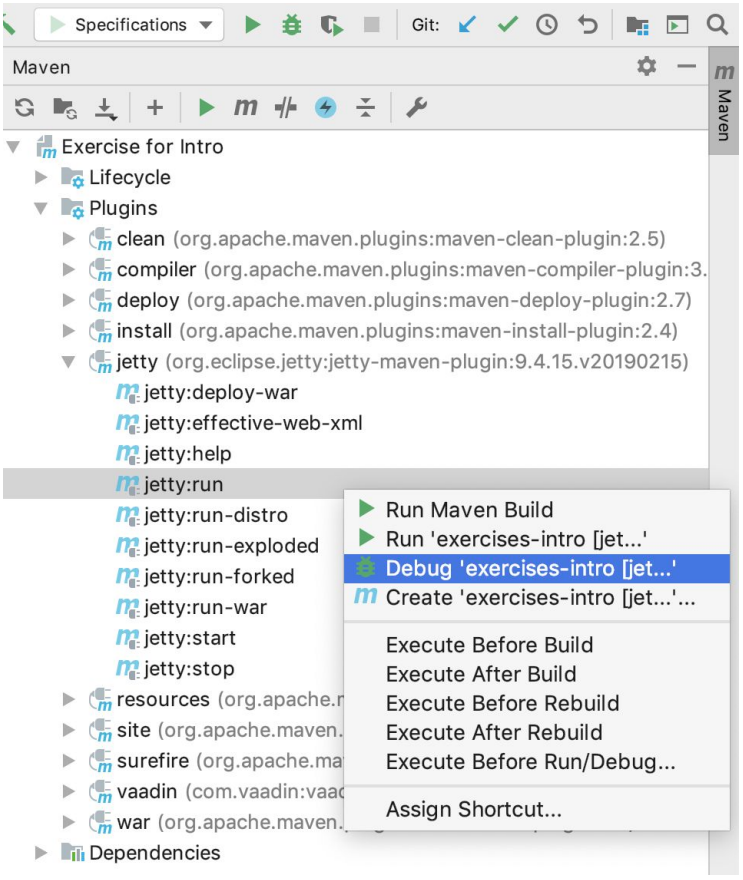
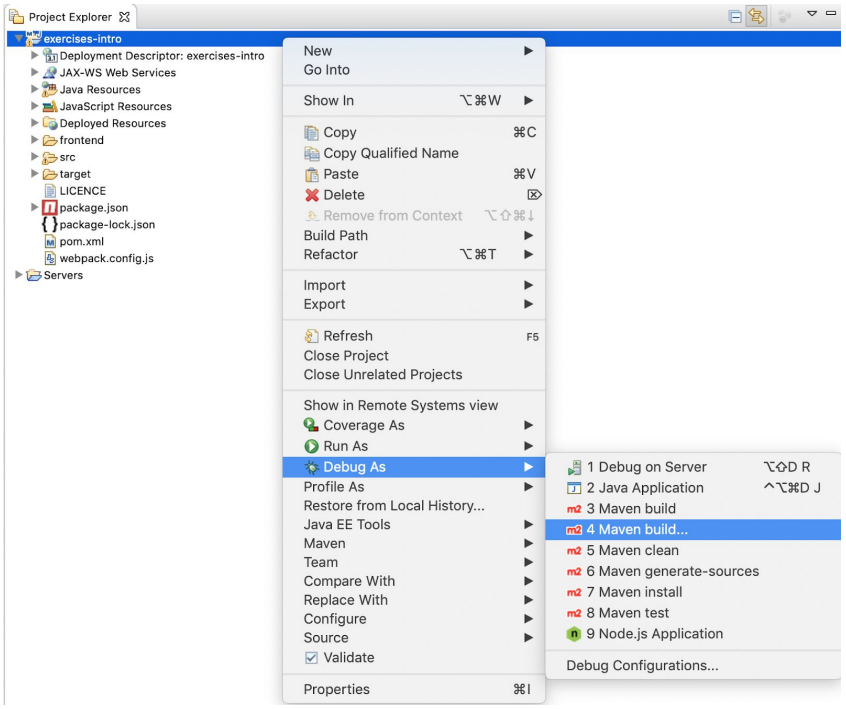
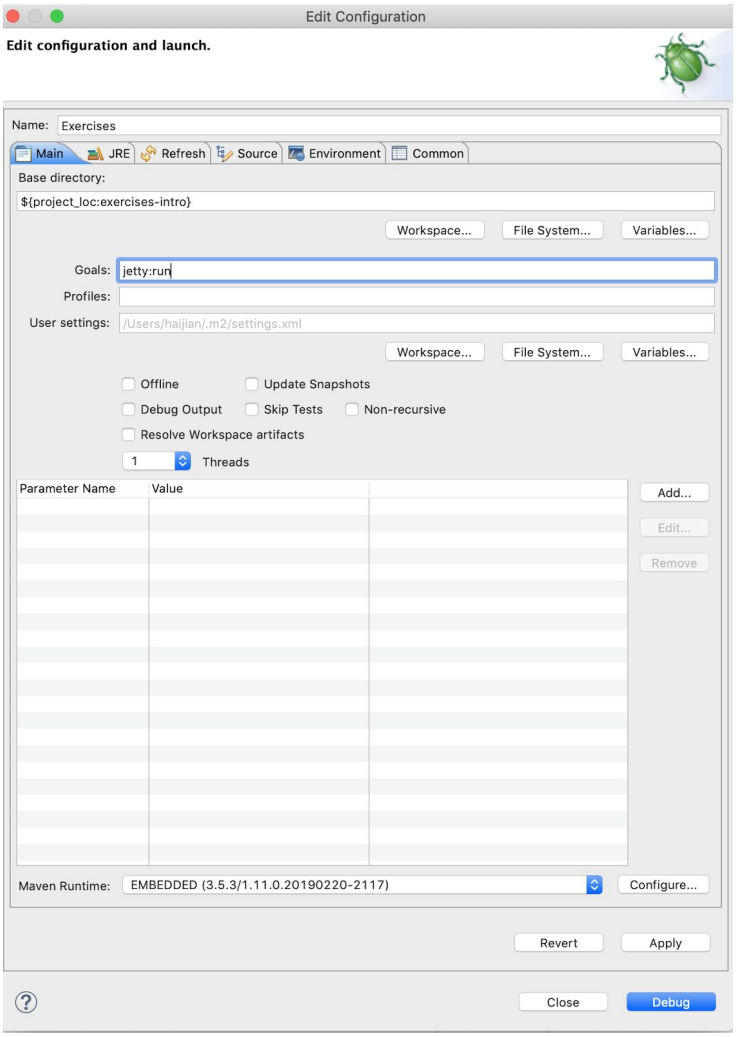
Instructions

1. Import the Maven project to your Favourite IDE.
2. Run the Maven goal jetty:run in the exercises submodule
    a. If you have command line Maven installed you can run `mvn jetty:run` in terminal OR
    b. Debug or run the `jetty:run` goal in IntelliJ



c. Debug or run the application in Eclipse

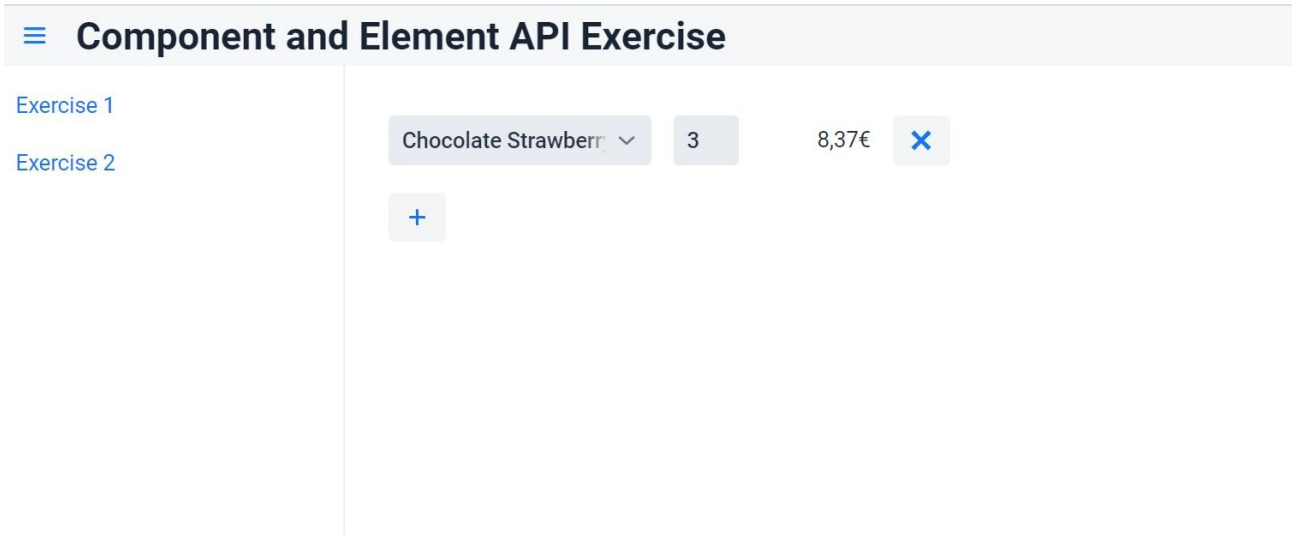




3. Go to localhost:8080

Exercise 1: Order Editor

Sometimes, you would need to create your own custom component, like this order editor element you are going to implement for this exercise. The UI looks like below:



Users can order different items, add a new order item with the '+' and remove an item with the 'x' button. Users can specify which product to order and how many. The price will be shown for each item and the total price is shown at the bottom. To implement this, there's the `OrderEditor` component which manages the entire order (including total cost) and an `OrderItemEditor` for each item. At the start, the `OrderItemEditor` is mostly empty.

You should modify `OrderItemEditor` so that it has a combo box to select a product, a text field to specify the amount, a Div to show the price for the item total, and a remove button. `OrderEditor` has a list of `OrderItemEditors` and an `Add` button, where user can add more order items to the order - you should add a keyboard shortcut to make adding new items easier. Here are the steps to follow:

1. Populate the `OrderItemEditor` with a combobox, a text field, a div, and a button.
2. The product combobox can be populated with some demo data by calling DemoDataGenerator.createProducts(5);
3. For the close button, you can use the font icon VaadinIcon.CLOSE.create(). In the button click listener, remove the OrderItemEditor from its parent.
4. Add value change listeners to product combobox and the amount text field. You can just call the updatePrice method.
5. By now, when we change the product or change the amount, the price for the particular item should change. But the total price at the bottom doesn't change, because we haven't informed the parent OrderEditor about the changes yet. Let's start working on that.
6. To inform OrderEditor about the change, we can fire an OrderItemChangeEvent. First create a OrderItemChangeEvent class which extends from ComponentEvent.

Then fire the event in the updatePrice method as well as in the click listener for the close button. Finally implement an addOrderItemChangeEventListener method by calling addListener.

7. In OrderEditor, use addButton.addClickShortcut to create a keyboard shortcut with control+enter.
8. In Exercise1View, get the Style object from totalPrice Div and use it to make the Div more visible. For example, change the color to red.
9. Now everything should work, as other parts of the OrderEditor have been preimplemented.

Exercise 2: Stepper

In this exercise, you are supposed to use the Element API (no Components!) to implement a Stepper component to control the number in a badge. You can follow the steps described below:

1. You should use Polymer `paper-badge` element to display the badge. Since it's a custom element, you need to import it first:
   - Add `@NpmPackage(value = "@polymer/paper-badge", version = "3.0.1")` to `Exercise2View`
   - Add `@JsModule("@polymer/paper-badge")` to `Exercise2View`
2. Create a `span` Element which shows some text like "Inbox"
3. Create a new `paper-badge` Element
4. Via the `element.getStyle()` object, set the badge's custom css property `"--paper-badge-marginleft"` to be `"20px"`.
5. Wrap the `span` and the `paper-badge` element together into a new `div` element. Then append the `div` element into the root element of `Exercise1View`.
6. Now implement the Stepper, which contains an `input` element, where user can input the value directly. Add a "+" button to increase the value and a "-" button to decrease the value.
7. Create an `input` element. Listen for the `value` property change, and in the listener, set the `label` property of the `paper-badge` element to be the `value` of the input element.
8. Create the "+" `button` element, react to the click event to increase the 'value' property of the `input` element. 8. Similarly, create the '-' button.

Bonus Task: Listen for the 'wheel' event for the input element, so that the value can be increased/decreased when user scrolls the mouse wheel.