Lab 4: Creating User Interfaces I

CMER Centre for Mobile Education and Research

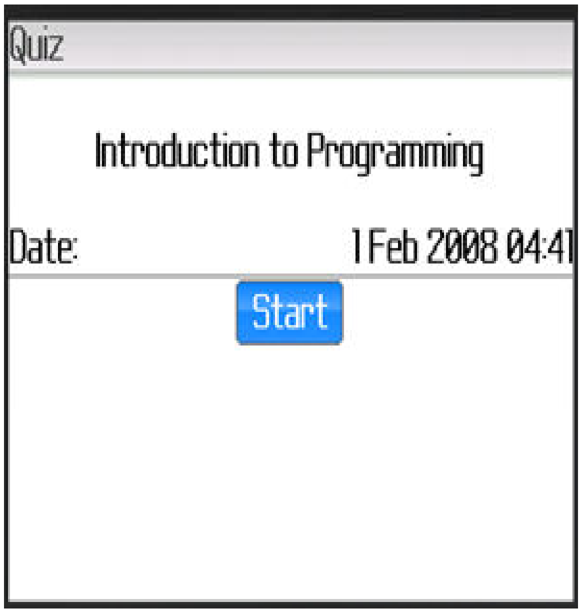
Objective

The objective to this lab is to get experience with:

1. BlackBerry User interface APIs
2. Displaying user interface components including button fields, choice fields, option fields, data fields, gauge fields and separator field.
3. Handling events in user interface components
4. Develop a quiz application using these components

Experiment 1 Develop a quiz application using User Interface components

In this experiment, you will learn how to write an application for BlackBerry devices using the BlackBerry User Interface APIs. You will develop a quiz application and get experience with button fields, choice fields, option fields, data fields, gauge fields and separator field. Figures 1 and 2 demonstrate how your application should be appeared on the screen.

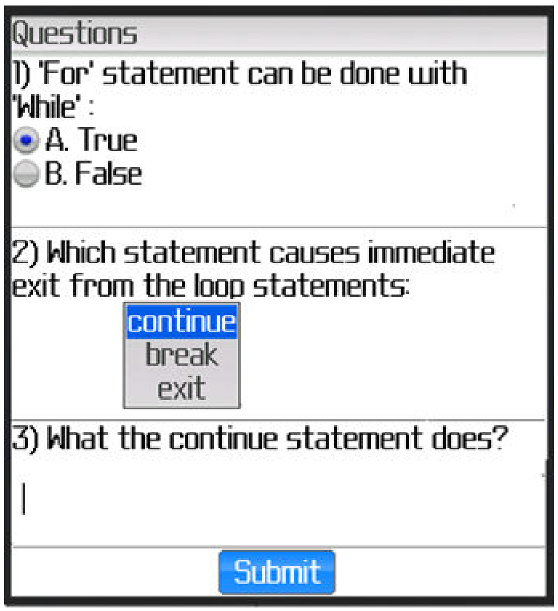


Quiz

Introduction to Programming

Date: 1 Feb 2008 04:41

Figure 1- First screen



Questions

1) 'For' statement can be done with 'While' :

● A. True

○ B. False

2) Which statement causes immediate exit from the loop statements:

continue
break
exit

3) What the continue statement does?

Figure 2 – Second screen

To start the experiment, do the following steps:

1. Create a class extended from **net.rim.device.api.ui.UiApplication** class. Name it "Quiz_App".

2. Write the constructor and main() methods and add the required codes to these methods.
3. Create the first screen class and name it "FirstScreen". Figure 1 depicts how it should look like.

```
final class FirstScreen extends MainScreen
   {
      ButtonField _startBtn;
      public FirstScreen() {  }
}
```

4. FirstScreen consists of the following UI components: RichTextField for displaying "Introduction to Programming", LabelField for the title "Quiz", DateField for "Date: ", SeparatorField and ButtonField. In order to handle UI objects events, you must add the listener interface to the class. This is done by implementing "*FieldChangeListener*" interface. By doing that, your class is able to listen to UI objects events and does the appropriate action when an event is fired. For instance, in this experiment when you click on the "Start" button, the second screen must be displayed (Fig. 2).

```
final class FirstScreen extends MainScreen implements FieldChangeListener
{
    public FirstScreen()
    {
         super();
         ……
    }
}
```

5. When you implement the listener interface in your class, you must override its routines as well.

```
final class FirstScreen extends MainScreen implements FieldChangeListener
{
          public FirstScreen()
          {….}

          public void fieldChanged (Field field, int context)
          {  ….  }
```

6. Set the title. This is done by LabelField component.

```
final class FirstScreen extends MainScreen implements FieldChangeListener
{
    public FirstScreen()
    {
         super();
```

```
        LabelField title = new LabelField("Quiz", LabelField.ELLIPSIS |
        LabelField.USE_ALL_WIDTH);
        setTitle(title);

    }
}
```

7. RichTextField creates a read-only field that can be formatted with different styles and font. Create an instance of this object and add it to the screen as follows:

```
add (new RichTextField("Introduction to Programming ",
RichTextField.TEXT_ALIGN_HCENTER  | RichTextField.NON_FOCUSABLE));
```

   TEXT_ALIGN_HCENTER and NON_FOCUSABLE in the above code are two field styles defined for RichTextField. They determine the field is centered horizontally and is non-focusable. For getting more information, go to "Help" menu in JDE, then select "API Reference".

8. The following code creates a blank line.

```
add(new RichTextField(""));
```

9. DateField is an editable component for presenting date and time. Create an instance of this component and add it to the screen as follows:

```
add(new DateField("Date: ",System.currentTimeMillis() + 3600000,
DateField.NON_FOCUSABLE | DateField.DATE_TIME));
```

   DateField.NON_FOCUSABLE and DateField.DATE_TIME are DateField styles determining that the component is non-focusable and represents date and time. You can also change DateField.DATE_TIME to DateField.DATE for presenting the date only or to DateField.TIME for presenting the time.

10. ButtonField displays a button. Create an instance and add it to the screen as follows.

```
_startBtn = new ButtonField("Start", ButtonField.FOCUSABLE |
ButtonField.FIELD_HCENTER);
_startBtn.setChangeListener(this);
add(_startBtn);
```

   Also, you must use a "*FieldChangeListener*" interface to get notifications when the user presses the button (done in Steps 4 and 5) and set a listener for this component using "setChangeListener" method as follows:

```
    _startBtn.setChangeListener(this);
```

11. At the last, your code looks like as follows:

```
final class FirstScreen extends MainScreen implements FieldChangeListener
{
    public FirstScreen()
    {
         super();
         LabelField title = new LabelField("Quiz", LabelField.ELLIPSIS |
         LabelField.USE_ALL_WIDTH);
         setTitle(title);

         add(new RichTextField(""));
         add(new RichTextField("Introduction to Programming ",
         RichTextField.TEXT_ALIGN_HCENTER |
         RichTextField.NON_FOCUSABLE  ));

         add(new RichTextField(""));

         add(new DateField("Date:  ",System.currentTimeMillis() + 3600000,
         DateField.NON_FOCUSABLE | DateField.DATE_TIME));

         add(new SeparatorField());

         _startBtn = new ButtonField("Start",ButtonField.FOCUSABLE |
         ButtonField.FIELD_HCENTER);
         _startBtn.setChangeListener(this);
         add(_startBtn);

    }
    public void fieldChanged (Field field, int context)
    { …. }

}
```

12. The next step is creating the second screen (See Figure 2). For this purpose, extend MainScreen and name “QuestionScreen”. Set a title like “Questions” for the screen.

```
final class QuestionScreen extends MainScreen {
      public QuestionScreen()
      {
         super();
         LabelField title = new LabelField("Questions", LabelField.ELLIPSIS |
         LabelField.USE_ALL_WIDTH);
         setTitle(title);
         …
       }
}
```

13. As displayed in Figure 2, the first question is a multiple-choice question. For this purpose, use a RichTextField component for the question and a RadioButtonGroup component for answers.

```
//create a question
add( new RichTextField("1. 'For' statement can be done with 'While' : ",
RichTextField.TEXT_ALIGN_LEFT | RichTextField.NON_FOCUSABLE));

//create answers
RadioButtonGroup Q1 = new RadioButtonGroup();
add( new RadioButtonField("A. True", Q1,false,RadioButtonField.FOCUSABLE));
add( new RadioButtonField("B. False", Q1,false,RadioButtonField.FOCUSABLE));
```

RadioButtonField enables users to select only one entry. A set of RadioButtonFields can be grouped in one RadioButtonGroup field. First, construct a RadioButtonGroup, then create two new RadioButton instances for this group with label, initial state and style and add it to screen.

14. The second question is a choice field which is similar to a drop-down list. For this purpose, construct an ObjectChoiceField component and add it to the screen as follows:

```
add( new ObjectChoiceField(
        "2. Which statement causes immediate exit from the loop statements:",
         new String[] { "continue", "break", "exit" },
         0,
         ObjectChoiceField.FIELD_HCENTER)
        );
```

The first argument in ObjectChoiceField is label for this field. The second argument presents choices for this field. In this experiment, you have three choices for this question including “continue”, “break” and “exit”. This argument can be any type of object such as Integer, String and so on. The next argument is the index of the initially selected value. In this experiment, for example, zero means the initial selected option is “continue”. The last argument is the style value for the newly created field.

15. The third question which is an essay question, uses an edit field. An EditField enables users to type text in the field. So, simply construct an EditField with label, initial contents and style, and add it to the screen a s follows:

```
add( new EditField( "3. What the continue statement does?" ,
                    "" , 20 ,
                    EditField.FOCUSABLE));
```

16. Construct a button field and add it to the screen.

```
ButtonField _btn = new ButtonField("Submit",ButtonField.FOCUSABLE |
                                   ButtonField.FIELD_HCENTER);
add(_btn);
```

17. At the last, you will have the following code for the second screen, ”QuestionScreen”.

```
final class QuestionScreen extends MainScreen {
    public QuestionScreen()
    {
        super();
         LabelField title = new LabelField("Questions", LabelField.ELLIPSIS |
                                           LabelField.USE_ALL_WIDTH);
        setTitle(title);

        add(new RichTextField("1. 'For' statement can be done with 'While' : ",
            RichTextField.TEXT_ALIGN_LEFT | RichTextField.NON_FOCUSABLE));

         RadioButtonGroup Q1 = new RadioButtonGroup();
         add(new RadioButtonField("A. True", Q1, false,
                                  RadioButtonField.FOCUSABLE));
         add(new RadioButtonField("B. False", Q1 ,false,
                                  RadioButtonField.FOCUSABLE));

         add( new RichTextField(""));
        add(new SeparatorField());

        add( new ObjectChoiceField(
                "2. Which statement causes immediate exit from the loop statements:",
                new String[] { "continue", "break", "exit" },0
                , ObjectChoiceField.FIELD_HCENTER));

         add( new RichTextField(""));
         add(new SeparatorField());

         add( new EditField("3. What the continue statement does?" ,"" ,20
                        ,EditField.FOCUSABLE));

         add( new RichTextField(""));
         add(new SeparatorField());

          ButtonField _btn = new ButtonField( "Submit", ButtonField.FOCUSABLE |
                                           ButtonField.FIELD_HCENTER);
         add(_btn);
    }
  }
```

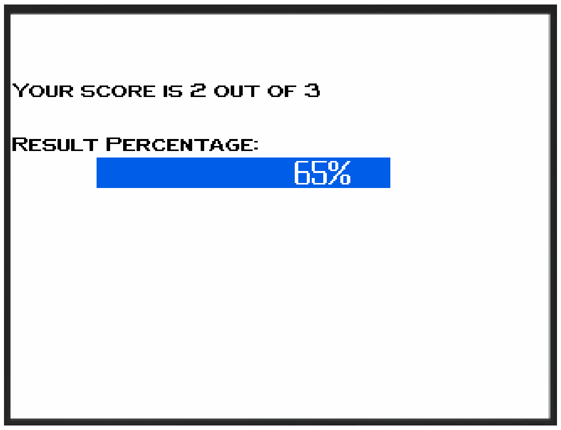
18. The only part remained for this experience is invoking the second screen when a user clicks on the "Start" button. For this purpose, just you should pop the first screen and push the second screen. These methods are invoked in "fieldChanged" method, the listener interface's method.

```
public void fieldChanged(Field field, int context) {
        if(field == _startBtn) {
            UiApplication.getUiApplication().popScreen(this);
            QuestionScreen qs = new QuestionScreen();
             UiApplication.getUiApplication().pushScreen (qs);
        }
}
```

19. Demo your work to the TA. [**5 marks**]

Exercise 1 Adding more advance features to your application

In this exercise, you complete your application developed in the Experiment 1 by adding some advance features such as adding GaugeField, changing font, and adding another screen to display the result of the quiz. The last screen which you add will look like follows:


YOUR SCORE IS 2 OUT OF 3

RESULT PERCENTAGE:

65%

Figure 3- Result screen

To do this exercise, the suggestion process is as follows:

- Reuse the code in the Experiment 1
- Implement a listener interface for the "QuestionScreen" class just like what you did for the "FirstScreen" class. This enables your class to listen to the "Submit" button event when a user clicks on this button. Write the required codes for handling this event. Perhaps, it is "poping" and "pushing" the screen.
- Add another screen for displaying the result, name it "ResultScreen"
- In the constructor of this class, write the required code as follows:

- Create an instance of the "RichTextField" component and add it to the screen to display the result
- Create an instance of a "GaugeField" component and add it to the screen. To create a GuageField, use the following format:

```
GaugeField percentGauge =
              GaugeField(String label, int min, int max,
                      int start, long style);
```

```
label - Optional label for the gauge (may be null).
min - Bottom of the value range.
max - Top of the value range.
start - Initial progress level of this field.
style - Style value(s) for this field.
```

For getting more information, just click on Help>API Reference on JDE menu toolbar.

- Implement the code to calculate the result. To note that each question in the quiz displayed in Figure 2 has one mark. To retrieve the index of the currently selected answer for "RadioButtonGroup" and "ObjectChoiceField" use getSelectedIndex() method.

  The third question is an essay question which can not be calculated. You can always assume one mark for this question under any circumstances.

- Demo your work to the TA. [**5 marks**]

Bonus Changing the font

In this exercise, you get the experiment changing the font size. The necessary classes and methods are included in net.rim.device.api.ui package. To accomplish this, do the following tasks: [**2 marks**]

1. Get the default font and the style.

```
Font font = Font.getDefault();
int style =  font.getStyle();
```

2. Get a list of the system font families

```
FontFamily fontFamily[] = FontFamily.getFontFamilies();
```

3. Create a font instance using getFont (). With the getFont () method two parameters should be specified: style and size. You already got the style from the default font in Step1. Select any size which you want.

```
Font font1 = fontFamily[0].getFont(style ,12);
```

4. The RichTextField only accepts an array of fonts. Thus, you should create an array of fonts and assign the font obtained in Step 3 to the first element

```
Font fonts[] = new Font[1];
fonts[0] = font1;
```

5. Pass the fonts as the fourth argument into the RichTextField constructor

```
RichTextField richTextField =
        new RichTextField (text, offset, attribute, fonts, style);
```

6. You can also set the "font1" to the default font to be used in your application

```
Font.setDefaultFont(font1);
```