14.4 Відображення HTML

Figure 14.12: RenderingDemo.

Ми вже бачили схожий пензель, використаний в прикладах лічильника та мультилічильника:

html horizontalRule.

На рисунку 14.12 зображено вивід багатьох базових пензлів, запропонованих Seaside. (Код для методу SeasideDemo >> renderContentOn: зазначений нижче знаходиться в пакеті PBE-SeasideDemo в проекті http://www. squeaksource.com/PharoByExample.) Коренева компонента SeasideDemo просто відображає підкомпоненти, які є зразками SeasideHtmlDemo, SeasideFormDemo, SeasideEditCallDemo і SeasideDialogDemo, які зазначено нижче.

**266**

Seaside на прикладі

14.4 Відображення HTML

SeasideDemo >> renderContentOn: html

html heading: 'Rendering Demo'.

html heading

level: 2;

with: 'Rendering basic HTML: '.

html div

class: 'subcomponent';

with: htmlDemo.

"render the remaining components ..."

Не забуваємо, що коренева компонента повинна оголосити всі похідні компоненти, в іншому випадку Seaside не відобразить їх.

SeasideDemo >> children

^ { htmlDemo . formDemo . editDemo . dialogDemo }

Зауважимо, що існують два різних способів інстанціації головного пензля. Перший спосіб – зазначити текст напряму, надіславши заголовок повідомлення.Іншй спосіб –інстанціювати пензель, надіславши заголовок повідомлення, а потім надіслати каскад повідомлень до пензля, зазначити його властивості і відобразити його. Багато різних пензлів можна використовувати двома способами.

**Важливо** Якщо надіслати каскад повідомлень до пензля, включаючи повідомлення *with:*, тоді *with:*повинно бути *останнім* повідомленням. with: зазначає контент і відображає результат .

В методі *renderContentOn*: вище перший заголовок є на першому рівні за замовчуванням. Ми явно зазначаємо рівень наступного заголовку до другого. Підкомпонента відображається HTML div з підкомпонентою CSS класу. (Більше про CSS в Розділі 14.5.) Зазначимо, що аргументи with: ключового повідомлення не повинне бути літерним рядком:воно може бути іншою компонентою, або навіть, як в наступному прикладі, блок, що містить наступні дії відображення.

SeasideHtmlDemo компонента демонструє більшість базових пензлів.Наступний код не повинен викликати багато питань.

SeasideHtmlDemo >> renderContentOn: html

self renderParagraphsOn: html.

self renderListsAndTablesOn: html.

self renderDivsAndSpansOn: html.

self renderLinkWithCallbackOn: html

На практиці доцільно розділяти великі методи для відображення на менші, як ми й зробили.

Порада: не пишіть весь код для відображення в одному методі.

Навпаки, розділіть його на допоміжні, використовуючи шаблон render\*On:. Всі повідомлення відображення потрапляють в протокол відображення. Не надсилайте renderContentOn: в свому коді, альтернатива –render:.

**267**

Seaside на прикладі

Розглянемо наступний код.Перший допоміжний метод, SeasideHtmlDemo>>renderParagraphsOn:, як згенерувати HTML-параграфи, постий і наголошений текст, та картинки. Зауважимо, що прості елементи в Seaside відоббражаються зазначаючи текст, який вони містять, напряму, хоча складні улументи уточняються за допомогою блоків. Це є просте вирішення правильно побудувати ваш код відображення.

SeasideHtmlDemo >> renderParagraphsOn: html

html paragraph: 'A plain text paragraph.'.

html paragraph: [

html

text: 'A paragraph with plain text followed by a line break. ';

break; emphasis: 'Emphasized text ';

text: 'followed by a horizontal rule.';

horizontalRule;

text: 'An image URI: '.

html image

url: self squeakImageUrl;

width: '50']

Наступний допоміжний метод, SeasideHtmlDemo>>renderListsAndTablesOn:, показує, як генерувати списки і таблиці. Таблиця використовує два рівні блоків для зображення кодного рядка та комірок в рядках.

SeasideHtmlDemo >> renderListsAndTablesOn: html

html orderedList: [

html listItem: 'An ordered list item'].

html unorderedList: [

html listItem: 'An unordered list item'].

html table: [

html tableRow: [

html tableData: 'A table with one data cell.']]

Наступний приклад показує, як можна визначити клас або id атрибути для div і span елементів(для роботи з CSS). Звичайно, повідомлення class: та id:може бути надіслано до інших пензлів, не тільки div і span. Метод SeasideDemoWidget>>style визначає, як HTML-елементи повинні бути відображеними (Розділ 14.5).

SeasideHtmlDemo >> renderDivsAndSpansOn: html

html div

id: 'author';

with: [

html text: 'Raw text within a div with id ''author''. '.

html span

class: 'highlight'; with: 'A span with class ''highlight''.']

Вкінці бачимо простий приклад посилання, за допомогою зв'язування простого callback та anchor. Clicking on the link will cause the subsequent text

**268**

14.4 Rendering HTML

to toggle between true and false by toggling the instance variable toggle- Value.

SeasideHtmlDemo >> renderLinkWithCallbackOn: html

html paragraph: [

html text: 'An anchor with a local action: '.

html span with: [

html anchor

callback: [toggleValue := toggleValue not];

with: 'toggle boolean:'].

html space.

html span

class: 'boolean';

with: toggleValue ]

Зауважемо, що дії повинні появлятися тільки в колбеках.

Код, виконаний під час відображення, не повинен змінювати стан аплікації!

**Форми**

Форми відображаються так само, як попередні приклади. Для прикладу розглянемо код компоненти SeasideFormDemo на рисунку 14.12.

SeasideFormDemo >> renderContentOn: html

| radioGroup |

html heading: heading.

html form: [

html span: 'Heading: '.

html textInput on: #heading of: self.

html select

list: self colors;

on: #color of: self.

radioGroup := html radioGroup.

html text: 'Radio on:'.

radioGroup radioButton

selected: radioOn;

callback: [radioOn := true].

html text: 'off:'.

radioGroup radioButton

selected: radioOn not;

callback: [radioOn := false].

html checkbox on: #checked of: self.

html submitButton

text: 'done' ]

Оскільки форма є складним об’єктом,вона відображається за допомогою блоку.Зуважимо, що всі зміни станів відбуваються в колбеках, і вони не є частиною відображення.

**269**

Seaside на прикладі

Існує одна особливість Seaside, що потребує окремого розгдяду - повідомлення on:of:. Наприклад, це повідомлення використовується для з’єднання поля для вхідного текстута заголовку змінної. Анкери та кнопки також підтримують це повідомлення. Перший аргумент – ім’я екземпляру, другий аргумент – об’єкт, якому він належить. Heading та heading повідомлення повинні бути розпізнані об’єктом, за домовленим павилом. Використовуючи on: #heading of: self, змінна заголовку автоматично обновляється кодного разу, коли користувач обновляє поле вхідних даних.

Те саме повідомлення використовується вдвічі більше в цьому випадку, щоб спричинити вибір кольору на HTML формі для обнови змінної color,та для поєднання перевірочного поля та змінної.Багато прикладів є у функціональних тестах Seaside.

**14.5 CSS: Каскадні таблиці стилів**

Каскадні таблиці стилів (http://www.w3.org/Style/CSS/), або коротко - CSS, Стандартний спосіб для розділення контенту і стилів. Seaside використовує CSS, щоб не захаращувати ваш код візуалізації.

Ви можете зазначити CSS таблицю стилів для ваших веб-компонент, визначивши стиль методу, який повинен повернути рядок, який містить CSSправила для цієї компоненти. Стилі всіх компонент ,відображених на веб-сторінці, з’єднані разом, отже,кодна компонента може мати свій стиль.Кращий підхід –визначити абстрактний клас для вашої веб-аплікації, що визначає загальний стиль для всіх похідних класів.

Насправді, для розгорнутих аплікацій звично визначати таблиці стилів за допомогою зовнішніх файлів. В цьому вигляд компоненти повністю відділений від функціональності.

Якщо ви вже знайомі з CSS,Тоді це все, що вам потрібно знати. В іншому випадку продовжимо короткий вступ по CSS.

Замість відображення атрибутів в параграфах та текстових елементах ваших веб-сторінок, CSS дозволяє визначити різні класи та елементи,та розмістити всу в окремій таблиці стилів.

**270**

14.5 CSS: Cascading style sheets

Іншими словами, таблиця стилів CSS містить набори правил, що зазначають, як форматувати HTML елементи.Кожне правило має дві частини. Селектор визначає, на які HTML-елементи діє правило, і є визначення, яке визначає кількість атрибутів для елемента.

SeasideDemoWidget >> style

^ '

body {

font: 10pt Arial, Helvetica, sans-serif, Times New Roman;

}

h2 {

font-size: 12pt;

font-weight: normal;

font-style: italic;

}

table { border-collapse: collapse; }

td {

border: 2px solid #CCCCCC;

padding: 4px;

}

#author {

border: 1px solid black;

padding: 2px;

margin: 2px;

}

.subcomponent {

border: 2px solid lightblue; padding:

2px; margin: 2px;

}

.highlight { background-color: yellow; }

.boolean { background-color: lightgrey; }

.field { background-color: lightgrey; } '

Попередній метод ілюструє просту таблицю стилів для відображення прикладу, показаного раніше на рисунку 14.12. Перше правило уточнює, який шрифт використовувати для тіла веб-сторінки. Наступні правила визначають властивості заголовків дрігого рівня (h2),таблиць (table) та даних таблиць(td).

Решта правил мають селектори, які поєднують HTML-елементи, що мають заданий класабо id-атрибути. CSS селектори для атрибутів класу починаються з a . id-атрибути з #. Головна різниця між класовими атрибутами та id-атрибутами – багато елементів можуть мати той самий клас,але кожен елемент може мати єдине id. Отже класовий атрибут, такий як виділення,може повторюватися декілька разів на сторінці, idповинне ідентифікувати унікальний елемент на сторінці , такий як меню, змінена дата, або автор .

**271**

Seaside на прикладі

Умови селектора можуть бути поєднані, тому селектор div.subcomponent буде поєднувати HTML-елемент, якщо він є div і має класовий атрибут - subcomponent.

Також є можливість визначати вкладені елементи, хоча це рідко потрібно. Наприклад, селектор p span поєднає span з параграфом (p) але не з div.

Існує багато книжок для вивчення CSS. Для демонстрації потіжної сили CSS, ми рекомендуємо вам переглянути CSS Zen Garden (http://www.csszengarden.com/), який показує, як однаковий контент може по різному відображатися, змінюючи таблиці стилів.

**14.6 Управління потоком контролю**

Seaside забезпечує легкий спосіб створення веб-аплікацій з нетривіальним потоком контролю. Існують два механізми, які ви можете використовувати :

1. Компонента може викликати іншу компоненту,надіславши *caller call: callee*. Колер тимачсово замінений на callee, поки callee не поверне контроль ,надіславши answer:. Зазвичай, колер - це self, але й може бути будь-яка наразі видима компонента.

2. Робочий процес визначається, як завдання. Це спеціальний вид компоненти, що є похідним класом WATask (замість WAComponent). Замість визначення renderContentOn: він визначає свій власний метод, який надсилає серію call:для активації різних підкомпонент.

**Виклик та відповідь**

Виклик та відповідь використовують для реалізації простих діалогів.

Розглянемо тривіальни приклад call: та answer: в прикладі відображення на рисунку 14.12. компонента SeasideEditCallDemo відображає текстове поле та посилання на редагування. Колбеки для посилань на редагування викликає новий екземпляр SeasideEditAnswerDemo, ініціалізований значенням текстового поля. Колбек також обновлюєце текстове поле результатом, який надісланий як відповідь.

SeasideEditCallDemo >> renderContentOn: html

html span

class: 'field';

with: self text.

html space.

html anchor

callback: [self text: (self call: (SeasideEditAnswerDemo new text:

self text))];

with: 'edit'

**272**

14.6 Керування потоком контролю

В ході роботи створюється нова сторінка, в якій SeasideEditCallDemo компоненту замінено на SeasideEditAn- swerDemo компоненту; батьківська компонента та інші не були змінені.

Важливо пам’ятати, що call: та answer: не можна використовувати під час відображення. Їх можна надіслати з колбеку, або з методу завдання.

SeasideEditAnswerDemo компонента також є дуже простою.Вона лише відображає форму з текстовим полем. Кнопка підтвердження зєднана з колбеком, який виведе фінальне значення текстового поля. SeasideEditAnswerDemo >> renderContentOn: html

html form: [

html textInput

on: #text of: self.

html submitButton

callback: [ self answer: self text ];

text: 'ok'. ]

Seaside відповідає за потоком контролю та правильне відображення всіх компонент.

**Допоміжні методи**

Оскільки call–answer діалоги є дуже поширені, Seaside забезпечує допоміжні методи для проблем в написанні компонент, наприклад, SeasideEditAnswerDemo. Збережені діалого зображені на рисунку 14.13. Ці допоміжні методи використовуються всередині Seaside-DialogDemo>>renderContentOn:

Повідомлення request: виконує викликкомпоненти, яка дозволить вам редагувати текстове поле. Компонента повертає редагований рядок. Необов’язковий ярлик значення по замовчуваннютакож можна зазначити.

SeasideDialogDemo >> renderContentOn: html

html anchor

callback: [ self request: 'edit this' label: 'done' default: 'some text' ];

with: 'self request:'. ...

Повідоилення inform: викликає компоненту, яка відображає повідомдення і чекає, поки користувач не натисне Ok. Викликана компонента повертає саму себе.

**273**

Figure 14.13: Some standard dialogs.

...

html space.

html anchor

callback: [ self inform: 'yes!' ];

with: 'self inform:'. ...

The message confirm: asks a question and waits for the user to select either Yes or No. The component answers a boolean, which can be used to perform further actions. ...

html space.

html anchor

callback: [

(self confirm: 'Are you happy?')

ifTrue: [ self inform: ':-)' ]

ifFalse: [ self inform: ':-(' ] ];

with: 'self confirm:'.

Наступні допоміжні методі, такі як chooseFrom:caption:, зазначені в допоміжному протоколі компоненти WAComponent.

**Завдання**

Завдання є компонентою, підкласом WATask. Сама вона нічого не відображає, але викликає інші компоненти в потоці контролю, визначені імплементацією методу go.

WAConvenienceTest є простим прикладом завдання, визначеного в Seaside-Tests-Functional. Щоб побачити його ефект, задайте адресу в браузері http://localhost:8080/tests/functional, виберіть WAFlowConvenienceFunctionalTest та натисніть Restart.

**274**

Seaside на прикладі

**Рисунок 14.14:Просте завдання.**

WAFlowConvenienceFunctionalTest >> go

[ self chooseCheese.

self confirmCheese ] whileFalse.

self informCheese

Це завдання по порядку викликає три компоненти. Перша, згенерована допоміжним методом chooseFrom: caption:, це WAChoiceDialog підкомпонента, яка питає користувача вибрати сир. WAFlowConvenienceFunctionalTest >> chooseCheese

cheese := self

chooseFrom: #('Greyerzer' 'Tilsiter' 'Sbrinz')

caption: 'What''s your favorite Cheese?'.

cheese isNil ifTrue: [ self chooseCheese ]

Друга компонента WAYesOrNoDialog для підтвердження вибору (генерується за допомогою допоміжного методу confirm:).

WAFlowConvenienceFunctionalTest >> confirmCheese

^self confirm: 'Is ', cheese, ' your favorite cheese?'

Третя компонента WAFormDialog викликається через допоміжний метод inform:.

WAFlowConvenienceFunctionalTest >> informCheese

self inform: 'Your favorite cheese is ', cheese, '.'

Згенеровані діаолги відображені на рисунку 14.14.

**Транзакції**

В розділі 14.3 ми бачили, що Seaside зберігає прослідковування відповідності міжстаном компоннети та окремими веб-сторінками, компоненти реєструють свій стан для зворотнього відслідковування:все, що компоненти мають зробити – імплементувати метод states ,що повертає масив всіх об’єктві, стан яких необхідно відслідковувати.

**275**

14.6 Управління потоком контролю

Рисунок 14.15: Ресторан Суші

Хоча деколи, ми не хочемо відслідковувати стан. Це часто називається ”проблема магазинного візка ”. Як тільки ви купили все, що було у вас в кошику, не повинно бути можливості повернутися назад і додати ще щось у візок!

Seaside дозволяє визначити завдання, в якому дії будуть погруповані разом як транзакції. Ви можете зворотньо відслідковувати транзакцію, але як тільки транзакція відбулася, можливість повернутися назад знакає.Відповідні сторінки стають недійсні,кожна спроба повернутися назад змушує Seaside генерувати попередження і перенаправляти коритсувача до найновішої валідної сторінки.

Крамниця суші Seaside є простою аплікацією, яка ілюструє багато можливостей Seaside,включаючи транзакції. Ця аплікація доступна в інстальованій версії Seaside http://localhost:8080/seaside/examples/store.

Крамниця суші підтримує такі процеси:

• відвідування крамниці

• Пошук суші.

• Додавання суші в кошик.

• Розрахування

• Підтвердження замовлення.

**276**

Seaside на прикладі

14.6 Managing control flow

• Задання адреси доставки.

• Підтвердження адреси доставки

• Задання інформації для оплати

• Ваша риба вже в дорозі!

Компонентою найвищого рівня є екземпляр WAStore. Вона відображає компоненту-завдання,екземпляр WAStoreTask.

WAStore >> renderContentOn: html

"... render the title bar ..."

html div id: 'body'; with: task

WAStoreTask захоплює цю послідовність дій.Буває критично, що користувач не може повернутися назаді змінити підтверджену інфоормацію.

Замовте суші і спробуйте повернутися назад, щоб поповнити свій кошик.

Ви отримаєте повідомлення, що сторінка застаріла.

Seaside дозволяє розробнику зазначити , що частина процесів є транзакціями: як тільки вона виконалася, користувач не може повернутися назад або відмінити попередні діх. Це зроблено за допомогою надсилання isolate: до завдання з транзакційним блоком, як його аргумент. На прикладі нашої крамниці:

WAStoreTask >> go

| shipping billing creditCard |

cart := WAStoreCart new.

self isolate:

[[ self fillCart.

self confirmContentsOfCart ]

whileFalse ].

self isolate:

[ shipping := self getShippingAddress.

billing := (self useAsBillingAddress: shipping)

ifFalse: [ self getBillingAddress ]

ifTrue: [ shipping ].

creditCard := self getPaymentInfo.

self shipTo: shipping billTo: billing payWith: creditCard ].

self displayConfirmation.

В даному прикладі бачимо дві транзакції. Перша заповнює кошик та закриває фазу шопінгу. (допоміжні методи, такі як fillCart, дбають про інстанціацію та виклик правильних підкомпонент.)Як тільки ви підтвердил вміст кошика,ви не можете повернутися назад і почати нову сесію.Друга транзакціє завершує надання інформацію по доставці та оплаті. Ви можете пересуватися по сторінках вперед-назад, поки не підтвердите оплату. Хоча, коли обидві транзакції заврешені, будь-яка спроба повернутися назад буде марною.

**277**

Seaside на прикладі

Транзакції також можуть бути вкладеними.Приклад вкладеності можна знайти в класі WANestedTransaction.Перший isolate: приймає як аргумент блок, який містить інший, вкалдений isolate:

WANestedTransaction >> go

self inform: 'Before parent txn'.

self isolate:

[self inform: 'Inside parent txn'.

self isolate: [self inform: 'Inside child txn'].

self inform: 'Outside child txn'].

self inform: 'Outside parent txn'

Перейдіть на http://localhost:8080/tests/functionals, виберіь WATransactionTest та натисніть Restart. Спробуйте пересуватися в батьківській та дочірній транзакції, натискаючи кнопку Back, потім Ok.Замітьте, як тільки транзакція відбулася,ви більше не можете повернутися назад всередину без генерування помилки.

**14.7 Повний приклад для ознайомлення**

Розглянемо, як побудувати аплікацію в Seaside з початку. (Вправа повинна зайняти декілька годин. Якщо ви тільки хочете подивитися но готовий код, його можна витягнути з проекту SqueakSource http://www.squeaksource.com/PharoByExample. Пакет для завантаження PBE-SeasideRPN. Ми побудуємо ЗВЗ (Зворотній Польський Запис) калькулятор, як Seaside аплікацію, яка використовує просту стек-машину,як головну модель. Більше того, інтерфейс Seaside дозволить користуватися двома дисплеями — перший, який показує нам значення на вершині стеку,інший показує повний стан стеку.Кфлькулятор зображений на рисунку 14.16.

Почнемо з імплементації стек-машини та її тестів.

Спочатку, визначимо новий клас MyStackMachine з екземпляром змінної, ініціалізованою new OrderedCollection.

MyStackMachine >> initialize

super initialize.

contents := OrderedCollection new.

Стек машинаповинна забезпечувати операції додавання та видалення значення, перегляд вершини стеку, і виконувати різні арифметичні операції з верхніми значеннями стеку.

Напишемо декілька тестів для операцій стеку. Нижче зображений простий тест:

**278**

Figure 14.16: RPN calculator and its stack machine.

MyStackMachineTest >> testDiv

stack

push: 3;

push: 4;

div.

self assert: stack size = 1.

self assert: stack top = (4/3).

Можна використати допоміжні методи для арифметичних операцій, щоб перевірити, чи є два значення в стеці перед роботою, генеруватиметься помилка якщо передумова не виконається. (Хороша ідея використовувати Object>>assert:, щоб визначити передумови для операції. Цей метод виведе AssertionFailure , якщо користувач пробуватиме застосувати стек-машину в невалідному стані)

Можна також визначити метод MyStackMachine>>printOn: для легшої перевірки стек-машини за допомогою інспектора об’єктів.

Завершіть MyStackMachine операціями dup (додати копію верхнього значення в стек), exch (поміняти місцями два верхніх значення стеку) і rotUp (повністю змінити порядок значень в стеці).

Тепер ми маємо просту імплементацію стек-машини.Ми можемо почати імплементацію Seaside ЗПЗ Калькулятора.

Ми використаємо 5 класів:

1. MyRPNWidget — абстрактний клас, що визначає CSSтаблицю стилів для аплікації, та нішу загальну поведінку для компонент ЗПЗ калькулятора. Це підклас WAComponent і прямий надклас наступних чотирьох класів.

**279**

14.7 A complete tutorial example

Seaside by example

2. MyCalculator —це коренева компонента.Вона повинна реєструвати аплікацію, інстанціювати та відображати підкомпоненти, і вона повинна реєструвати будь-який стан для зворотнього відслідковування.

3. MyKeypad – ця компонента відображає кнопки, за допомогою яких ми зможемо взаємодіяти з калькулятором.

4. MyDisplay — ця компонента відображає вергину стеку та має кнопку, що викликає іншу компоненту для відображення детального вигляду.

5. MyDisplayStack — компонента відображає детальний вигляд стеку та має кнопку для зворотньої відповіді. Це підклас MyDisplay.

Створюємо MyCalculator, та визначаємо MyRPNWidget.

Визначаємо загальний стиль аплікації.

Знизу записаний мінімальний CSS код для аплікації. Ви можете його змінити відповідно до своїх ідей.

MyRPNWidget >> style

^ 'table.keypad { float: left; }

td.key {

border: 1px solid grey;

background: lightgrey;

padding: 4px;

text-align: center;

}

table.stack { float: left; }

td.stackcell {

border: 2px solid white;

border-left-color: grey;

border-right-color: grey;

border-bottom-color: grey;

padding: 4px;

text-align: right;

}

td.small { font-size: 8pt; }'

Визначаємо MyCalculator кореневою компонентою і реєструємо її як аплікацію

Імплементуємо MyCalculator>>renderContentOn: ждя відображення чогось тривіального, і перевіряємо, чи аплікація відкривається в браузері.

MyCalculator відповідальний за інстанціацію MyStackMachine, MyKeypad та MyDisplay.

Визначаємо MyKeypad і MyDisplay , як похідні класи MyRPNWidget.

Всі три компоненти потребують доступ до екземпляру стек-машини, тому визначаэмо екземпляр змінної stackMachine та метод ініціалізації setMyStackMachine:в спільному батьку, MyRPNWidget. Додаємо екземпляр змінних keypad та display до MyCalculator та ініціалізуємо їх в MyCalculator>>initialize.

**280**

Figure 14.17: Displaying the top of the stack.

Імплементуємо MyCalculator>>renderContentOn: для спрощення відображення клавіатури та дисплею. Щоб правильно зобразити підкомпоненти, потрібно імплементувати MyCalculator>>children ,щоб повертати масив з дисплеєм та клавіатурою. Імплементуємо методи відображення для клавіатури та дисплею та перевіряємо, чи калькулятор відображає дві підкомпоненти

Тепер змінимо імплементацію дисплею, щоб показувати верхнє значення стеку.

Змінимо метод відображення в клавіатурі для перевірки, чи 0 був доданий в стек у випадку, коли стек пустий. (визначте та використайте метод MyKeypad>>ensureStackMachineNotEmpty.) Також зробіть так, щоб відображалася порожня таблиця з класом клавіатури. Тепер калькулятор повинен відображати кожну комірку зі значенням . В результаті отримаємо вигляд, схожий до рисунку 14.17

Тепер настав час імплементації взаємодії зі стеком.

Спочатку, визначимо допоміжні методи, які зроблять інтерфейс легшим:

MyKeypad >> renderStackButton: text callback: aBlock colSpan: anInteger

on: html

html tableData

class: 'key';

colSpan: anInteger;

with:

[ html anchor

callback: aBlock;

with: [ html html: text ]]

MyKeypad >> renderStackButton: text callback: aBlock on: html

self

renderStackButton: text

**281**

14.7 A complete tutorial example

Seaside by example

callback: aBlock

colSpan: 1 on:

html

Ми будемо використовувати ці два методи для визначення кнопок на клавіатуріз відповідними зворотніми викликами.

Використаємо два допоміжні методи, щоб запрограмувати клавіатуру.

MyKeypad >> renderContentOn: html

self ensureStackMachineNotEmpty.

html table

class: 'keypad';

with: [

html tableRow: [

self renderStackButton: '+' callback: [self stackOp: #add] on:html.

self renderStackButton: '&ndash;' callback: [self stackOp:#min] on: html.

self renderStackButton: '&times;' callback: [self stackOp:#mul] on: html.

self renderStackButton: '&divide;' callback: [self stackOp:#div] on: html.

self renderStackButton: '&plusmn;' callback: [self stackOp:#neg] on: html ].

html tableRow: [

self renderStackButton: '1' callback: [self type: '1'] on:html.

self renderStackButton: '2' callback: [self type: '2'] on:html.

self renderStackButton: '3' callback: [self type: '3'] on:html.

self renderStackButton: 'Drop' callback: [selfstackPopIfNotEmpty] colSpan: 2 on: html ]. " і так далі... "

html tableRow: [

self renderStackButton: '0' callback: [self type: '0']colSpan: 2 on: html.

self renderStackButton: 'C' callback: [self stackClearTop] on:html.

self renderStackButton: 'Enter'callback: [self stackOp: #dup. self setClearMode]

colSpan: 2 on: html ]]

Перевірте, чи клавіатура відображається правильно. Якщо пробуєте натиснути на кнопки, то легко бачити, що функціонал ще не працює.

**282**

14.7 A complete tutorial example

Імплементуємо MyKeypad>>type: для обновлення вершини стеку для додавання заданого числа. Потрібно конвертувати значення в рядок, обновити його,і конвертувати його назад в ціле число , як в прикладі:

MyKeypad >> type: aString

stackMachine push: (self stackPopTopOrZero asString, aString)

asNumber.

Два методи stackPopTopOrZero та stackPopIfNotEmpty потрібні для перевірки, чи стек порожній.

MyKeypad >> stackPopTopOrZero

^ stackMachine isEmpty

ifTrue: [ 0 ]

ifFalse: [ stackMachine pop ]

MyKeypad >> stackPopIfNotEmpty

stackMachine isEmpty

ifFalse: [ stackMachine pop ]

Тепер коли ви натискаєте на кнопки з цифрами, дисплей повинен бути оновлений. (Переконайтеся, що MyStackMachine>>pop повертає видалене значення,в іншому випадку це не працюватиме!)

Тепер, нам потрібно імплементувати MyKeypad>>stackOp:

MyKeypad >> stackOp: op

[ stackMachine perform: op ] on: AssertionFailure do: [ ].

Проблема, що ми не знаємо, чи всі операції успішно виконаються. Наприклад, додавання не відбудеться , якщо немає як мінімум двох значень в стеці.На даний момент ми можемо ігнорувати такі помилки.

Перша версія калькулятора повинна працювати на даний момент.Спробуйте додати декілька чисел, натисніть Enterдля отримання копій поточного значення.

Ви помітите, що введення цифр не проходить так, як ви очікували. Насправді, калькулятор повинен знати чи ви вводите нове число, чи воно вже існує

Додамо MyKeypad>>type: для різної поведінки в задежності від режиму введення.

Представимо екземпляр змінної mode, який приймає одне з трьох значень: символ #typing (коли ви набираєте), #push (коли ви виконпали арифметичні операції і введеня змушує вершину стеку посунутися), або #clear (після того, як ви натиснули Enter і верхнє значення потрібно видалити перед введенням). Новий type: виглядатиме так:

MyKeypad >> type: aString

self inPushMode ifTrue: [

**283**

Seaside by example

stackMachine push: stackMachine top.

self stackClearTop ].

self inClearMode ifTrue: [ self stackClearTop ].

stackMachine push: (self stackPopTopOrZero asString, aString)

asNumber.

Введення має тепер працювати краще, але досі немає подивитися на стек детільніше.

Визначимо MyDisplayStack як похідний клас MyDisplay.

Додамо кнопку для відображення методу MyDisplay ,який викличе новий екземпляр MyDisplayStack. Вам буде потрібно HTML-анкор,який виглядає так:

html anchor

callback: [ self call: (MyDisplayStack new setMyStackMachine:stackMachine)];

with: 'open'

Зворотній виклик спричинить тимчасову заміну екземпляру MyDisplay на новий екземпляр MyDisplayStack, чия робота – відобразити стек повністю. Коли компонента повідомляє, що вона готова ( надіславши self відповідь), тоді контроль поверне екземпляр MyDisplay.

Визначимо метод відображення MyDisplayStack, щоб зобразити всі значення в стеці.

(Вам потрібно буде визнаити доступ до контенту стек-машини або ви можете визначити MyStackMachine>>do: для проходження па значеннях стеку). Дисплей стеку повинен також мати кнопку, чий зворотній виклик просто виконає self відповідь.

html anchor

callback: [ self answer];

with: 'close'

Тепер ви повинні мати змогу відкрити та закрити стек, поки ви використовуєте калькулятор.

Ще існує одна річ, про яку ми забули.Спробуйте виконати декілька операцій в стеці. Тепер використайте кнопку Back і спробуйте виконати ще декілька операцій. Наприклад, відкрийте, введіть 1, введіть двічі і натисніть +. Стек повинен відобразити 2 та 1. Тепер натисніть кнопку Back. Тепер стек показує знову три одиниці. Тепер коли ви натиснете +, стек покаже 3. Зворотнє відстеження ще не працює.

Імплементуємо MyCalculator>>states для повернення масиву з вмістом стек-машини.

Перевірте, чи тепер зворотнє відстеження працює правильно.

**284**

14.8 A quick look at AJAX

**14.8 Швидкий огляд AJAX**

AJAX (Asynchronous JavaScript and XML) це техніка для розробки веб-аплікацій більш інтерактивними за допомогою використання функціональності JavaScript на частні клієнта.

Дві загальновідомі JavaScript бібліотеки є Prototype (http://www.prototypejs.org) та script.aculo.us (http://script.aculo.us). Prototype надає фреймворк для легкого написання JavaScript. script.aculo.us надає додаткові можливості для підтримки анімацій.Обидва фреймворки підтримуються Seaside за допомогою пакету Scriptaculous.

Розширення Scriptaculous має той самий підхід, як Seaside —просто налаштовуємо об’єкти Pharo до моделі вашої аплікації, необхідний Javascript код згенерується для вас.

Розглянемо простий приклад, як підтримка Javascript на клієнтській частині може зробити поведінку нашого ЗПЗ калькулятора натуральнішою . На даний момент кожен раз, коли ми вводим цифру, генерується запит на обновлення сторінки.Ми б хотіли змінювати дисплей на клієнській частині , обновляючи дисплей на частині клієнта.

Змінимо відображення калькулятора наступним чином. (якщо ви імплементували ознайомчий проект, ви можете легко завантажити повний проект (PBE-SeasideRPN) з http://www.squeaksource.com/PharoByExample.

MyCalculator >> renderContentOn: html

html div id: 'keypad'; with: keypad.

html div id: 'display'; with: display.

Щоб змогти повторно відобразити дисплей, коли натиснути кнопку клавіатури, клавіатура повинна знати компоненту дисплею.

Додамо екземпляр змінної дисплею до MyKeypad, метод ініціалізації MyKey- pad>>setDisplay:, і викличемо це з MyCalculator>>initialize. Тепер ми можемо призначити JavaScript код до кнопок, обновивши MyKey- pad>>renderStackButton:callback:colSpan:on: as follows:

MyKeypad >> renderStackButton: text callback: aBlock colSpan: anInteger

on: html

html tableData

class: 'key';

colSpan: anInteger;

**285**

Seaside by example

with: [

html anchor

callback: aBlock;

onClick: "handle Javascript event"

(html scriptaculous updater

id: 'display';

callback: [ :r |

aBlock value.

r render: display ];

return: false);

with: [ html html: text ] ]

onClick: специфікує JavaScript обробник подій. html updater повертає екземпляр PTUpdater, а об’єкт Smalltalk представляє об’єкт JavaScript AJAX Updater (http://www.prototypejs.org/api/ajax/updater). Цей об’єкт виконує AJAX-запити і обновляє вміст контейнера, базуючись на тексті відповіді. id: вказує користувачу, який HTML DOM елемент потрібно обновити, в цьому випадку вміст div елемента з id’display’. callback: специфікує блок, який викликається, коли користувач натисне на кнопку. Блоковий аргумент є новим відображувачем r, який ми можемо використовувати для відображення дисплей компоненти. Перед відображенням дисплей компоненти очікуємо, що aBlock виконаєзадуману дію.

Спробуйте калькулятор ще раз, і помітите, що сторінка обновлюється кожного разу при натисканні кнопки з цифрою.

Хоча ми реалізували поведінку клієнт-сервера,ми ще її не активували.Тепер нам потрібно розблокувати Javascript обробник подій.

Натсніть на посилання Configure в інструментах калькулятора.

Налаштуйте атрибути бібліотеки під розділом General. (Можливо буде потрібно розблокувати модифікацію цього атрибуту, спочатку вибравши Modify). ЗІ списку доступних бібліотек, виберіть PTDevelopmentLibrary та застосуйте зміни.

Замість ручного налаштування бібліотеки, можна це зробити при реєстрації аплікації:

MyCalculator class >> initialize

(WAAdmin register: self asApplicationAt: self applicationName)

addLibrary: PTDevelopmentLibrary

**286**

Figure 14.18: Seaside AJAX processing (simplified).

В цьому випадку PTDevelopmentLibrary достатньо, але для повного набору розширень scriptaculous вам потрібно також додати SUDevelopmentLibrary.

Зпробуйте запустити переглянуту аплікацію. Зауважте, що враження набагато натуральніше. Зокрема,нова URL не генерується з кожного натискання клавіші.

Ви можете спитати,як це працює? Рисунок 14.18 зображає, як ЗПЗ аплікації працюватимуть з і без AJAXу. В основному, AJAX викликає відображення, щоб тільки обновити компоненту дисплею. Javascript відповідає за обробку запиту та обновлення відповідного DOM-елементу. Перегляньте згенероманий код, особливо JavaScript-частину:

new Ajax.Updater(

'display',

'http://localhost/seaside/RPN+Calculator',

{'evalScripts': true,

'parameters': ['UNDERSCOREs=zcdqfonqwbeYzkza',

'UNDERSCOREk=jMORHtqr','9'].join('&')});

return false

**287**

14.8 A quick look at AJAX

Seaside by example

Більше прикладів ви можете знайти за посиланням http://localhost:8080/ javascript/scriptaculous.

Підказки:

У випадку наявності проблем на серверній частині викоистовуйте Debugger. У випадку проблем на клієнтській частині використовуйте FireFox (http://www.mozilla.com) з JavaScript дебаґером FireBug (http://www.getfirebug.com/).

**14.9 Chapter summary**

• Найлегший спосіб завантажити Seaside - One-Click Experience на http://seaside.st

• Виключіть та включіть сервер, викорстовуючи ZnZincServerAdaptor startOn: 8080 та ZnZincServerAdaptor stop.

• Переключіть Halos, щоб напряму переглянути код аплікації, об’єкти, які виконуються, CSS та HTML.

• Видаліть кореневий клас WAToolDecoration в конфігурації аплікації, щоб заблокувати панель інструментів.

• Надішліть WAAdmin applicationDefaults removeParent: WADevelopment- Configuration instance щоб заблокувати панель інструментів для нових компонент.

• Seaside веб-аплікації складаються з компонент, які є екземпляри похідного класу від WAComponent.

• Тільки коренева компонента може бути зареєстрована, як аплікація. В іншому випадку вона може реєструвати себе, як аплікацію в методі initialize, надсилаючи WAAdmin register: self asApplicationAt: шлях до аплікація.

• Для зворотнього відслідковування стану , компонента повинна містити метод, що повертає масив об’єктів, чий стан буде збережений, коли користувач натисне кнопку Back.

• компонента відображає себе, імплементуючи renderContentOn:. Аргументом цього методу є HTML відображаюче полотно.

• Компонента може відобразити підкомпоненту, надіславши self render: підкомпонента.

**288**

14.9 Chapter summary

• HTML генерується, надсилаючи повідомлення до пензлів.

• Якщо ви надішлете каскад повідомлень до пензля, які мають повідомлення with:, тоді with: повинне бути останнім надісланим повідомленням. With: визначає вміст і відображає результат.

• Дії повинні появлятися тільки в зворотніх викликах. Не можна змінювати стан аплікації, поки ви її відображаєте.

• Ви можете зєднати багато форм та анкорів з екземплярами змінних, надсилаючи *on: екземпляр змінної of: об’єкт до пензля*.

• Ви можете визначити CSS для ієрархії компонент ,зазначивши стиль методу, який повинен повернути рядок, що мітить таблицю стилів.

• Контрольний потік може бути визначеним, як завдання — екземпляром похідного класу від WATask.

• Використовуйте допоміжні методи з WAComponents request:, inform:, confirm: і chooseFrom:caption: для базової взаємодії.

• Ви можете зворотньо відслідковувати транзакцію, але як тільки транзакція відбулася, можливість повернутися назад зникає.