6. Exemplary Solutions: Seaside: Components

Exercise 6.1

STBuyTicketTask class >> canBeRoot
  ^true

STBuyTicketTask >> go
  self inform: 'Hello World'

Exercise 6.2

WAComponent subclass: #STPlayChooser
  instanceVariableNames: 'plays'
  classVariableNames: ''
  poolDictionaries: ''
  category: 'Tutorial-Theater-View'

STPlayChooser >> initialize
  super initialize.
  self plays: OrderedCollection new.

STPlayChooser >> plays: aCollection
  plays := aCollection asOrderedCollection

STPlayChooser >> plays
  ^ plays

STBuyTicketTask >> go
  self call: (STPlayChooser new plays: STTheater default plays)

Exercise 6.3

WAComponent subclass: #STPlayChooser
  instanceVariableNames: 'plays'
  classVariableNames: ''
  poolDictionaries: ''
  category: 'Tutorial-Theater-View'

STPlayChooser >> renderContentOn: html
  self plays do: [:play |
    html div class: #play; with: [
      html div class: #head; with: [
        html text: play title; space;
        text: '(',play kind, ',')'; space;
Exercise 6.4

STPlayChooser >> renderContentOn: html
  self plays do: [:play |
    html div class: #play; with: [
      html div class: #head; with: [
        html anchor callback: [self answer: play];
        with: [html text: play title].
      html space
        text: '('; play kind; ')'; space;
      html text: ' - '; space;
      html text: play author].
      html div class: #body; with: [html text: play description] ] ]
  ]

STBuyTicketTask >> go
| answer |
  self inform: 'Selected play: ', answer title.

Exercise 6.5

STPlayChooser >> initialize
  super initialize.
  self plays: OrderedCollection new.
  self sortState: #title.
self sortOrder: #<=.

STPlayChooser >> toggleSortOrder
    sortOrder = #<= ifTrue: [sortOrder := #>=] ifFalse: [sortOrder := #<=].

STPlayChooser >> plays
    ^plays asSortedCollection: [:a :b |
        (a perform: self sortState) perform: self sortOrder
        with: (b perform: self sortState)]

STPlayChooser >> renderContentOn: html
    | sortTypes |
    sortTypes := #(#title #kind #author).
    html div class: #sort; with: [
        sortTypes do: [:sortType |
            html anchor callback: [self sortState: sortType.
                self toggleSortOrder];
            with: [ | textSelector |
                textSel := sortType = self sortState
                    ifTrue: [#strong:] ifFalse: [#text:].
                html perform: textSel with: sortType asString capitalized.
                html space] ].

Exercise 6.6

WAComponent subclass: #STShowChooser
    instanceVariableNames: 'shows selection'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Tutorial-Theater-View'

STShowChooser >> initialize
    super initialize.
    self shows: OrderedCollection new.
    self selection: nil.

STBuyTicketTask >> go
    | play |
    play := self call: (STPlayChooser new plays: STTheater default plays).
    self call: (STShowChooser new shows: play shows).

Exercise 6.7

STShowChooser >> shows
^ shows asSortedCollection: [:a :b | a timestamp <= b timestamp].

STShowChooser >> selectNext
| now nextShow |
now := TimeStamp now.
nextShow := self shows detect: [:show | show timestamp >= now] ifNone: [nil].
self selection: nextShow.

STShowChooser >> renderContentOn: html
html form: [
    html select
    class: #shows;
    list: self shows;
    selected: self selection;
    labels: [:each | each play title, ' - ', each date displayString,
             ' ', each time displayString];
    callback: [:show | self selection: show];
    size: 10.
    html break.
    html button value: 'Next'; callback: [self selectNext].
    html space.
    html submitButton value: 'Ok'; callback: [
        self selection ifNotNil: [self answer: self selection]
        ifNil: [self inform: 'No show selected'] ] ]

STShowChooser >> style
^`.filter {
    background: #eeeeee;
    padding: 5px;
    margin-bottom: 5px;
}
.shows {
    width: 500px
}

STBuyTicketTask >> go
| play answer |
play := self call: (STPlayChooser new plays: STTheater default plays).
answer := self call: (STShowChooser new shows: play shows).
self inform: 'Selected show: ', answer timestamp asString.

Exercise 6.8

STShowChooser >> initialize
    super initialize.
self shows: OrderedCollection new.
self selection: nil.
self startDate: nil.
self endDate: Date today + (Duration days: 30)

STShowChooser >> shows
| sorted |
sorted := shows asSortedCollection: [:a :b | a timestamp <= b timestamp].
^ self startDate ifNil: [sorted]
ifNotNil: [sorted select: [:show |
show date between: self startDate and: self endDate]].

STShowChooser >> dates
^Array streamContents:
[:stream |
0 to: 30*6 do: [:i |
stream nextPut: (Date today + (Duration days: i)) ] ].

STShowChooser >> renderContentOn: html
html form: [
  html div class: #filter; with: [
    html text: 'Filter from: '.
    html select
      list: self dates;
      selected: self startDate;
      labels: [:each | each asString];
      callback: [:date | self startDate: date];
      beSubmitOnChange.
    html text: ' to: '.
    html select
      list: self dates;
      selected: self endDate;
      labels: [:each | each asString];
      callback: [:date | self endDate: date];
      beSubmitOnChange.
    html space.
    html submitButton value: 'Update'
  ].
...
Exercise 6.10

WAComponent subclass: #STTicketChooser
  instanceVariableNames: 'show requiredTickets'
classVariableNames: '
  poolDictionaries: '
category: 'Tutorial-Theater-View'

STTicketChooser >> requiredTickets: anIntegerOrString
  requiredTickets := anIntegerOrString asInteger.

"The mutator of 'show' should not be called #show:
  as this method exists in superclasses"
STTicketChooser >> setShow: aShow
  show := aShow

STTicketChooser >> buyTickets
  self answer: (self show nextTickets: self requiredTickets)

STTicketChooser >> renderContentOn: html
  html form: [
    html table: [
      html tableRow: [
        html tableData: [html text: 'Free places:'].
        html tableData: [html text: self show placesFree asString] ].
      html tableRow: [
        html tableData: [html text: 'Required places:'].
        html tableData: [
          html textInput
            value: self requiredTickets;
            callback: [:value | self requiredTickets: value];
            size: 4 ] ] ]
    html submitButton value: 'Ok'; callback: [
self requiredTickets isInteger
   ifFalse: [self inform: 'Please enter a valid number of places']
   ifTrue: [self show placesFree < self requiredTickets
       ifTrue: [self inform: 'Not enough places available']
       ifFalse: [self buyTickets]]].

html cancelButton
value: 'Cancel';
callback: [requiredTickets := nil] ].

STBuyTicketTask >> go
| play show |
play := self call: (STPlayChooser new plays: STTheater default plays).
show := self call: (STShowChooser new shows: play shows).
self call: (STTicketChooser new setShow: show).

Exercise 6.11

WAComponent subclass: #STTicketPrinter
   instanceVariableNames: 'tickets'
   classVariableNames: ''
   poolDictionaries: ''
   category: 'Tutorial-Theater-View'

STTicketPrinter >> style
  \'.ticket {
  padding: 5px;
  border: 1px solid #000000;
  margin-bottom: 10px;
  width: 500px;
}
big {
  font-weight: bold;
}'

STTicketPrinter >> renderContentOn: html
   self tickets withIndexDo: [:ticket :i | html div class: #ticket; with: [
   html text: ticket show play theater name; break.
   html big: ticket show play title; space;
   big: '(' i asString, '/', self tickets size asString, ')';
   break.
   html paragraph: [
   html text: ticket show date asString;
   break;
   text: ticket show time].
html text: 'Ticket: ', ticket id asString] ].

STBuyTicketTask >> go
  | play show tickets |
  play := self call: (STPlayChooser new plays: STTheater default plays).
  show := self call: (STShowChooser new shows: play shows).
  tickets := self call: (STTicketChooser new setShow: show).
  self call: (STTicketPrinter new tickets: tickets).