Lab 7: Timer

Objective

The objective to this lab is to get experience with:

1. Timer and TimerTask
2. The device's vibration, audio system and led flash
3. Develop an application on the Blackberry device using Timer

Experiment 1 Timer and TimerTask.

In this experiment, you will learn how to use timers. Timers are used for time-based events. They are a single background thread which allows scheduling tasks for further execution. Tasks are defined in TimerTask object. A task can be scheduled for either one time execution or repeated execution at time intervals. Timer and TimerTask are two classes defined in *Java.util* package.
In this experiment, you simulate a message log application using the Timer and TimerTask classes: your device receives messages at regular time intervals. Incoming messages are demonstrated by the device vibration, a sound and the led flash. Then the message is shown on the screen.

To start the experiment, do the following steps:

1. Create a class extended from **net.rim.device.api.ui.UiApplication** class. Name it "lab7_ex1".
2. Write the constructor and main () methods and add the required codes to these methods.

```
class lab7_ex1 extends net.rim.device.api.ui.UiApplication {
     public FirstScreen screen;
     public RichTextField msg, msg1 ;

    public lab7_ex1()
    {
       screen  = new FirstScreen();
       pushScreen(screen );
    }

    public static void main(String[] args) {
       lab7_ex1 theApp = new lab7_ex1();
       theApp.enterEventDispatcher();
    }
     }
```

3. Create the first screen class and name it “FirstScreen”. It includes RichTextField objects to display the messages and LabelField to show the title. Figure 1 depicts how it should look like.

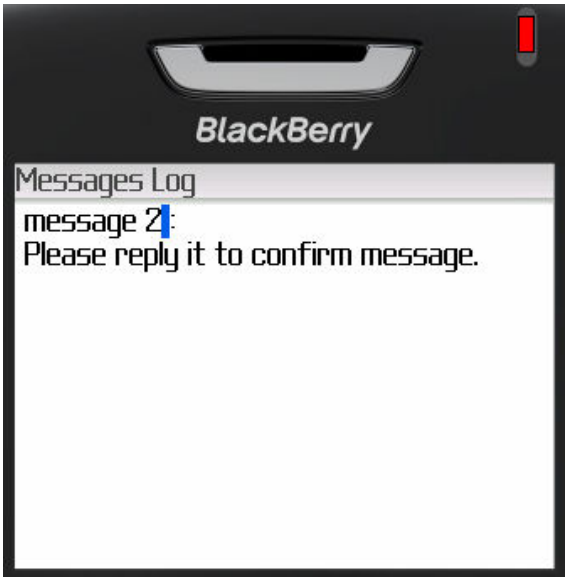

Figure 1- First Screen

```
class lab7_ex1 extends net.rim.device.api.ui.UiApplication {

        ..........

    final class FirstScreen extends MainScreen     {

            public FirstScreen()
        {
            super();
             LabelField title =
                    new LabelField("Phone calls",   LabelField.USE_ALL_WIDTH);
            setTitle(title);
            msg = new RichTextField("message 1");
            add(msg);
            msg1 = new RichTextField("This is a test");
            add(msg1);
        }
    }
}
```

4. In order to simulate incoming messages, you use Timer and TimerTask. Thus, your application receives messages at time intervals, for instance, every 4 seconds. While the device is receiving a new message, it should vibrate along with sound and flash the led. For this purpose, you create a new instance of Timer

and TimerTask in the main class (or create a procedure called startTimer() and then call it in the main class).

```
public lab7_ex1()
{
    screen  = new FirstScreen();
    pushScreen(screen );

    startTimer();

}

public void startTimer() {

}
```

5. In the startTimer(), you should do the following steps:
   1. Import the *java.util.Timer* and *java.util.TimerTask* to the class file
   2. Construct a new instance of Timer class.
      ```
      timer = new Timer();
      ```
   3. Schedule the timer in repeated execution at time intervals, e.g. 4 seconds
      To note that time intervals are represented in milliseconds.

      ```
      timer.schedule (task, 2000, 4000);
      ```

      Thus, regarding to the schedule, your timer waits 2 seconds before executing, then it executes the task every 4 seconds.
   4. Define a task for your timer. For this purpose, construct a new instance of TimerTask that implements the run method.

      ```
      task = new TimerTask() {

          private boolean isPaused;
          private int count=1;

          public void run() {

          }
      };
      ```

   5. In run method, do the following task:
      - Set a flag, name it *"isPaused*". It allows your timer's task to switch between on and off state Thus, in **on-state** you do the following :
      - Start vibration. To do this:
        First import *net.rim.device.api.system.Alert* to your class. Alert class provides access to the device's audio and vibration system.

        Second, add the following to the run method.
        ```
        Alert.startVibrate(75);
        ```

It starts a vibrate alert for 75 milliseconds.

- Starts a buzzer alert. But before that, you should specify the tunes. It is just an array of short integer.

```
short[] tuneAudio =      { 300, 50, 500, 50, 300, 50, 500, 50, 300, 50  };

Alert.startBuzzer(tuneAudio, 100);
```

The second argument in the method specifies the volume which is a number between 0 and 100.

- Change the state of the led to Blinking. Before that, import net.rim.device.api.system.LED.

```
LED.setState(LED.STATE_BLINKING);
```

- Set the text for RichTextField object created in the FirstScreen's constructor.

```
count++;
msg.setLabel(" message "+count );
msg1.setText( " Please confirm." );
```

- In off-state, do the following :

```
Alert.stopVibrate();
Alert.stopAudio();
LED.setState(LED.STATE_OFF);
```

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

Listing 1 – lab7_ex1.java:

```
import net.rim.device.api.system.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import java.util.Timer;
import java.util.TimerTask;
import net.rim.device.api.system.Alert;
import  net.rim.device.api.system.LED;

class lab7_ex1 extends net.rim.device.api.ui.UiApplication {
   private Timer timer;
   private TimerTask task;
   public FirstScreen screen;
   public RichTextField msg, msg1 ;
```

```
public lab7_ex1()  {
   screen  = new FirstScreen();
   pushScreen(screen );
   startTimer();
}

 public static void main(String[] args) {
   lab7_ex1 theApp = new lab7_ex1();
   theApp.enterEventDispatcher();
}

final class FirstScreen extends MainScreen     {

   public FirstScreen()
   {
      super();
      LabelField title =
                  new LabelField("Messages Log", LabelField.USE_ALL_WIDTH);
      setTitle(title);
      msg = new RichTextField("message 1");
      add(msg);
      msg1 = new RichTextField("This is a test");
      add(msg1);
   }

}


// Stops the timer
/*  private void stopTimer() {
   if (timer != null) {
      timer.cancel();
   }
}*/

 public void startTimer() {

      // Create a task to be run
      task = new TimerTask() {
         private boolean isPaused;
         private int count=1;

         public void run() {

            short[] tuneAudio =      { 300, 50, 500, 50, 300, 50, 500, 50, 300, 50  };

            if (isPaused) {         //on-state
               count++;
               isPaused = false;
               Alert.startVibrate(75);
               Alert.startBuzzer(tuneAudio, 100);
```

```
                LED.setState(LED.STATE_BLINKING);
                msg.setText(" message "+count );
                msg1.setText( " Please confirm." );
            } else {                        //off-state
                isPaused = true;
                Alert.stopVibrate();
                Alert.stopAudio();
                LED.setState(LED.STATE_OFF);

            }
        }
    };

    timer = new Timer();
    timer.schedule(task, 2000, 4000);

  }

}
```

Exercise 1 Re-implement the quiz application developed in Lab#4.

In this exercise, you reuse the code experienced in Lab #4: Exercise #1, Quiz application. But this time you create a non-interactive gauge that displays visually the remaining time to submit the quiz. A non-interactive mode is used as an "activity indicator" or "progress indicator" to give the user feedbacks on the state of long-running operation.

To accomplish this exercise, the following tasks are suggested:

1. Reuse the code written in the Lab#4: Exercise#1.
2. Construct a new instance of the GaugeField class and add it to the top of the second screen.
3. The gauge is updated every second using a Timer/TimerTask object. Set the label of timer to the "Time Left".
4. When the time for the test is over (the remaining time is zero), your application should demonstrate the Result screen along with the result of the quiz.
5. Demo your work to the TA. [**5 marks**]