7. Exemplary Solutions: Seaside: Composition

Exercise 7.1

```
STMainFrame class >> canBeRoot
    ^true

STMainFrame >> renderContentOn: html
    html div class: #header; with: [
        html div class: #theater; with: [
            html text: STTheater default name].
        html div class: #season; with: [
            html text: STTheater default season] ].
    html div class: #headerLine; with: [html break].
    html div class: #menu; with: [html break].

STMainFrame >> style
^'.theater {
    font-size: 30px;
    font-weight: bold;
    color: #FFCC00;
}
.season {
    font-size: 18px;
    font-weight: bold;
    padding-bottom: 5px;
}
.header {
    padding: 8px;
    background-color: #006699;
}
.headerLine {
    background-color: #000066;
}
.menu {
    position: absolute;
    top: 110px;
    left: 20px;
    width: 120px;
    padding: 5px;
    border-wdith: 1px;
    border-style: dotted;
}
.main {
    position: absolute;
```

```
    top: 110px;
    left: 180px;
    width: 80%;
    padding: 5px;
}
```

Exercise 7.2

```
WAComponent subclass: #STMainFrame
    instanceVariableNames: 'child'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Tutorial-Theater-View'

STMainFrame >> initialize
    super initialize.
    self buyTicket.

STMainFrame >> buyTicket
    child := STBuyTicketTask new.

STMainFrame >> children
    ^ Array with: child

STMainFrame >> renderContentOn: html
    ...
    html div class: #menu; with: [
        html anchor callback: [self buyTicket]; with: 'Buy ticket'].
    html div class: #main; with: child.
```

Exercise 7.3

When using the back button after clicking on *Buy Ticket*, the application wants to go back to a state in which the child component, STBuyTicketTask, has not yet been registered. Thus Seaside cannot locate this child component and raises an error as the two components, the main frame and the buying ticket component are not in sync. To synchronize them, you can answer in the #states method a list of components that should be in sync. For more information, visit `http://book.seaside.st/book/components/calling/back-button`.

```
STMainFrame >> states
    ^Array with: self
```

Exercise 7.4

A very simple solution is to use the play and show chooser to select from which play and show the customer wants to change tickets. Then we can use the ticket printer to select all the tickets sold for this specific show. For this we have to extend the ticket printer with check boxes allowing the customer to select the right tickets. Finally, we again use the show chooser to select another show of the previously selected play and move the selected tickets there. Of course we do not pass the show to the show chooser from which we want to move tickets. Also we just allow the user to select shows that have enough seats available to move all the selected tickets to that show. Note that in this exercise we assume that tickets are not directly assigned to a specific customer; in reality this would be different of course.

In the model we just add a method #removeTicket: to class STShow. Very important, we also write a test for this method, #testShowRemoveTicket.

Please find in the following the respective code in STChangeTicketTask and in the model. We also show the adaptations in the STTicketPrinter.

```
WATask subclass: #STChangeTicketTask
    instanceVariableNames: 'tickets fromShow toShow'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Tutorial-Theater-View'

STChangeTicketTask >> go
    | play |
    play := self call: (STPlayChooser new plays: STTheater default plays).
    fromShow := self call: (STShowChooser new shows: play shows).
    tickets := self call: (STTicketPrinter new tickets:
            (fromShow tickets asSortedCollection: [:a :b | a id <= b id])).
    toShow := self call: (STShowChooser new shows:
            (play shows reject: [:s |
                s = fromShow and: [s placesFree < tickets size]])).
    self moveTickets.

STChangeTicketTask >> moveTickets
    self tickets do: [:ticket |
        self fromShow removeTicket: ticket.
        self toShow addTicket: ticket]

STTicketPrinter >> renderContentOn: html
    html form: [
        self tickets withIndexDo: [:ticket :i |
            html checkbox onTrue: [self addTicket: ticket] onFalse: [].
                html div class: #ticket; with: [
        ...].
        html submitButton value: 'Ok'; callback:
            [self answer: selectedTickets] ].
```

```
STShow >> removeTicket: aTicket
    aTicket setShow: nil.
    ^ self tickets remove: aTicket.

STModelTests >> testShowRemoveTicket
    self assert: (show addTicket: ticket) = ticket.
    self assert: ticket show = show.
    self assert: show placesSold = 1.
    self assert: show placesFree + show placesSold = show placesTotal.
    self assert: (show removeTicket: ticket) = ticket.
    self assert: ticket show isNil.
    self assert: show placesSold = 0.
    self assert: show placesFree + show placesSold = show placesTotal.
```

Exercise 7.5

```
WAComponent subclass: #STShowReport
    instanceVariableNames: 'batchedList table'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Tutorial-Theater-View'

STShowReport >> initialize
    super initialize.
    self initializeBatchedList.
    self createTable.

STShowReport >> initializeBatchedList
    batchedList := WABatchedList new
        items: self allShows;
        batchSize: 10;
        yourself

STShowReport >> allShows
    ^STTheater default shows asSortedCollection:
                        [:a :b | a timestamp <= b timestamp]

STShowReport >> children
    ^ Array with: table with: batchedList

STShowReport >> createTable
    table := WATableReport new
        rowPeriod: 1;
        rowColors: #( 'lightgrey' 'white' );
```

```
        columns: (OrderedCollection new
            add: (WAReportColumn renderBlock:
                    [:e | e play title] title: 'Play');
            add: (WAReportColumn renderBlock:
                    [:e | e play kind] title: 'Kind');
            add: (WAReportColumn renderBlock:
                    [:e | e play author] title: 'Author');
            add: ((WAReportColumn renderBlock: [:e | e] title: 'Timestamp')
                    formatBlock: [:e | e date asString, ' ', e time asString];
                    sortBlock: [:a :b | a timestamp <= b timestamp]);
            add: (WAReportColumn selector: #placesFree title: 'Free');
            add: (WAReportColumn selector: #placesSold title: 'Sold');
            add: (WAReportColumn selector: #placesTotal title: 'Total');
            yourself);
        yourself.

STShowReport >> renderContentOn: html
    table rows: batchedList batch.
    html render: table.
    html break.
    html render: batchedList.
```