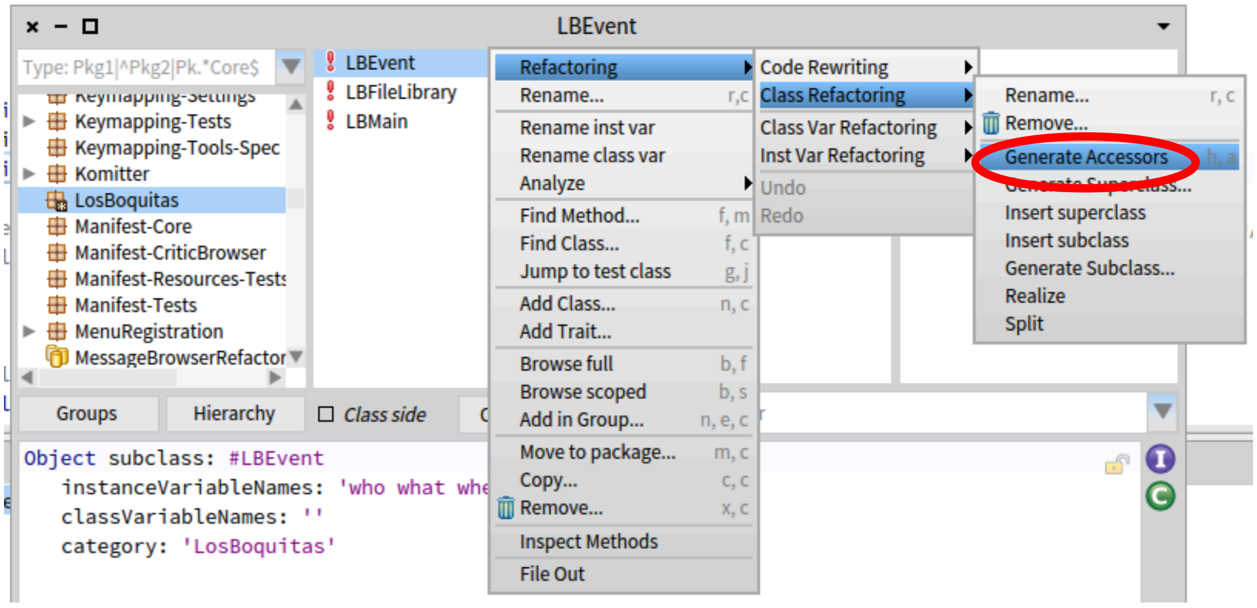
In this chapter we enhance the Los Boquitas application with a new component showing upcoming events in a table.

1. First, we need to have some events to display.

   a. We will start by defining an event class.

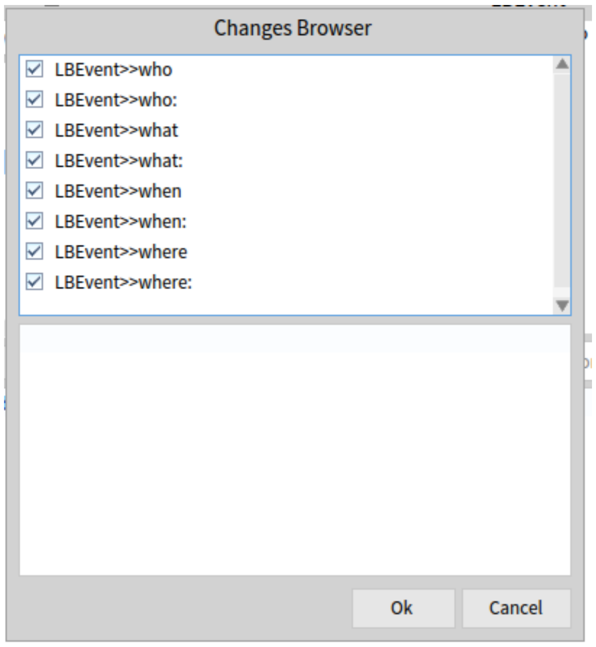
```
Object subclass: #LBEvent
   instanceVariableNames: 'who what when where'
   classVariableNames: ''
   category: 'LosBoquitas'
```

   b. Next we will create accessors for the instance variables. Rather than creating the methods one at a time, you can use some of Pharo's refactoring tools to create the methods. Select LBEvent, right-click and select 'refactor class' then 'accessors'.



```
Object subclass: #LBEvent
    instanceVariableNames: 'who what whe
    classVariableNames: ''
    category: 'LosBoquitas'
```

c. The refactoring tool will show you the proposed new methods and give you a chance to accept or cancel before the changes are installed. Click the ‘Ok’ button.



LBEvent>>who
LBEvent>>who:
LBEvent>>what
LBEvent>>what:
LBEvent>>when
LBEvent>>when:
LBEvent>>where
LBEvent>>where:

d. Add a method to support sorting the events.

```
<= anEvent

   ^self when <= anEvent when.
```

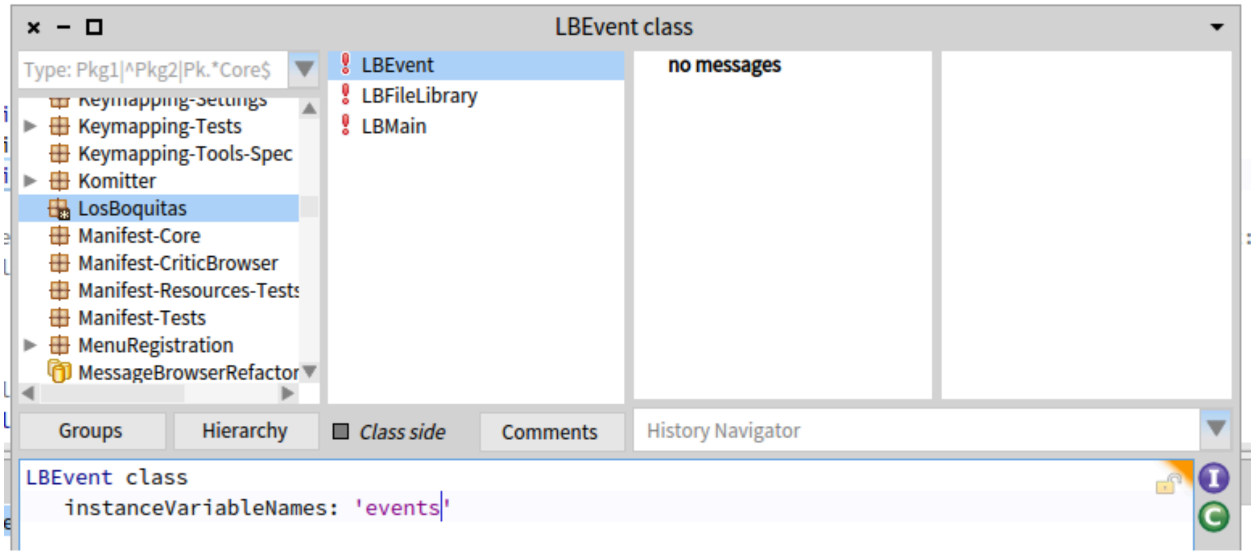
e. Add an initialize method to ensure that something is in each instance variable.

```
initialize

   super initialize.
   who := 'players'.
   what := 'practice'.
   when := DateAndTime noon.
   where := 'field'.
```

2. Now we need a place to put the events. In most web frameworks and languages we would now start a discussion of setting up a relational database. In Smalltalk, however, we prefer to avoid the 'object-relational impedance mismatch' problem (see http://en.wikipedia.org/wiki/Object-Relational_impedance_mismatch) as long as possible. Instead of an external database that must be configured and mapped to, we will save our event objects in a *class instance variable* on the LBEvent class.

   a. In the Pharo System Browser, select LBEvent in the class list and then click on the '*Class side*' checkbox below the class list. This will change the class definition to a definition for the class instance variable. Edit the text area to add an 'events' class instance variable and save the text.



```
LBEvent class
   instanceVariableNames: 'events'
```

   b. Now click in the method category list to get a method template. Add a *class-side* method to access the events.

```
events

   events isNil ifTrue: [events := IdentitySet new].
   ^events.
```

c. Add a *class-side* method to create some sample events.

```
createEvents
"
   LBEvent createEvents.
"
   events := nil.
   self events
      add: (self new
         who: 'family';
         what: 'registration';
         when: DateAndTime noon;
         where: 'Clubhouse';
         yourself);
      add: (self new
         who: 'players';
         what: 'practice';
         when: (DateAndTime noon + (Duration days: 1));
         where: 'field';
         yourself);
      add: (self new
         who: 'guests';
         what: 'game';
         when: (DateAndTime noon + (Duration days: 2));
         where: 'Memorial Park';
         yourself);
      yourself.
```

d. In the SystemBrowser, click anywhere on the third line of the method (the one sending the ‘createEvents’ message) and press <Ctrl>+<D> (for ‘do-it’). By adding the expression to the method as a comment, we can evaluate it without having to go to a workspace.

3. Now we will define a component to display the schedule.

```
WAComponent subclass: #LBScheduleComponent
   instanceVariableNames: 'listComponent'
   classVariableNames: ''
   category: 'LosBoquitas'
```

a. The goal is to embed this component into the main application, but for purposes of development and testing we will treat this as a stand-alone component (or application). Register the application by evaluating the following in a workspace.

```
WAAdmin register: LBScheduleComponent asApplicationAt: 'boquitas-schedule'.
```

b. Add a place-holder render method.

```
renderContentOn: html

   html heading: self class name.
```

c. In a web browser, navigate to the dispatcher (http://localhost:8080/browse) and confirm that the new component is in the list and that it displays the class name.

4. Now we will add a real display capability to the component.

   a. Add four methods to define report columns and an initialize method to create a table report using those columns.

```
whoReportColumn

   ^WAReportColumn new
      title: 'Who';
      selector: #who;
      clickBlock: nil;
      yourself.
```

```
whatReportColumn

   ^WAReportColumn new
      title: 'What';
      selector: #what;
      clickBlock: nil;
      yourself.
```

```
whenReportColumn

   ^WAReportColumn new
      title: 'When';
      selector: #when;
      clickBlock: nil;
      yourself.
```

```
whereReportColumn

   ^WAReportColumn new
      title: 'Where';
      selector: #where;
      clickBlock: nil;
      yourself.
```

```
initialize

   | columns |
   super initialize.
   columns := Array
      with: self whoReportColumn
      with: self whatReportColumn
      with: self whenReportColumn
      with: self whereReportColumn.
   listComponent := WATableReport new
      columns: columns;
      rowPeriod: 1;
      yourself.
```

b. Now modify the render method to show the table.

```
renderContentOn: html

   listComponent rows: LBEvent events asSortedCollection.
   html render: listComponent.
```

c. Starting from the dispatcher (http://localhost:8080/browse) in a web browser, view the schedule component and confirm that it shows three rows of four columns. (Don't click on the anchors yet!)

5. Now we will update our main application to make room for a child component.

   a. Change the class schema for LBMain to add an instance variable to hold the component being displayed in the main region.

```
WAComponent subclass: #LBMain
   instanceVariableNames: 'mainArea'
   classVariableNames: ''
   category: 'LosBoquitas'
```

b. Modify LBMain>>#renderSidebarOn: to change the heading.

```
renderSidebarOn: html

   html div
      id: 'sidebar';
      class: 'section';
      with: [
         html heading
            level2;
            with: 'Menu'.
      ].
```

c. Return to your web browser and display the home page (http://localhost:8080/boquitas). It should have the new text now ('Menu' instead of 'Sidebar').

6. Add a menu to the sidebar.

   a. Modify the sidebar render method as follows:

```
renderSidebarOn: html

   html div
      id: 'sidebar';
      class: 'section';
      with: [
         html heading
            level2;
            with: 'Menu'.
         html anchor
            callback: [mainArea := LBScheduleComponent new];
            with: 'Events'.
      ].
```

   b. View the home page in a browser and confirm that the <Events> link is present. Clicking on it does not have any impact, but it is there!

c. We want the render method to use the mainArea component if it exists; otherwise, the image will be displayed. Modify the renderMainOn: method as follows.

```
renderMainOn: html

   html div
      id: 'main';
      class: 'section';
      with: [
         mainArea notNil ifTrue: [
            html render: mainArea.
         ] ifFalse: [
            html image
               altText: 'children playing soccer';
               url: LBFileLibrary / 'boquitas.jpg';
               yourself.
         ].
      ].
```

d. View the home page in a web browser and confirm that the event list displays when you click on the ‘Events’ link in the sidebar.

e. We now want a way to return to the home page. Modify the sidebar render method as follows:

```
renderSidebarOn: html

   html div
      id: 'sidebar';
      class: 'section';
      with: [
         html heading
            level2;
            with: 'Menu'.
         html anchor
            callback: [mainArea := nil];
            with: 'Home'.
         html break.
         html anchor
            callback: [mainArea := LBScheduleComponent new];
            with: 'Events'.
      ].
```

f. View the application in a web browser and confirm that you can switch between the image and the schedule.

7. The ‘renderMainOn:’ method in LBMain includes a conditional that hints for the need of a refactoring. Now that we have one subcomponent, we might as well have more.

    a. Create a new component:

```
WAComponent subclass: #LBHome
   instanceVariableNames: ''
   classVariableNames: ''
   category: 'LosBoquitas'
```

    b. Add a render method to LBHome with code from LBMain>>#’renderMainOn:’.

```
renderContentOn: html

   html image
      altText: 'children playing soccer';
      url: LBFileLibrary / 'boquitas.jpg';
      yourself.
```

    c. LBHome is done. Now we will go back and add an initialize method to LBMain to use our new component.

```
initialize

   super initialize.
   mainArea := LBHome new.
```

    d. Now we can simplify LBMain>>#’renderMainOn:’ considerably by always rendering a subcomponent instead of having conditional code:

```
renderMainOn: html

   html div
      id: 'main';
      class: 'section';
      with: [html render: mainArea].
```

e. Finally, we need to modify one line of the sidebar menu creation to use our new component.

```
renderSidebarOn: html

   html div
      id: 'sidebar';
      class: 'section';
      with: [
         html heading
            level2;
            with: 'Menu'.
         html anchor
            callback: [mainArea := LBHome new];
            with: 'Home'.
         html break.
         html anchor
            callback: [mainArea := LBScheduleComponent new];
            with: 'Events'.
      ].
```

f. Return to a web browser and start the application over from http://localhost:8080/boquitas. You should be able to switch back and forth between the home page and the schedule.

8. We would like to be able to edit events. We will start with deleting an event.

   a. Add LBScheduleComponent>>#'actionReportColumn'.

```
actionReportColumn

   ^WAReportColumn new
      title: 'Action';
      valueBlock: [:anEvent | 'delete'];
      clickBlock: [:anEvent | self delete: anEvent];
      yourself.
```

b. Edit LBScheduleComponent>>#'initialize' to use the new column. Note that instead of using the instance creation message 'with:with:with:with:' on Array, we are using an OrderedCollection and adding items to it. This is because once you reach more than four items, there might not be a class-side method that accepts enough arguments. Also note that this is an example of where the 'yourself' message at the end of the cascade is not just cosmetic but is necessary since the 'add:' method returns the argument (a column) rather than the receiver (an OrderedCollection).

```
initialize

   | columns |
   super initialize.
   columns := OrderedCollection new
      add: self whoReportColumn;
      add: self whatReportColumn;
      add: self whenReportColumn;
      add: self whereReportColumn;
      add: self actionReportColumn;
      yourself.
   listComponent := WATableReport new
      columns: columns;
      rowPeriod: 1;
      yourself.
```

c. If you return to your web browser and refresh, the new column will likely not appear. This is because the component is still holding an instance of WATableReport that was initialize with only four columns. To see the new table you need to click on the 'Events' link to install a new component.

d. If you click on a <delete> link now, you should get a MessageNotUnderstood error because we have not implemented the #delete: method in LBScheduleComponent. Add the following method (and note how we are not doing any SQL or other database related activity):

```
delete: anEvent

   LBEvent events remove: anEvent.
```

e. Try refreshing your web browser and note that one of the events has been removed.

9. Before you delete all the events, we will add a confirm dialog using JavaScript. Edit LBScheduleComponent>>#'actionReportColumn' to change how the 'Delete' anchor is generated. Note that having the column definition in its own method means that we don't have to edit a big method to make this change. The definition of the column is encapsulated in a single method and does not share the method with another definition.

```
actionReportColumn

   ^WAReportColumn new
      title: 'Action';
      valueBlock: [:each :html |
         html anchor
            onClick: 'return confirm(''Are you sure?'')';
            callback: [self delete: each];
            with: 'delete'.
      ];
      yourself.
```

   Here we are creating an anchor and giving it JavaScript for the 'onclick' event. The JavaScript code will run before the link is followed, and if the JavaScript returns false the new page is not requested.

   In order for this change to be visible, you must recreate the component. A simple way to do this is to click the 'Home' link and then click the 'Events' link.

   After deleting events, recreate them by evaluating 'LBEvent createEvents' in a workspace or using the instructions at step 2d above.

10. Save your Pharo image and quit.