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 3: Optimization for mobile application development

This chapter outlines the optimal coding techniques for the BlackBerry® smartphone. This chapter demonstrates a number of coding techniques that you can use to write efficient and effective applications to meet the needs of the BlackBerry smartphone user.
3.1 Set up breakpoints

In this exercise you will use two breakpoints to pause an application. You will complete exercises 3.1 through 3.4. You will need Lab_17_Profiler.zip.

13. In the Package Explorer, right-click the Lab_17_Profiler project.

14. Click **Activate for BlackBerry**.

15. Expand the Lab_17_Profiler project, and then open **ProfilerDemo.java** in the editor.

16. In the **ProfilerDemoScreen** constructor, place a breakpoint on the line with “String string = “.

17. In the **ProfileDemoScreen** class constructor, complete the TODO step titled Set First Breakpoint.
   **Hint**: Place a breakpoint by pressing Ctrl+SHIFT+B.

18. Place a second breakpoint on the line with **labelField1 = new LabelField(string)**.

19. In the Package Explorer, right-click the Lab_17_Profiler project.

20. Click **Debug As > BlackBerry Simulator**.

21. When the BlackBerry Smartphone Simulator Home screen appears, click the **menu** button to display all applications and folders.

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

23. Enter the folder by pressing **ENTER** on your keyboard.

24. Use the arrow keys on your keyboard to navigate to the generic application icon with the title **Profiler Lab**.

25. Invoke the application by pressing **ENTER** on your keyboard.

26. When the application starts, the first breakpoint is reached and causes Eclipse to go into the Debug mode (if you are asked to switch to the Debug Perspective, select **yes**).

27. If you are not in the Eclipse® Debug View, on the **Window** menu, click **Open Perspective > Debug**.

28. On the **Window** menu, click **Show View > Other**.

29. Double click the **BlackBerry** folder.
30. Click **BlackBerry Profiler View**.

31. Click **OK**.
3.2 Analyze code

In this lab you will analyze code using the BlackBerry Profiler View. The application takes randomly generated integers, converts them to hexadecimal, and then displays the result.

1. In the BlackBerry Profiler View, click Options.
2. In the Method Attribution, click Cumulative.
3. In the What to profile drop-down list, click Time including native methods.
4. Click OK.
5. In the BlackBerry Profiler View, click Clear to ensure that only the code between the two breakpoints is profiled.
6. In the Run menu, click Resume to unsuspend the process. The application should run until the second breakpoint.
7. In the BlackBerry Profiler View, click Refresh.
8. Click the Method tab.
9. Find and expand Lab_17_Profiler. Observe that most of the time spent in this application was spent in the appendInHex() method.
10. Double-click on the method name to bring up the Source tab.
11. On the Source tab, expand ProfilerDemoScreen.appendInHex(). You can now see details about each line of code, including how long each line took to execute and what methods the line called.
12. Expand the line string += The hex value of: ;. This step includes a lot of redundant work. A significant amount of time is spent calling the StringBuffer constructor, toString(), and append() methods. When you expand the other steps, every line involves the creation of a new StringBuffer object, appending Strings (or other objects) to it, and then calling toString()!
3.3 Optimize code

In this exercise you will modify an application and observe the effect the changes have on performance.

1. Exit the simulator and return to the Java® editor.

2. Add code that calculates and displays the hex value of a given int using a StringBuffer instead of String objects. Use the StringBuffer parameter object to append the same five lines as the appendInHex() method.

3. In the appendInHexUsingStringBuffer() method of the ProfilerDemoScreen class, complete the TODO step titled Append Strings.
   
   ```java
   sb.append("The hex value of ");
   sb.append(value);
   sb.append(" is 0x");
   sb.append(Integer.toHexString(value));
   sb.append("
   ");
   ```

4. To compare the efficiency of the new appendInHexUsingStringBuffer() method with the existing appendInHex() method, call both methods from within the same for loop.

5. In the ProfilerDemoScreen constructor, complete the TODO step titled Build StringBuffer.
   
   ```java
   appendInHexUsingStringBuffer(sb, random);
   ```
Chapter 3: Optimization for mobile application development

Notes
In this exercise you will use the BlackBerry IDE profiler tool to analyze and compare two different methods of appending strings.

1. In the Package Explorer, right-click the Lab_17_Profiler project.
2. Click **Debug As > BlackBerry** Simulator.
3. When the BlackBerry Smartphone Simulator Home screen appears, click menu to display all applications and folders.
4. Use the arrow keys on your keyboard to navigate to the Downloads folder.
5. Enter the folder by pressing ENTER on your keyboard.
6. Use the arrow keys on your keyboard to navigate to the generic application **Profiler Lab** icon.
7. Invoke the application by pressing ENTER on your keyboard.
8. When the application starts, the first breakpoint is reached and causes Eclipse to go into Debug mode (if you are asked to switch to the Debug Perspective, click Yes). If you are not in the Eclipse Debug View, on the **Window** menu, click **Open Perspective > Debug**.
9. On the **Window** menu, click **Show View > Other**.
10. Double-click the **BlackBerry** folder.
11. Click **BlackBerry Profiler View**.
12. Click **OK**.
13. On the **Method** tab of the BlackBerry Profiler View, open Lab_17_Profiler.
14. Compare the percentage of time attributed to the `appendInHex()` method with the percentage of time attributed to the `appendInHexUsingStringBuffer()` method. Notice that the time spent in the `appendInHex()` method is about three times that of the time spent in the `appendInHexUsingStringBuffer()` method. This result is because the former involves constructing, appending to, and then returning a StringBuffer for each step, while the latter uses a single StringBuffer object.
15. Resume the BlackBerry Smartphone Simulator.
16. Exit the Profiler Lab application.
17. Restart the Profiler Lab application.
18. Let the application run until the first breakpoint.

19. In the BlackBerry Profiler View, click **Options**.

20. In the **What to profile** drop-down list, click **Number of Objects Created**.

21. Resume the application and let it run to the next breakpoint.

22. In the BlackBerry Profiler View, click **Refresh**.

23. Click the **Method** tab. Notice that the `appendInHex()` method created about 10 times as many objects as the `appendInHexUsingStringBuffer()` method. This results in decreased performance and inefficient use of memory.
3.5 Find a memory leak

In this exercise you will use the Memory Statistics View to locate an object leak. You will complete exercises 3.5 through 3.7. You will need Lab_15_MemoryLeak.zip.

1. In the Package Explorer View, expand the Lab_15_MemoryLeaks project.

2. Open MemoryLeak.java in the editor.

3. In the fieldChanged() method of the breakListener anonymous FieldChangeListener class, place the cursor on the line with Dialog.inform(Breakpoint here.).

4. On the Run menu, click Toggle Breakpoint to add a breakpoint to the current line.

5. In the Package Explorer, right-click the Lab_15_MemoryLeak project.

6. Click Activate for BlackBerry. The project folder icon changes indicating that the project is activated.

7. Right-click again, and then click Debug As > BlackBerry Simulator.

8. When the BlackBerry Smartphone Simulator Home screen appears, click the menu button to display all applications and folders.

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

10. Enter the folder by pressing ENTER on your keyboard.

11. Use the arrow keys on your keyboard to navigate to the generic application Memory Leak Lab icon.

12. Invoke the application by pressing ENTER on your keyboard.

13. In the Memory Leak Lab application, click the Break button. This causes the application to hit the breakpoint and causes Eclipse to go into Debug mode.

14. If you are prompted to switch to Debug View, click Yes. If you are not in the Eclipse Debug View, on the Window menu, click Open Perspective > Debug.

15. On the Window menu, click Show View > Other.

16. Expand the BlackBerry folder.

17. Multiselect the BlackBerry Memory Statistics View and BlackBerry Objects View by holding CTRL and clicking each of them.
3.6 Create snapshots

In this exercise you will use the BlackBerry Memory Statistics View and BlackBerry Objects View to create snapshots.

1. In both the BlackBerry Memory Statistics View and BlackBerry Objects View, click the Refresh button in the top right corner of the pane. The most current information on the suspended process appears.

2. Look at the BlackBerry Objects View. The large number of objects displayed belong to the many different threads and processes that are running. Look for the objects that belong to Lab_15_MemoryLeak.

3. In the Debug View, find the suspended thread running Lab_15_MemoryLeak. A number of brackets appears beside the name. This is the process number.

4. In the BlackBerry Objects View, click Filter.

5. In the Process field, enter the process number.

6. Click OK. You should now see only objects belonging to the process.

7. Click Snapshot at the top right of both the BlackBerry Memory Statistics View and BlackBerry Objects View.

8. In the Run menu, click Resume. Snapshots let you easily compare data between two points in the execution of an application. Take a snapshot after initialization (but before you do any work) to establish a baseline for comparison.
Chapter 3: Optimization for mobile application development

Notes
3.7 Compare data

In this exercise you will compare new program data against the snapshots you took in exercise 3.6.

1. In the running Memory Leak Lab application, clear the Breakpoint here dialog box, and then click the Change Image button 10 times.

2. Click Break, and then click to suspend the process.

3. Click Refresh in the BlackBerry Memory Statistics View.

4. Click Compare. The view now displays the statistical differences in terms of memory usage between the time of the last snapshot and the current point of execution. You can switch between seeing this comparison and the total numbers by clicking Compare and Refresh respectively. In the RAM statistics, the number of objects and bytes in use have increased significantly. The allocated number has also gone up, so the free number has gone down accordingly. This indicates that something is causing memory usage to go up even though it appears not much work is being done by the process. This fact alone is not always indicative of a problem, but in this case the difference in memory usage is substantial enough to warrant further investigation. A process whose memory usage continues to grow over time and under normal use is an indication of a potential memory leak.

5. Click Refresh in the BlackBerry Objects View, and then click Filter.

6. Change the Snapshot Filter option to Compare to Snapshot.

7. Enter the process number.

8. Click OK. All the objects that were created since the last snapshot was taken now appear.

9. Click on the top of the Size column to sort the objects by size in descending order. The largest objects are of type byte[] or Object[]. Scroll down until you see a significant number of objects of type com.rim.samples.labs.memoryleak.MemoryLeakScreen. Right-click on one of them, and then choose Show where id=xxx allocated.

10. The preceding steps take you to the line where that object was created. The line is in the FieldChangeListener for the Change Image button. Every time you click the button, a new instance of the MemoryLeakScreen is created and pushed on to the stack, and the existing screen is not closed. To plug this memory leak, stop the creation of new screens every time the picture is changed. Change the bitmap that _bitmapField displays. First, delete the code that is causing the memory leak, and then set _currentBitmap to the correct bitmap. Use a method in the BitmapField class to change the picture.
11. In the `fieldChanged()` method of the `pictureListener` anonymous `FieldChangeListener` class, complete the TODO step titled Plug Memory Leak.

```java
if (_currentBitmap == _colourBitmap) {
    _currentBitmap = _blackBitmap;
} else {
    _currentBitmap = _colourBitmap;
}
_bitmapField.setBitmap(_currentBitmap);
```