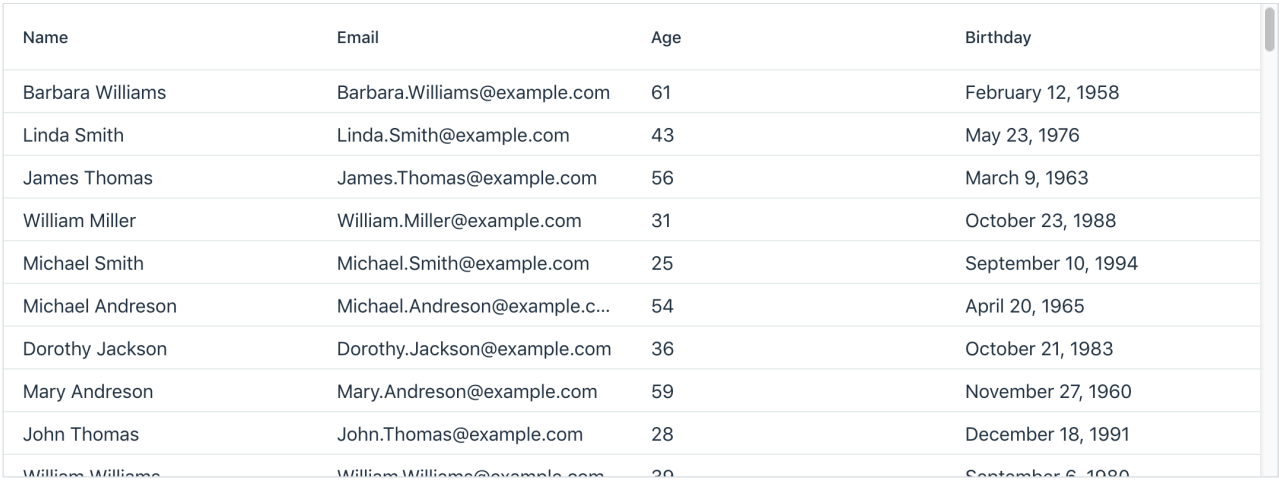


Exercises

Exercise 1: Populating a Grid

The exercise stub contains a Layout that should contain a Grid. Your task is to create a Grid containing four columns: name, email, age and birthday. You can get a List of test data from the PersonService class to populate the Grid.



Name	Email	Age	Birthday
Barbara Williams	Barbara.Williams@example.com	61	February 12, 1958
Linda Smith	Linda.Smith@example.com	43	May 23, 1976
James Thomas	James.Thomas@example.com	56	March 9, 1963
William Miller	William.Miller@example.com	31	October 23, 1988
Michael Smith	Michael.Smith@example.com	25	September 10, 1994
Michael Andreson	Michael.Andreson@example.c...	54	April 20, 1965
Dorothy Jackson	Dorothy.Jackson@example.com	36	October 21, 1983
Mary Andreson	Mary.Andreson@example.com	59	November 27, 1960
John Thomas	John.Thomas@example.com	28	December 18, 1991
William Williams	William.Williams@example.com	39	September 6, 1980

You have a choice how to create the Columns in your Grid, between these two:

```
Grid<Person> grid = new Grid<>(Person.class)

// OR

Grid<Person> grid = new Grid<>();

grid.addColumn(Person::getAge);

...
```

Both options are equally valid, but remember that the first version might not order the columns like you want.

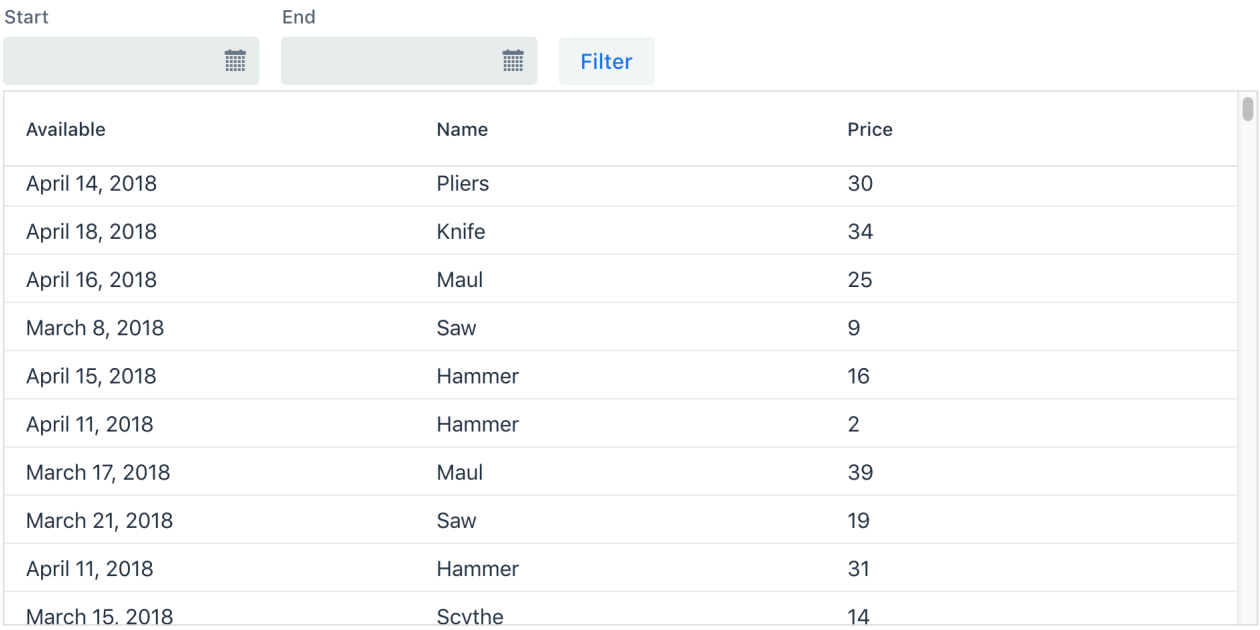
Exercise 2: Filtering a DataProvider

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

Note that filtering should be done in the DataProvider, not in the grid!

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

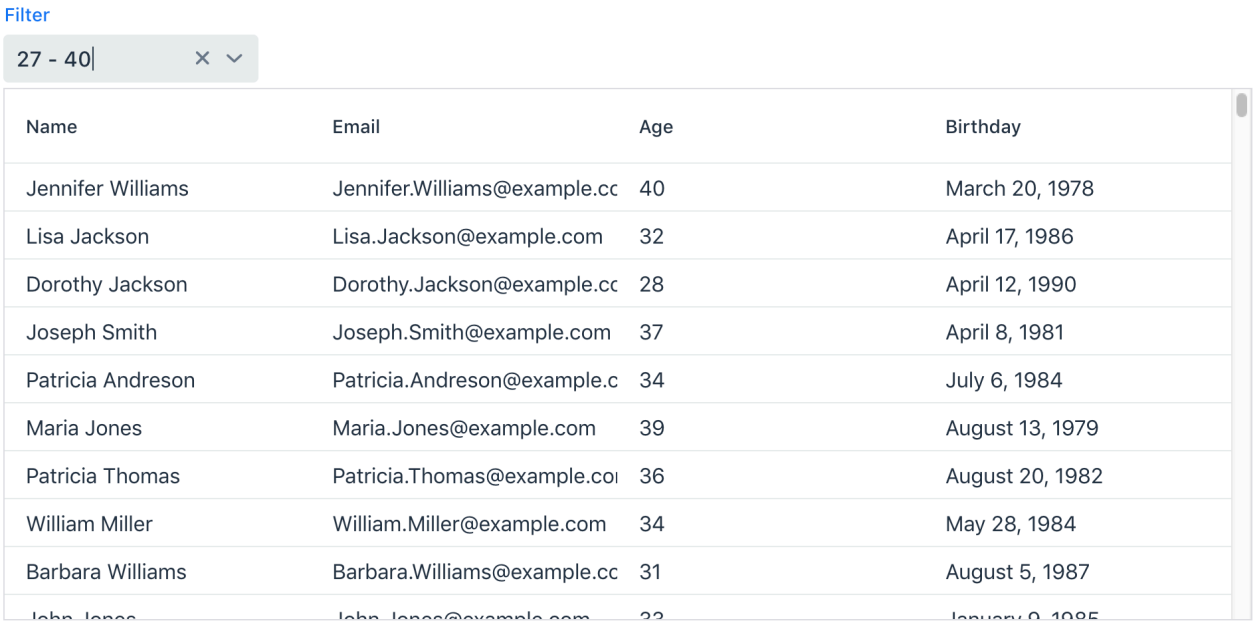
Once you've created the layout, start implementing the filtering with the help of the `filterProduct` method. As a hint, you should use `ListDataProvider.setFilter(SerializablePredicate<T> filter)`.



REMEMBER that your filter should be able to handle 4 situations; both dates being null, both dates being non-null, and either date being null and the other not. In other words, the filter can be open-ended in either direction. You can handle each case, in turn, in the `filterProduct` method.

Exercise 3: Filtering a back end data provider

In the third exercise we will create a filtering BackEndDataProvider for a Grid. You are provided with a ComboBox, a Grid, and a Service for data. Your task is to create a DataProvider that fetches items lazily form the Service. In addition, the provided ComboBox should be able to filter the content of the Grid; when a user selects an age group, the grid should refresh itself with persons only from that group. This is done by providing a filter to the back end.



The lazy DataProvider can be done in the following steps:

1. Create the filtering call back provider with `DataProvider.fromFilteringCallbacks()`. The first parameter is a call to `PersonService.getPersons()`, the second to `PersonService.countPersons()`. Save the resulting data provider to a `CallbackDataProvider` variable.

2. Add filtering with `CallbackDataProvider.withConfigurableFilter()`. You don't need any parameters for this one. Save the resulting `ConfigurableFilterDataProvider` to a variable as your actual data provider. It should have the following type:

   ```
   ConfigurableFilterDataProvider<Person, Void, AgeGroup>
   ```

3. call `grid.setDataProvider()` with your `ConfigurableFilterDataProvider`.

4. Add a ValueChangeListener to the ComboBox that gets the value from it and calls `dataProvider.setFilter(selectedAgeGroup)`