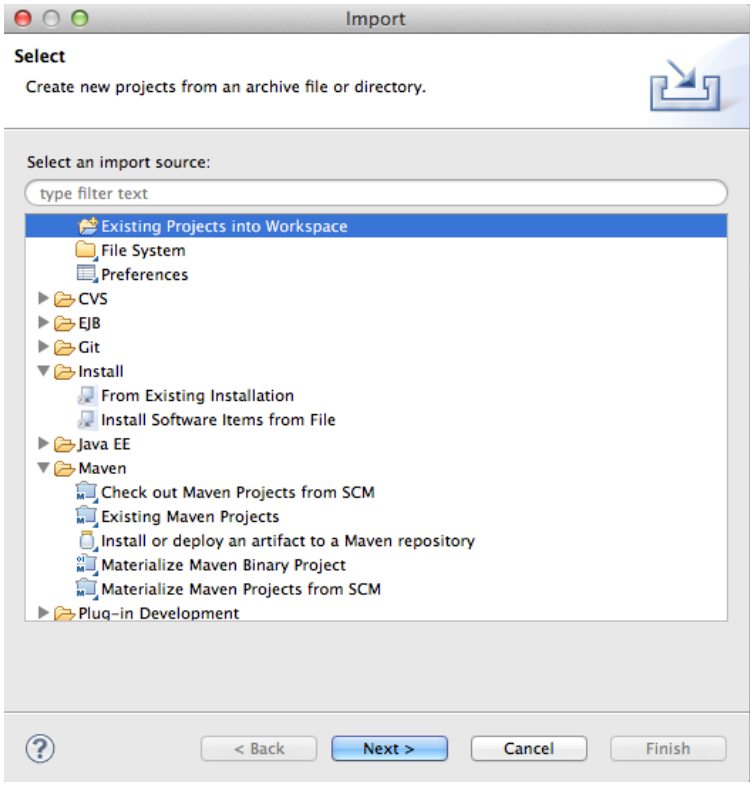
Instructions

Setting up the environment

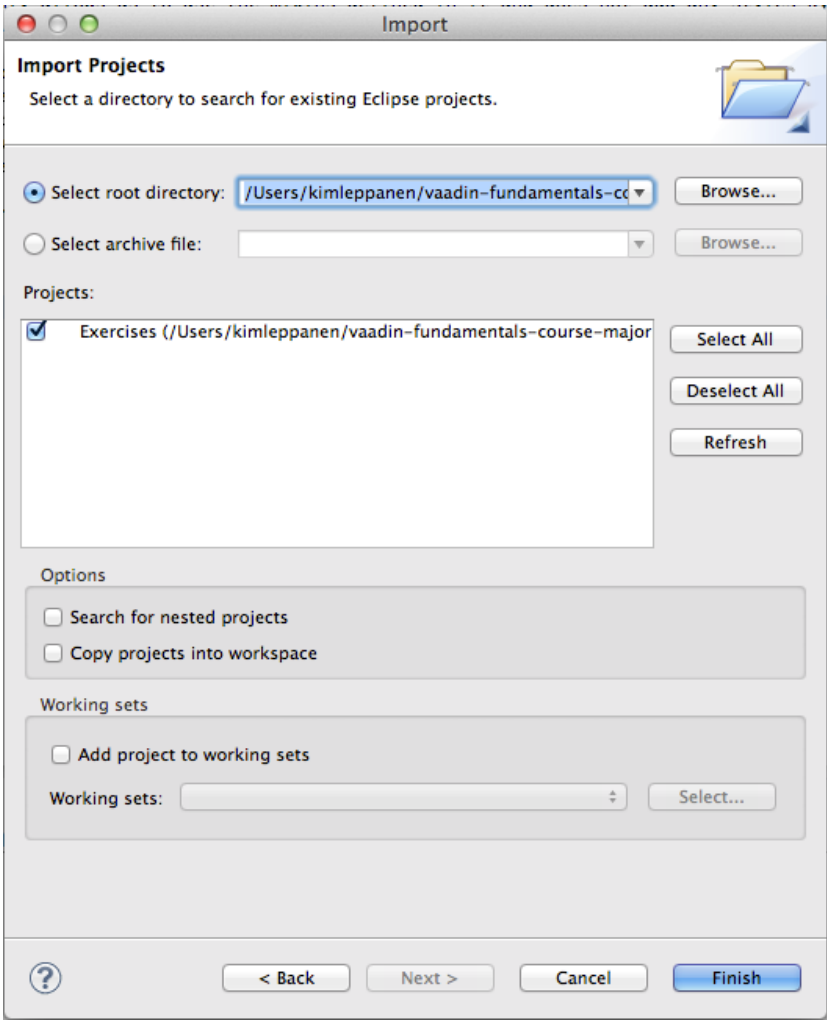
Start by downloading the exercises.zip file and extract it to location of your choice.

Open Eclipse, choose File > Import and choose “Existing project into Workspace”

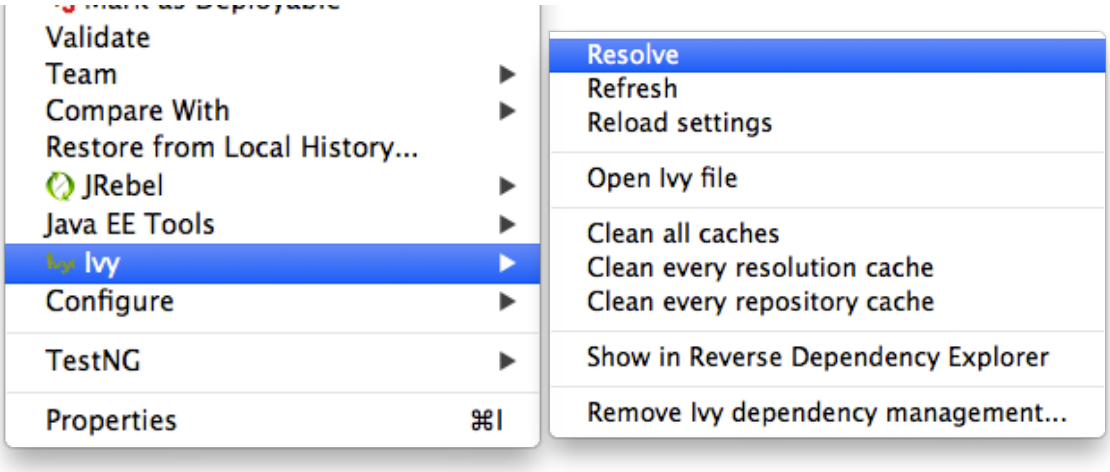




Browse to the folder where you extracted the zip, and choose the Exercises folder. Select the Exercises project and click on finish.



Right-click on the imported project and choose Ivy > Resolve.



You should now be all set up. Try deploying the project to Tomcat and browse to http://localhost:8080/Exercises/

Exercises

Exercise 1 : Polling

This exercise is simple enough to get familiar of using simple components and events in Vaadin, but also demonstrates the synchronous model.

The goal is to create an application that contains three buttons and a progress indicator. One button is for showing a notification with the text “Hello” in it. The two other buttons should start a long running operation on the server, which simulates a time consuming process on the server.

First of the buttons should start the operation synchronously, so that the operation is ran in the same thread. There is a helper method, `startOperation(boolean)`, that will actually execute the long running operation. The second button should start the same process, except that the process is ran asynchronously in a separate thread. Again, use the same helper method to start the process.

1. Create a Button instance for all the three buttons in the application
2. Set the width of the buttons to 150px, so that they are equally wide.
3. Add a Button.ClickListeners to the Buttons
    a. In ClickListener, call `startOperation()` with either the parameter `true` or `false`
    b. When the notification button is clicked, open a notification with the text “Hello” using the Notification.show(“”) method.

Book of Vaadin
Button: 154
Handling events with listeners: 74
Notifications: 83

Exercise 2: Navigation Bar

There is some kind of a navigation bar in almost every Vaadin application. A typical navigation bar is at the top of the page, most of the buttons or links aligned to the left side of the page and one button (for example, “logout”), aligned separately to the right side of the page.

Your task is to create a navigation bar containing four buttons. Three of the buttons should be aligned to the left and one to the right. The caption of the buttons doesn’t matter, nor do you need to implement any ClickListeners.

1. Choose an appropriate layout for your navigation bar
2. Create an instance of your layout in `createNavigationLayout()`-method
3. Create four buttons and add them to the layout
4. Align the buttons so that three of the buttons are to the left and one is to the right
5. Add a small spacing between the buttons

Book of Vaadin
Layouts: 223
Layout cell alignment: 264
Layout cell spacing: 267
CustomComponent: 221

Exercise 3: Application layout

In this exercise you will learn to create a typical layout structure for a Vaadin application. The goal is to create an application layout containing a header, a footer and a main area that consists of a navigation area and a content area. The screenshot below illustrates the desired result.

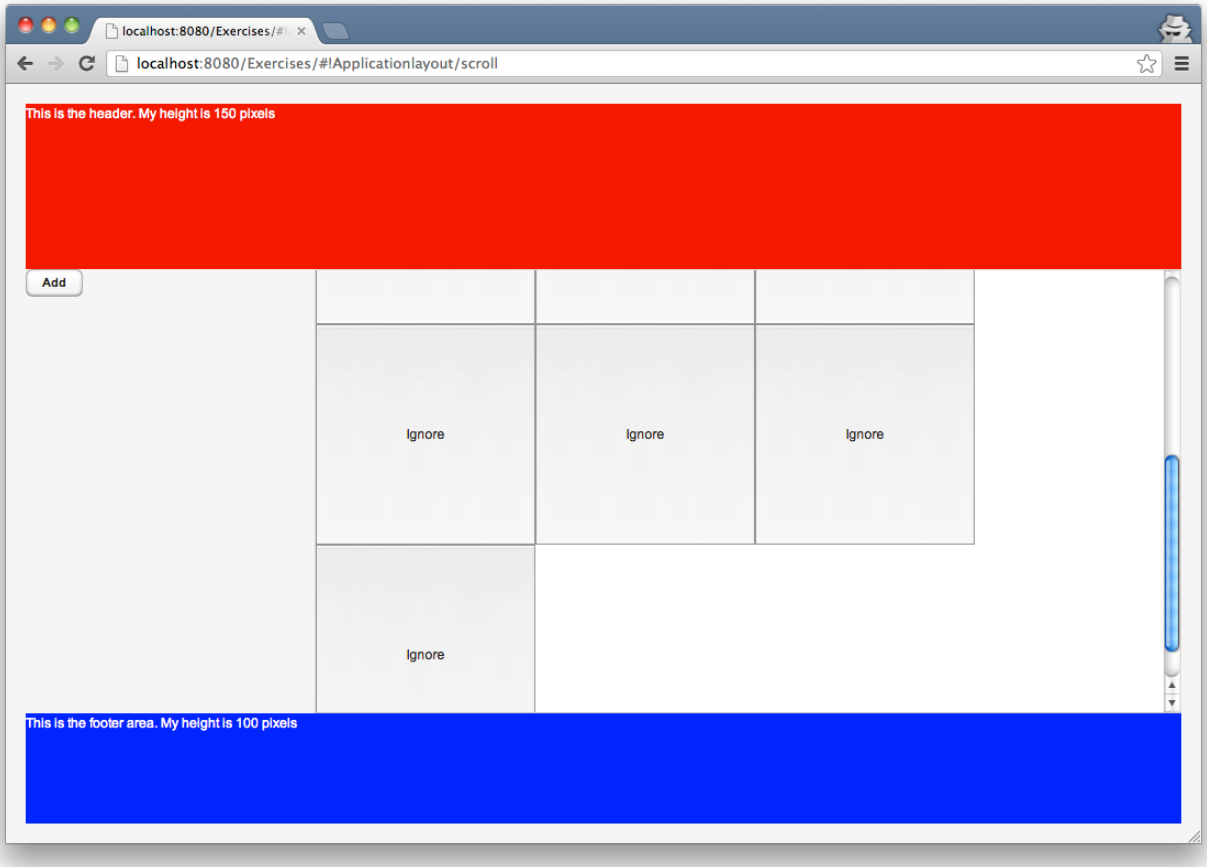
The header marked with a red color should be 100% in width and 150px in height. The footer (in blue) should be 100px in height and 100% in width. The area between the header and the footer should take up the rest of the space in the application. The navigation area (in yellow) should be 25% of the main areas width, while the content area (in green) should be 75% of the content area's width.

The stub application contains four labels each with a caption and the corresponding color. Your task is to complete the layout. In order to complete the application layout, you need to add one extra layout to the application and adjust the size and expand ratios of the components.

Book of Vaadin
Layouts: 223
Layout cell alignment: 264
Sizing components: 119

Exercise 3 continued: Enabling scrollbars

You can continue with this task once you've completed the first part of the application layout. This exercise's purpose is to enable scrollbars in the content area. Start by replacing the content label with a component container, that enables scrolling. After this, replace the navigation label with a button. Implement a click listener which will add a new NativeButton to the content area. You can use the `createButton()` helper method for creating the buttons for the content area.



Note that if I resize the window, the amount of buttons shown on one row in the content area should adjust to my browser size so, that at any given time, the maximum amount of buttons are shown on one row.

The challenges of this task is to figure out how to enable scrollbars and how to make the buttons wrap according to the browser size.

Book of Vaadin
Chapter 6: Managing layouts: 223-272

Exercise 4: Binding properties

The goal with this exercise is to demonstrate how you can leverage Vaadin's data model in order to automatically keep your data up-to-date in your views. A `Property` in Vaadin contains some sort of a value. If you change this value, the Property object will send out an event letting listeners know, that the value has changed. You can bind properties to different components (those that implement `Property.Viewer` interface) and those components get their values from the property. In other words, if you change the value of a property, the component's value also changes. Remember, `Field` extends the `Property` interface, so Fields can be used as any other property.

In this exercise, you should create a `Slider` component whose minimum value is 0 and maximum value is 100. Create a `Label` and bind the slider to the label so, that when the slider's value changes, its numeric representation is shown in the label. **Do not use ValueChangeListeners!**

To bind a property to a Property.Viewer, call
`component.setPropertyDatasource(property)`

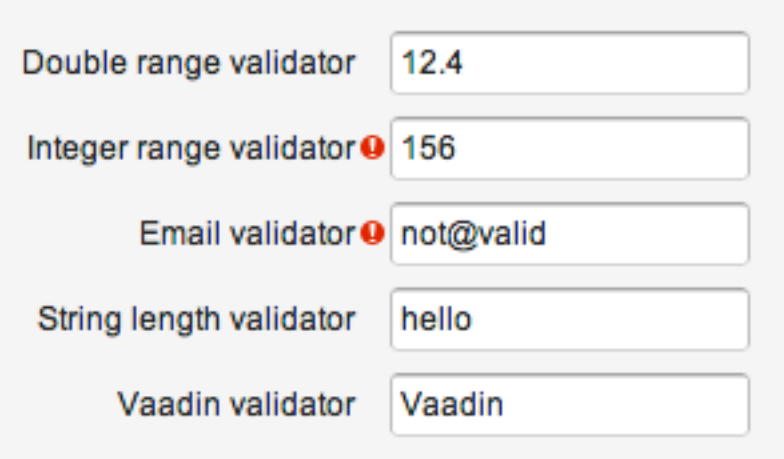
Bonus: If you are quick with this exercise, try adding a `ProgressBar` component to the layout. ProgressBar is a visual representation of a progress. It takes as an argument a float value between the value 0 and 1. In other words, if we want the ProgressBar to be halfway, the its value needs to be 0.5.

Bind the Slider to the ProgressBar, so that if you choose the value 75 in the Slider, then the ProgressBar would be at the state 75%.There is one problem yet to solve and that is that Slider handles type Double while ProgressBar expects a Float. You need to implement your own `Converter` which converts the data model value Double into the presentation type Float (75d => 0.75f).

Book of Vaadin:
Binding components to data; Properties: 314
Converting Between Property Type and Representation: 317

Exercise 5: Validation

An essential part of receiving user entered data is validating the input values. In this exercise we practice applying validators on fields.



Create a view having five `TextField` components. Each `TextField` should have its ow validator. The validators are: `DoubleRangeValidator`, `IntegerRangeValidator`, `EmailValidator`, `StringLengthValidator` and Custom. For the `DoubleRangeValidator`, add the validator to the TextField which show a validation error in case the user enters something that is not of type double or the value is not between 1 and 100. Same logic applies for `IntegerRangeValidator`. For the Custom validator, implement the Validator interface to validate that the user has entered the text “Vaadin”.

Note that the value of a TextField is always a String. If you enter the value “54”, it is still a String and will not pass the Integer/DoubleRangeValidator. For the validator to pass, you’ll first have to apply a converter to the TextField. The appropriate converters are `StringToIntegerConverter` and `StringToDoubleConverter`.

Validators are *added* by calling `field.addValidator(<validator>)`

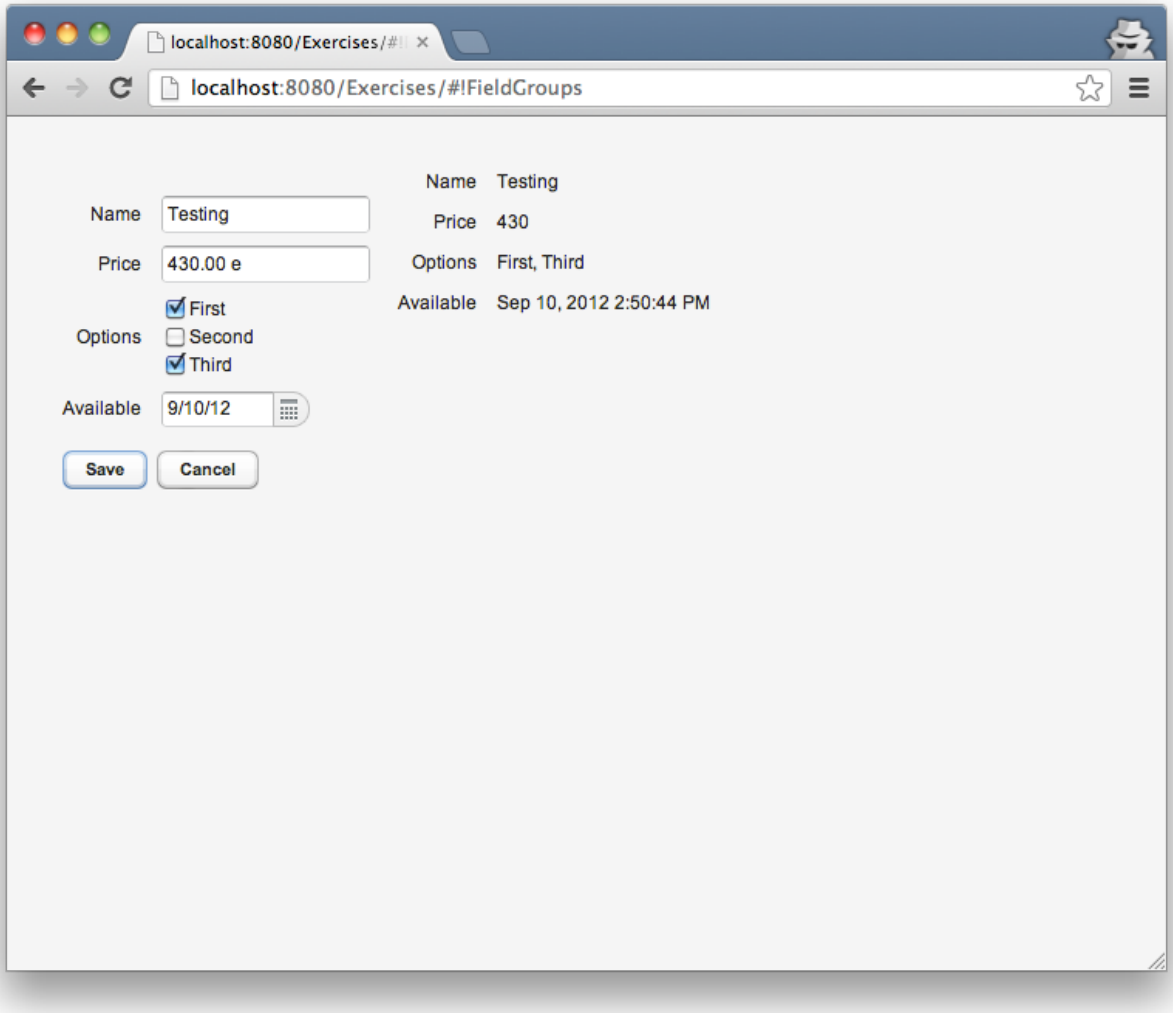
Book of Vaadin
Field Validation: 126
Converting Between Property Type and Representation: 317

Exercise 6: FieldGroups

In this exercise we want to create a form for editing a `Product` entity. Your task is to create the layout for the form and then use `FieldGroup` to bind the Product object to the form's layout. Selection component called `OptionGroup` should be used for options. When you click on save, the form's values should be committed into the Product property. If you click on cancel, any changes in the form should be reverted.

Next to the form is a layout that represent the actual values in the Product object. Bind the product object's properties to the labels so that they always show the most up-to-date information about the actual Product object's values.

Bonus task: If you are quick with this exercise, try implementing a Converter that allows you to enter currencies to the Price field. In the application below, I can enter euro values with either the postfix "e" or "EUR" and it will be correctly interpreted as a double value.



Book of Vaadin
DateField: 148
OptionGroup: 168
Binding Fields to Items: 324
Converting Between Property Type and Representation: 317

Exercise 7: Populating a container

In this exercise we practice using Vaadin's data model, more specifically, using containers. The exercise stub contains a table that is populated by a container. Your task is to initialize a `IndexedContainer`, containing two properties: name and sector.

We want to show three rows in the table. Each row in a table is represented by an item in the data model. Populate the container with the following data:

name	sector
John Doe	Private customer
Vaadin	Company
EFF	Organization

Properties are defined by calling
`container.addContainerProperty(Object propertyId, Class<?> propertyType, Object defaultValue)`

To add an item, call `container.addItem(Object itemId)` which will return you and Item object. You can also use `container.addItem()` without any parameters, but it will create an item instance for you and automatically assign an item id. The **item id** is returned in this case.

To get an **item** instance from a container, call `container.getItem(Object itemId)`

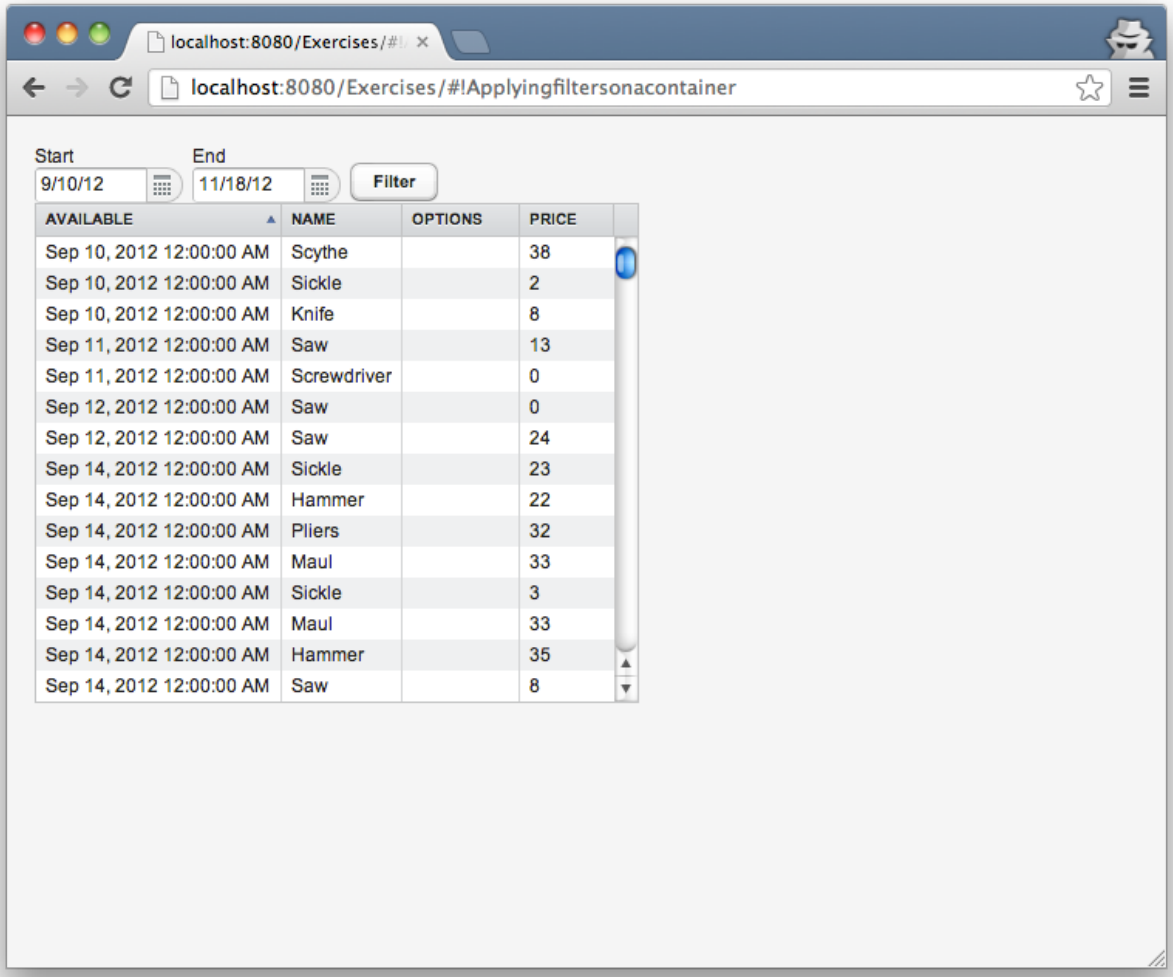
To get a property from an item, call `item.getItemProperty(Object propertyId)`

Book of Vaadin
IndexedContainer: 335

Exercise 8: Filtering a table

The target of this exercise is to practice filtering values in a table. The view should have two `DateFields` in which you select a date range. When the filter-button is clicked, the Table's content should be filtered so, that only rows where the "available" property is between the given range are visible.

Note that filtering is done in the container, not in the table!



The layout for the view is not build for you, so you'll have to start by creating the view layout, it shouldn't be too hard for you at this point.

Once you've created the layout, start implementing the custom DateRangeFilter, there is already a simple stub for that one in the view.

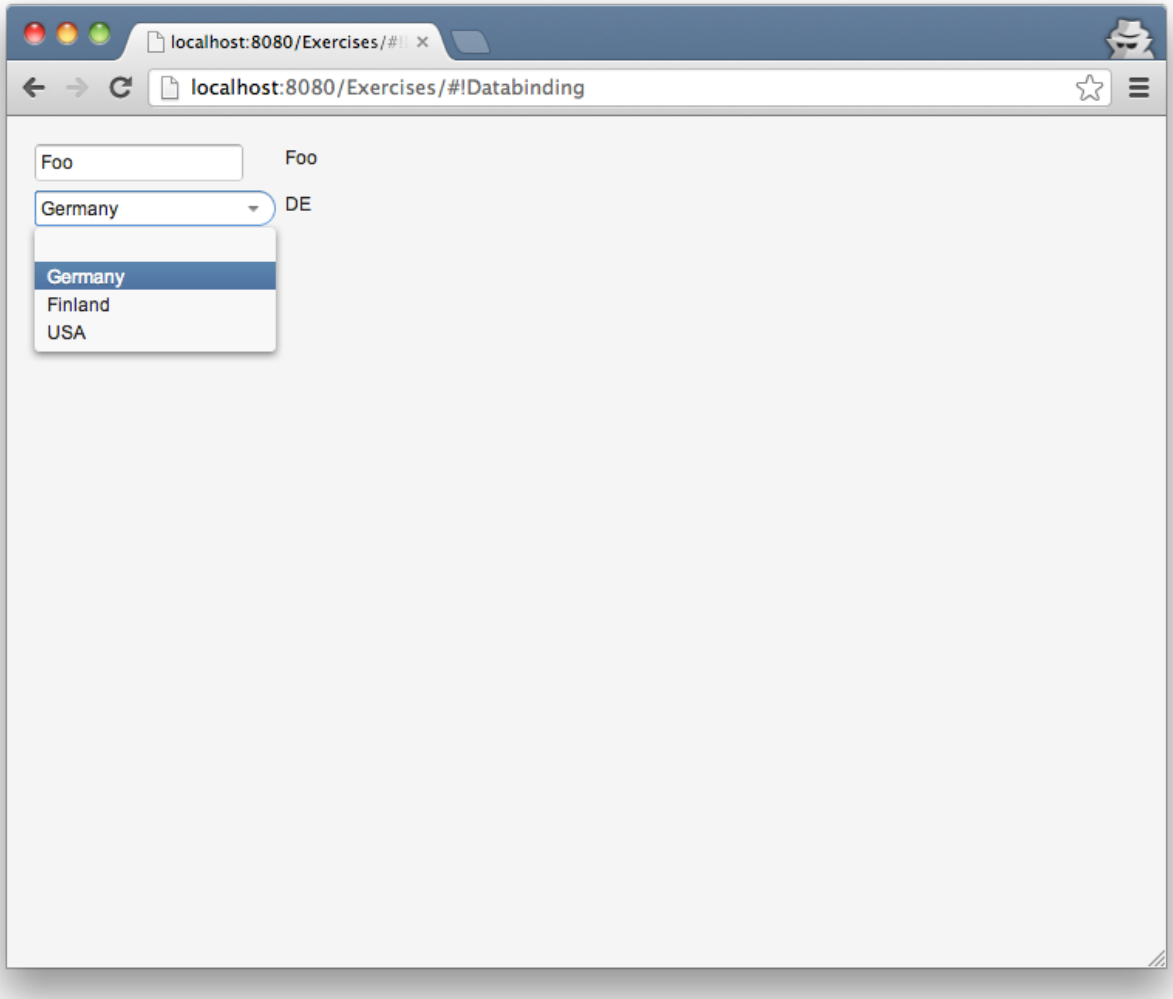
In principe, the DateRangeFilter is simply a composition of existing Filters. Try taking a look at the And, Or and Compare filters. Here's a hint ;)

```
new Compare.GreaterOrEqual("available", startDate);
```

Book of Vaadin
Filterable Containers: 341

Exercise 9: Data Binding

The idea of this exercise is to have a small recap of Vaadin's data binding features and to understand the difference between property data sources and container data sources.



Create a view that has two `Labels`, one `TextField` and one `ComboBox` in a `GridLayout`. When typing a value to the TextField, its value should automatically be shown in the Label.

The ComboBox should have three values, "Finland", "Germany" and "USA". When "Finland" is selected, the text "FI" should be shown in the Label, DE for Germany and US for USA, respectively.

This is exercise is all about Vaadin's data model, **do not use ValueChangeListeners**!

To bind a property to a Property.Viewer, call
`component.setPropertyDatasource(property)`

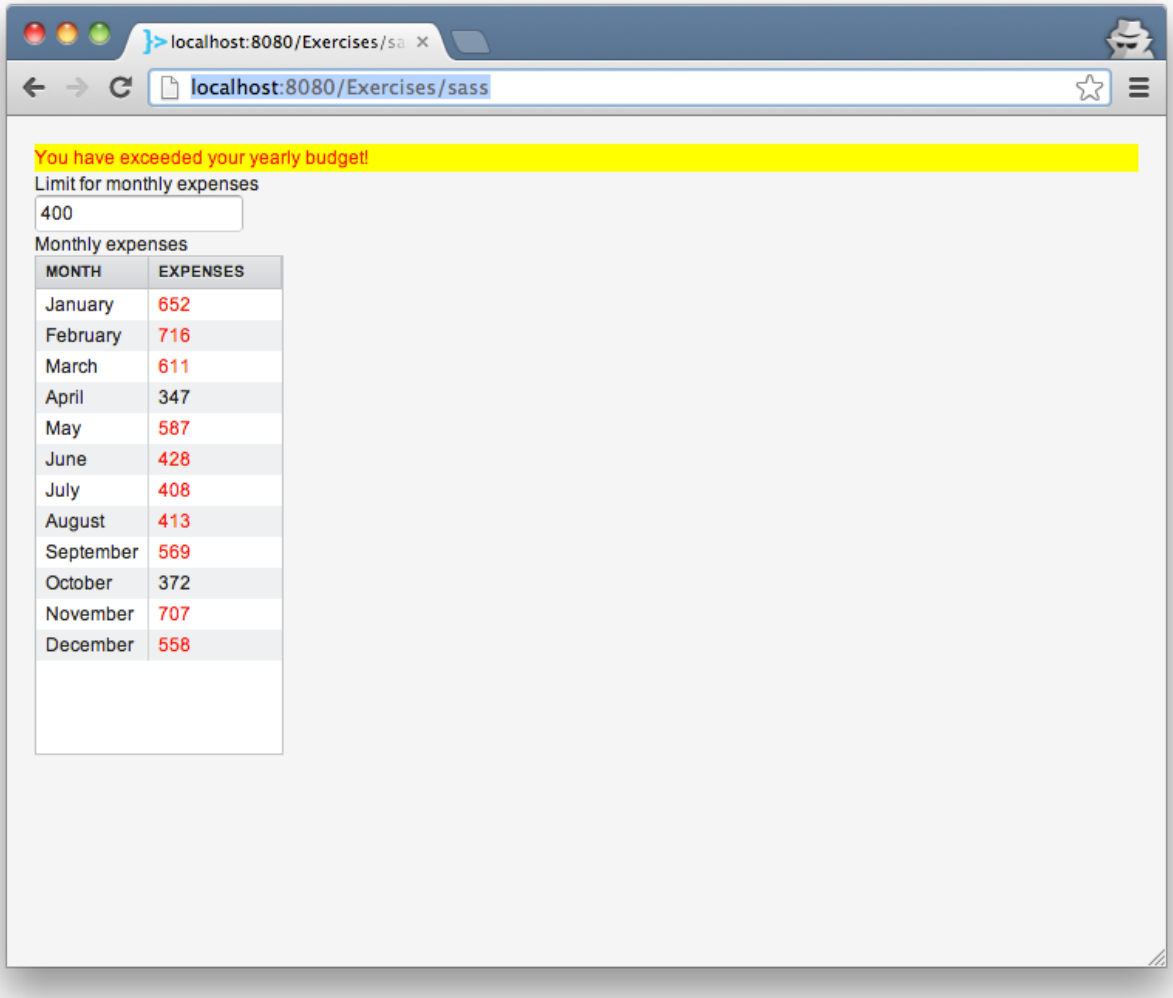
Book of Vaadin
Binding components to data; Properties: 314
Selecting Items: 157

Exercise 10: Creating a theme

In this exercise you'll create a simple, custom theme for your application. The ex10 package will contain a separate Vaadin UI, your task is to create a theme for the application.

The application has a TextField and a Table. The table contains a listing of expenses for 12 months. The textfield is for entering a monthly expense limit. If a month's expenses listed in the table are higher than the expense limit defined in the textfield, then the expense cell's text should be highlighted with a red color.

If the average monthly expenses exceeds the given limit, a warning label will appear. The warning label should have a yellow background and a red text. Your customer is not completely sure about the red color for the text, all he knows is that he wants the text in the table and in the warning label to be of the same color. To make it easier to change the color later on, make it a scss variable.



You'll need to do the following things

1. Create a new theme. Do this by right clicking on the SassTheme class, choose New > Vaadin > Vaadin Theme. In the wizard, select only the SassTheme UI (unselect exercises), name the theme "expenses".

2. Add a style name to the warning label, so that you can apply the CSS on only that label.
3. Implement a CellStyleGenerator in order to define a style name for the cells

Hints: Remember how CSS attribute names are defined, v-component-attribute-name, use for example Firebug or Chromes developer tools to inspect the code, in order to find out the appropriate style names to use.

Book of Vaadin
Themes: 289