Advanced Java Application Development for the BlackBerry Smartphone Lab manual
REALIZE ANY EXPECTED SAVINGS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, LOSS OF
BUSINESS OPPORTUNITY, OR CORRUPTION OR LOSS OF DATA, FAILURES TO TRANSMIT OR RECEIVE ANY
DATA, PROBLEMS ASSOCIATED WITH ANY APPLICATIONS USED IN CONJUNCTION WITH RIM PRODUCTS
OR SERVICES, DOWNTIME COSTS, LOSS OF THE USE OF RIM PRODUCTS OR SERVICES OR ANY PORTION
THEREOF OR OF ANY AIRTIME SERVICES, COST OF SUBSTITUTE GOODS, COSTS OF COVER, FACILITIES OR
SERVICES, COST OF CAPITAL, OR OTHER SIMILAR PECUNIARY LOSSES, WHETHER OR NOT SUCH DAMAGES
WERE FORESEEN OR UNFORESEEN, AND EVEN IF RIM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH
DAMAGES.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW IN YOUR JURISDICTION, RIM SHALL HAVE
NO OTHER OBLIGATION, DUTY, OR LIABILITY WHATSOEVER IN CONTRACT, TORT, OR OTHERWISE TO
YOU INCLUDING ANY LIABILITY FOR NEGLIGENCE OR STRICT LIABILITY.

THE LIMITATIONS, EXCLUSIONS, AND DISCLAIMERS HEREIN SHALL APPLY: (A) IRRESPECTIVE OF THE
NATURE OF THE CAUSE OF ACTION, DEMAND, OR ACTION BY YOU INCLUDING BUT NOT LIMITED TO
BREACH OF CONTRACT, NEGLIGENCE, TORT, STRICT LIABILITY OR ANY OTHER LEGAL THEORY AND
SHALL SURVIVE A FUNDAMENTAL BREACH OR BREACHES OR THE FAILURE OF THE ESSENTIAL PURPOSE
OF THIS AGREEMENT OR OF ANY REMEDY CONTAINED HEREIN; AND (B) TO RIM AND ITS AFFILIATED
COMPANIES, THEIR SUCCESSORS, ASSIGNS, AGENTS, SUPPLIERS (INCLUDING AIRTIME SERVICE
PROVIDERS), AUTHORIZED RIM DISTRIBUTORS (ALSO INCLUDING AIRTIME SERVICE PROVIDERS) AND
THEIR RESPECTIVE DIRECTORS, EMPLOYEES AND INDEPENDENT CONTRACTORS.

IN ADDITION TO THE LIMITATIONS AND EXCLUSIONS SET OUT ABOVE, IN NO EVENT SHALL ANY
DIRECTOR, EMPLOYEE, AGENT, DISTRIBUTOR, SUPPLIER, INDEPENDENT CONTRACTOR OF RIM OR ANY
AFFILIATES OF RIM HAVE ANY LIABILITY ARISING FROM OR RELATED TO THE DOCUMENTATION.

Prior to subscribing for, installing or using any Third Party Products and Services it is your responsibility to
ensure that your airtime service provider has agreed to support all of their features. Some airtime service
providers may not offer Internet browsing functionality with a subscription to BlackBerry® Internet Service.
Check with your service provider for availability, roaming arrangements, service plans and features. Installation
or use of Third Party Products and Services with RIM's products and services may require one or more patent,
trademark, copyright or other licenses in order to avoid infringement or violation of third party rights. You are
solely responsible for determining whether to use, Third Party Products and Services and if any third party
licenses are required to do so. If required you are responsible for acquiring them. You should not install or use
Third Party Products and Services until all necessary licenses have been acquired. Any Third Party Products and
Services that are provided with RIM's products and services are provided as a convenience to you and are
provided "AS IS" with no express or implied conditions, endorsements, guarantees, representations or warranties
of any kind by RIM and RIM assumes no liability whatsoever, in relation thereto. Your use of Third Party Products
and Services shall be governed by and subject to you agreeing to the terms of separate licenses and other
agreements applicable thereto with third parties, except to the extent expressly covered by a license or other
agreement with RIM.

Certain features outlined in this documentation require a minimum version of BlackBerry® Enterprise Server
software, BlackBerry® Desktop Software, and/or BlackBerry® Device Software and may require additional
development or Third Party Products and Services for access to corporate applications.

This product includes software developed by the Apache Software Foundation (http://www.apache.org/) and/or
licensed pursuant to Apache License, Version 2.0 (http://www.apache.org/licenses/). For more information, see
the NOTICE.txt file included with the software. Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF
ANY KIND, either express or implied. See the License for the specific language governing permissions and
limitations under the License.
The terms of use of any RIM product or service are set out in a separate license or other agreement with RIM applicable thereto. NOTHING IN THIS DOCUMENTATION IS INTENDED TO SUPERSEDE ANY EXPRESS WRITTEN AGREEMENTS OR WARRANTIES PROVIDED BY RIM FOR PORTIONS OF ANY RIM PRODUCT OR SERVICE OTHER THAN THIS DOCUMENTATION.
Chapter 4: Advanced BlackBerry user interface development

This chapter outlines the structure of the BlackBerry® smartphone UI, and describes how they work together. This chapter describes how you can customize Managers and Fields to create a dynamic and easy to use interface for the smartphone user, while taking into consideration their unique requirements. This chapter also describes how to create and use SVG to enhance the BlackBerry smartphone UI experience.
4.8 Create a choice field

In this exercise, you will create a choice field that allows a user to select cities. Complete exercises 4.1 through 4.6. You will need the Lab_02_UserInterface.zip.

1. Construct a new ObjectChoiceField and supply a String label and an Object array as parameters.
   *Hint: One of the required parameters has been defined for you in the CitySelectionScreen class. The other parameter is defined in the CityDetailsScreen class.*

2. In the CitySelectionScreen constructor, complete the TODO step titled Create List.
   ```java
   citiesChoiceField = new ObjectChoiceField(SELECT_CITY_STRING, CityDetailsScreen.cities);
   ```
4.9 Create a screen

In this exercise, you will create a Screen and add UI components to it.

1. Initialize the three BasicEditFields with the following criteria:
   - The labels must be self-descriptive.
   - The initial value must be null.
   - The maximum number of characters must be 40.
   - The style must be Field.READONLY.

2. In the CityDetailsScreen() constructor, complete the TODO step titled Create BasicEditFields.
   _popField = new BasicEditField("Population: ", null, 40, Field.READONLY);
   _stateField = new BasicEditField("State: ", null, 40, Field.READONLY);
   _sightsField = new BasicEditField("Sights: ", null, 40, Field.READONLY);

3. Display the city name LabelField.

4. Add a separator field.
   Hint: Use the following ode to add the separator field:
   
   add(new SeparatorField());

5. Add the three information fields.

6. In the CityDetailsScreen() constructor, complete the TODO step titled Add UI Fields.
   add(_cityNameField);
   add(new SeparatorField())
   add(_popField);
   add(_stateField);
   add(_sightsField);

7. Implement the update() method. This method updates text in BasicEditFields based on the selected index in ObjectChoiceField.
8. In the `CityDetailsScreen()` constructor, complete the TODO step titled Update Text.

```java
if (index < cities.length) {
    _cityNameField.setText(cities[index]);
    _popField.setText(populations[index]);
    _stateField.setText(states[index]);
    _sightsField.setText(sights[index]);
}
```
4.10 Update and display the screen

In this exercise, you will update and display the CityDetailsScreen.

1. Call a method on the ObjectChoiceField, which retrieves the selected index to pass to the CityDetailsScreen.update() method.

2. In the pushDetailsScreen() method of the CitySelectionScreen class, complete the TODO step titled Get Index.
   
   int selectedIndex = _citiesChoiceField.getSelectedIndex();

   The static method UiApplication.getUiApplication() returns the running UiApplication.

3. Call the correct method on the UiApplication instance to push the details screen on to the stack.

4. In the pushDetailsScreen() method of the CitySelectionScreen class, complete the TODO step titled Push Screen.
   
   UiApplication.getUiApplication().pushScreen(_detailsScreen);
4.11 Create a MenuItem

In this exercise, you will create a MenuItem subclass that displays a CityDetailsScreen. Create an anonymous inner class that extends MenuItem with the label View Details, an ordinal of 110, and a priority of 10. The run() method should invoke the pushDetailsScreen() method you implemented in exercise 4.4.

1. In the CitySelectionScreen class, complete the TODO step titled Create Menu Item.
   ```java
   private MenuItem _viewItem = new MenuItem("View Details", 110, 10) {
       public void run() { pushDetailsScreen();
   }
   }
   ```

2. Add the menu item to the CitySelectionScreen. In the CitySelectionScreen() constructor, complete the TODO step titled Add Menu Item.
   ```java
   addMenuItem(_viewItem);
   ```
4.12 Add a FieldChangeListener

In this exercise, you will implement a FieldChangeListener to update the details when the user selects a city from the list. You have implemented the FieldChangeListener interface in the CitySelectionScreen class. Next, call a method on the ObjectChoiceField to identify the class as a change listener.

1. In the CitySelectionScreen() constructor, complete the TODO step titled Add Listener.
   _citiesChoiceField.setChangeListener(this);
4.13 Build and run the user interface project

In this exercise you will save your changes from exercises 4.1 through 4.5 and run the application on the BlackBerry Smartphone Simulator.

1. In the Package Explorer view, right click the Lab_02_UserInterface project.

2. Click Activate for BlackBerry. The project folder icon changes to indicate that the project is activated.

3. In the Package Explorer view, right click on the Lab_02_UserInterface project.

4. Click Run BlackBerry Simulator.

5. When the BlackBerry Smartphone Simulator home screen appears, click Menu to view all applications and folders.

6. Use the arrow keys on your keyboard to navigate to the Downloads folder.

7. Enter the folder by pressing the Enter key on your keyboard.

8. Use the arrow keys on your keyboard to navigate to the generic application icon called User Interface Lab.

9. Press the Enter key on your keyboard to invoke the application.
4.14 Create a border and background for a screen

In this exercise, you will add a field with simple borders and backgrounds to the screen. Complete exercises 4.7 through 4.11. You will need the Lab_12_Decor.zip.

1. Create a RichTextField and assign it a custom border and a custom background. Specify the field text as "Solid rounded border, solid background". In the DecorDemoScreen constructor, complete the TODO step titled Create RichTextField.
   ```java
   simpleField = new RichTextField("Solid rounded border, solid background");
   ```

2. Use the appropriate static method in the BorderFactory class to create a rounded border for use with the RichTextField. Use the predefined "thickPadding"; XYEdges object for the first parameter, and then use a constant in the Border class representing the solid style for the second parameter. In the DecorDemoScreen constructor, complete the TODO step titled Create Rounded Border.
   ```java
   roundedBorder = BorderFactory.createRoundedBorder(thickPadding, Border.STYLE_SOLID);
   ```

3. Use the appropriate static method in the BackgroundFactory class to create a Background object. To set a color, pick a static field in the Color class. In the DecorDemoScreen constructor, complete the TODO step titled Create Solid Background.
   ```java
   solidBackground = BackgroundFactory.createSolidBackground(Color.LIGHTSTEELBLUE);
   ```

4. Set the RichTextField border and background to the Border and Background objects we created in steps 4 and 5. In the DecorDemoScreen constructor, complete the TODO step titled Set Border and Background.
   ```java
   simpleField.setBorder(roundedBorder);
   simpleField.setBackground(solidBackground);
   ```

5. Now that the RichTextField is initialized, you can add it to the screen. In the DecorDemoScreen constructor, complete the TODO step titled Add RichTextField.
   ```java
   dd(simpleField);
   ```
In this exercise, you will work with the XYEdges class.

YEdges objects have two purposes in this application. First, they are used to define how much padding is given for a border. The four int parameters of the XYEdges objects represent the amount of space between the box and border of a field for each side, clockwise from the top.

1. Following the format of the already created XYEdges objects, create two XYEdges objects: one to represent horizontal-only padding and one for vertical-only padding. In the DecorDemoScreen constructor, complete the TODO step titled Create Padding XYEdges.
   verticalPadding = new XYEdges(10, 0, 10, 0);
   horizontalPadding = new XYEdges(0, 10, 0, 10);

2. XYEdges objects are also used to define border colours. You can assign a different colour to each edge of a border. Create an XYEdges object where each edge of the border is a different color. In the DecorDemoScreen constructor, complete the TODO step titled Create Colour XYEdges.
   multiColours = new XYEdges(Color.BLUEVIOLET, Color.AZURE, Color.DARKRED, Color.KHAKI);
4.16 Create borders

In this exercise, you will explore some of the options available with the `BorderFactory` class.

You can manipulate both simple and rounded borders in various ways, depending on what parameters you use. Create a vertical, blue, and dashed border using the appropriate static method in the `BorderFactory` class. Use the `XYEdges` objects (defined in exercise 4.8) and static constants in the `Border` class as parameters.

1. **In the DecorDemoScreen constructor, complete the TODO step titled Create Simple Border.**
   ```java
   dashedBorder = BorderFactory.createSimpleBorder(verticalPadding, blueColours, Border.STYLE_DASHED);
   ```

2. **Bevel borders emulate the look of a 3-D border. Bevel borders require two sets of colors - one set for raised lines and the other for the shaded areas between them. Use the appropriate static method in the `BorderFactory` class to create a bevel border with thick padding, multicolour shading, and single-colour lines. In the DecorDemoScreen constructor, complete the TODO step titled Create Bevel Border.**
   ```java
   bevelBorder = BorderFactory.createBevelBorder(thickPadding, multiColours, pinkColours);
   ```
Notes
4.17 Create backgrounds

In this exercise, you will explore some of the options available with the BorderFactory class.

Use gradient backgrounds to create a gradient of four colors, with each color stronger in one corner. Use the appropriate static method in the BackgroundFactory class to create a gradient background with four different colors.

1. In the DecorDemoScreen constructor, complete the TODO step titled Create Gradient Background.
   
   ```java
   gradientBackground = BackgroundFactory.createLinearGradientBackground(Color.RED, Color.GREEN, Color.BLUE, Color.WHITE);
   ```

2. Included in this project is a bitmap file named Smiley. Use the appropriate static method in the BackgroundFactory class to create a bitmap background that uses the bitmap, repeating on both axes for tiling.
   
   *Hint:* Use a static method in net.rim.device.api.system.Bitmap to create a Bitmap object.

3. In the DecorDemoScreen constructor, complete the TODO step titled Create Bitmap Background.
   
   ```java
   bitmapBackground = BackgroundFactory.createBitmapBackground(Bitmap.getBitmapResource("smiley.bmp", Background.POSITION_X_CENTER, Background.POSITION_Y_CENTER, Background.REPEAT_BOTH);
   ```
4.18 Build and run the borders and backgrounds project

In this exercise you will save your changes from exercises 4.7 through 4.10 and run the application on the BlackBerry Smartphone Simulator.

1. In the Package Explorer view, right click the Lab_12_Decor project.

2. Click Activate for BlackBerry. The project folder icon changes to indicate that the project is activated.

3. In the Package Explorer view, right click on the Lab_12_Decor project.

4. Click Run BlackBerry Simulator.

5. When the BlackBerry Smartphone Simulator home screen appears, click Menu to view all applications and folders.

6. Use the arrow keys on your keyboard to navigate to the Downloads folder.

7. Enter the folder by pressing the Enter key on your keyboard.

8. Use the arrow keys on your keyboard to navigate to the generic application icon called Decor Lab.

9. Press the Enter key on your keyboard to invoke the application.
4.19 Create a custom UI component using managers

In this exercise, you will learn how to implement an effective user interface for BlackBerry smartphones. This lab uses the built-in APIs as well as demonstrates how to extend existing interface elements to create a new, custom look and feel for your application.

In this exercise, you will create an application that uses a custom UI manager. This manager will lay out its fields on a diagonal and manipulate the natural focus order. In this exercise you will discover how to create a custom UI component and how UI managers control field layout and handle focus events.

1. Download and double-click CustomUserInterfaces_Lab 4.12.zip.

2. Open custom_manager_incomplete.zip.

3. In the IDE, open the custom_manager.jdw workspace.

4. Open the DiagonalApp.java source file.

5. Open the DiagonalManager.java source file.

6. From the DiagonalManager, implement the getPreferredWidth and getPreferredHeight methods so that they define the size of the manager.

7. Fill in the loop of DiagonalManager sublayout method so that each controlled field is laid out and arranged.

8. Complete the DiagonalManager nextFocus method so that focus does not leave the manager.
4.20 Create an SVG animation

In this exercise you will learn how to display an animated SVG on the BlackBerry® smartphone. This exercise uses APIs implemented using the JSR 226 standard for interactive and animated 2-D graphics on the Java® ME platform.

This exercise contains an SVG file, which includes the definitions for three distinct animations: a bouncing ball, a running stick figure, and an orbiting solar system. The exercise allows you to change which of the three animations you want to display at any time and provides a play control for starting each animation.

1. Install the sample application.
   1. Download the sample application svganimatordemo_4.13.zip.
   3. In Eclipse®, on the File menu, click Import.
   4. In the Import dialog box, expand the BlackBerry folder.
   5. Click Existing BlackBerry Projects into Workspace.
   6. Click Next.
   7. Click Browse.
   8. In the Open dialog box, navigate to the folder where you extracted the sample application.
   9. Click the SVGAnimatorDemo.jdp file.
   10. Click Open.
   11. Click Finish.

2. Run the sample application.
   1. In Eclipse, in the Navigator window, right-click the HelloWorldDemo folder.
   2. Click Activate for BlackBerry.
   3. Right-click the HelloWorldDemo folder again.
   4. Click Run As > BlackBerry Simulator.
   5. On the Home screen of the BlackBerry® Smartphone Simulator, click the Downloads folder.
   6. Click the Hello World Demo icon.
3. The `javax.microedition.m2g.SVGImage` class represents an SVG image that conforms to the W3C® SVG Tiny 1.2 profile. Observe how the private method `loadSVGimage()` in the `SVGScreen` class demonstrates how to create an `SVGImage` object from a file.

4. The `javax.microedition.m2g.SVGAnimator` class provides methods that you can use to load an animation from an `SVGImage` object and to control the animation. Observe how the application uses the `SVGAnimator` to switch between the three distinct animations that are defined in the `sample.svg` file using the `createAnimator()` method.

5. Start the animation by clicking a button on the screen.
4.21 Create an SVG Map application

In this exercise you will initialize the SVG variables needed to create the SVG Map application. Complete exercises 4.14 through 4.17. You will need Lab_18_SVG.zip.

1. Load the SVG image from the sample.svg file.

2. An InputStream object has been created to access this file. Use the InputStream object to create an SVGImage by calling a method inherited from ScalableImage and casting the returned object to an SVGImage.

   ```java
   _image = (SVGImage)SVGImage.createImage(is, null);
   ```

3. The method asks for an ExternalResourceHandler as a parameter. Use null instead of supplying an object for this parameter. In the SVGMapScreen constructor, complete the TODO step titled Load SVGImage.

   ```java
   _image = (SVGImage)SVGImage.createImage(is, null);
   ```

4. With an SVGImage, you can get the Document associated with this image to interact with the content. In the SVGMapScreen constructor, complete the TODO step titled Obtain Document.

   ```java
   _document = _image.getDocument();
   ```

5. The SVGAnimator renders updates and animations for the SVGImage. Use a static method in the SVGAnimator class to associate this animator with the net.rim.device.api.ui.Field component. In the SVGMapScreen constructor, complete the TODO step titled Create Animator.

   ```java
   _animator = SVGAnimator.createAnimator(_image
   net.rim.device.api.ui.Field
   ```
In this exercise you will create SVGElements and Fields.

1. Retrieve the field that displays the SVG Map.

2. Use a method in SVGAnimator to get the proper component, and then cast the method to the Field type. In the initializeUI() method of the SVGMapScreen class, complete the TODO step titled Get SVG Field.
   
   
   ```java
   _svgField = (Field)_animator.getTargetComponent();
   ```

3. Add the SVG Field you retrieved in step 1 to the SVGMapManager. In the initializeUI() method of the SVGMapScreen class, complete the TODO step titled Add SVG Field.

   ```java
   _mapManager.add(_svgField);
   ```

4. Use the document variable to get the root SVGElement, and then cast the root SVGElement to the SVGSVGElement type. In the initializeUI() method of the SVGMapScreen class, complete the TODO step titled Get Root SVG Element.

   ```java
   _svg = (SVGSVGElement)_document.getDocumentElement();
   ```

5. Use the root element to get the view box. Use a method inherited from SVGElement to retrieve the viewBox SVGRect trait. In the initializeUI() method of the SVGMapScreen class, complete the TODO step titled Get View Box.

   ```java
   _svgViewBox = _svg.getRectTrait("viewBox");
   ```

6. You can use the viewport SVGElement for selection and panning purposes. Use the Document object to retrieve the SVGElement named viewport, Cast to the appropriate type. In the initializeUI() method of the SVGMapScreen class, complete the TODO step titled Get ViewPort.

   ```java
   _viewportElement = (
   SVGElement)_document.getElementById("viewport");
   ```

7. To facilitate panning, the viewport SVGElement needs an event listener. Add the SVGMapScreen class as an eventListener to listen for the DOMActivate type. In the initializeUI() method of the SVGMapScreen class, complete the TODO step titled Add Event Listener.

   ```java
   _viewportElement.addEventListener("DOMActivate", this, false);
   ```

8. Add the SVGMapScreen class as an eventListener to listen for the DOMActivate type. In the initializeUI() method of the SVGMapScreen class, complete the TODO step titled Add Event Listener.

   ```java
   _viewportElement.addEventListener("DOMActivate", this, false);
   ```
4.23 Edit SVG traits

In this exercise you will edit SVG traits to change SVG elements.

1. Select the map in the application.

2. Extend the size of the border to give a visual cue to the user that the field is activated for panning.

3. The EventListener implementation can handle the DOMActivate event for the viewportElement variable. If the map is ready for panning, change the stroke-width trait of the SVGElement to 12 (represented as a String) so that the border becomes thicker. In the handleEvent() method of the SVGMapScreen class, complete the TODO step titled Select Map.

   `_viewportElement.setTrait("stroke-width", "12");`

4. When the map is panned, update the location by updating the viewport SVGElements x and y float traits to their new positions. In the update() method of the SVGMapScreen class, complete the TODO step titled Update Positions.

   `_viewportElement.setFloatTrait("x", _positionX);`
   `_viewportElement.setFloatTrait("y", _positionY);`
4.24 Build and run the SVG project

In this exercise you will save your changes from exercises 4.14 through 4.16 and run the application on the BlackBerry Smartphone Simulator.

1. In the Package Explorer view, right click the Lab_18_SVG project.

2. Click Activate for BlackBerry. The project folder icon changes to indicate that the project is activated.

3. In the Package Explorer view, right click on the Lab_18_SVG project.

4. Click Run BlackBerry Simulator.

5. When the BlackBerry Smartphone Simulator home screen appears, click Menu to view all applications and folders.

6. Use the arrow keys on your keyboard to navigate to the Downloads folder.

7. Enter the folder by pressing the Enter key on your keyboard.

8. Use the arrow keys on your keyboard to navigate to the generic application icon called SVG Lab.

9. Press the Enter key on your keyboard to invoke the application.