Mobile Devices in Software Engineering

Lab 2

Objective

The objective of this lab is to:

1. Get you started with network based mobile applications
2. Get you started with persistent storage on mobile devices

Experiment 1

In this experiment we will be using network connections and persistent storage on our mobile application to create an application that will store a list of stocks and retrieve the stock price.

Exercise 1

The first step in creating the stock quote application is a Stock.java class. This class will parse the string that is returned from the stock quotes server. The name, time, and price of the stock are extracted from the line.

Stock.java

```
public class Stock {
   private static String name, time, price;
   // Given a quote from the server,
   // retrieve the name,
   //price, and date of the stock
   public static void parse(String data) {
     int index = data.indexOf('"');
     name = data.substring(++index, (index = data.indexOf('"',
index)));
     index +=3;
     time = data.substring(index, (index = data.indexOf('-',
index))-1);
     index +=5;
     price = data.substring(index, (index = data.indexOf('<',
index)));
   }
  // get the name of the stock from
  // the record store
   public static String getName(String record) {
     parse(record);
     return(name);
   }
  // get the price of the stock from
  // the record store
   public static String getPrice(String record) {
     parse(record);
     return(price);
   }

}
```

Take some time to read the above code and become familiar with the actions it takes.

Let us now add this code to our application.

1. Run the Sun Java Wireless Toolkit for CLDC.
2. Click on **New Project...**
3. Name the Project Name and MIDlet class name 'QuotesMIDlet'
4. Proceed to the directory displayed in the Wireless Toolkit console instructing you the location to place the source files
5. Copy the above code into Stock.java in the given folder

Now we will need to store the stock quotes the users input into our application. We will do this through the use of Record Management System (RMS). It will allow us to open, add, modify, and enumerate stocks that we store in the database.

StockDB.java

```
import javax.microedition.rms.*;
import java.util.Enumeration;
import java.util.Vector;
import java.io.*;

public class StockDB {
   RecordStore recordStore = null;
   public StockDB() {}

   // Open a record store with the given name
   public StockDB(String fileName) {
      try {
        recordStore =
              RecordStore.openRecordStore(
                       fileName, true);
      } catch(RecordStoreException rse) {
        rse.printStackTrace();
      }
   }
// Close the record store
   public void close()
          throws RecordStoreNotOpenException,
                       RecordStoreException {
        if (recordStore.getNumRecords() == 0) {
            String fileName =
                           recordStore.getName();
            recordStore.closeRecordStore();
            recordStore.deleteRecordStore(
                                    fileName);
```

```
        } else {
            recordStore.closeRecordStore();
        }
    }

   // Add a new record (stock)
   // to the record store
   public synchronized void
             addNewStock(String record) {
      ByteArrayOutputStream baos = new
            ByteArrayOutputStream();
      DataOutputStream outputStream = new
            DataOutputStream(baos);
      try {
          outputStream.writeUTF(record);
      }
      catch (IOException ioe) {
          System.out.println(ioe);
          ioe.printStackTrace();
      }
      byte[] b = baos.toByteArray();
      try {
          recordStore.addRecord(b,
                           0, b.length);
      }
      catch (RecordStoreException rse) {
          System.out.println(rse);
          rse.printStackTrace();
      }
    }

    // Enumerate through the records.
    public synchronized
                RecordEnumeration enumerate()
        throws RecordStoreNotOpenException {
       return recordStore.enumerateRecords(
            null, null, false);
    }
}
```

Take a moment now to read and understand the above code. Pay special attention to the addNewStock() method to get an idea of how adding records to the RMS works.

6. Copy the above code into the 'src' folder for your application

The final piece of code for this application has commands allowing us to add stocks and list the stocks we add. It also connects to the Yahoo! Server to retrieve the stock quote we will be using. It uses the previous two classes to retrieve data and parse the quote from the server.

QuotesMIDlet.java

```
import javax.microedition.rms.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import javax.microedition.io.*;
import java.io.*;
import java.util.Vector;

public class QuotesMIDlet extends MIDlet implements
CommandListener, Runnable {
    Display display = null;
    List menu = null; // main menu
    List choose = null;
    TextBox input = null;
    Ticker ticker = new Ticker("Database Application");
    String quoteServer =
"http://download.finance.yahoo.com/d/quotes.csv?s=";
    String quoteFormat = "&f=slc1wop"; // The only quote
format supported

    static final Command backCommand = new Command("Back",
Command.BACK, 0);
    static final Command mainMenuCommand = new Command("Main",
Command.SCREEN, 1);
    static final Command saveCommand = new Command("Save",
Command.OK, 2);
    static final Command exitCommand = new Command("Exit",
Command.STOP, 3);
    String currentMenu = null;

    // Stock data
    String name, date, price, pr, userInput;

    // record store
    StockDB db = null;

    public QuotesMIDlet() { // constructor
    }

    // start the MIDlet
    public void startApp()
      throws MIDletStateChangeException {
      display = Display.getDisplay(this);
      // open a db stock file
      try {
        db = new StockDB("mystocks");
      } catch(Exception e) {}
      menu = new List("Stocks Database", Choice.IMPLICIT);
      menu.append("List Stocks", null);
```

```
      menu.append("Add A New Stock", null);
      menu.addCommand(exitCommand);
      menu.setCommandListener(this);
      menu.setTicker(ticker);

      mainMenu();
    }

    public void pauseApp() {
      display = null;
      choose = null;
      menu = null;
      ticker = null;

      try {
        db.close();
        db = null;
      } catch(Exception e) {}
    }

    public void destroyApp(boolean unconditional) {
      try {
        db.close();
      } catch(Exception e) {}
      notifyDestroyed();
    }

    void mainMenu() {
      display.setCurrent(menu);
      currentMenu = "Main";
    }

    // Construct a running ticker
    // with stock names and prices
    public String tickerString() {
       StringBuffer ticks = null;
       try {
          RecordEnumeration enumm = db.enumerate();
          ticks = new StringBuffer();
          while(enumm.hasNextElement()) {
            String stock1 = new String(enumm.nextRecord());
            ticks.append(Stock.getName(stock1));
            ticks.append(" @ ");
            ticks.append(Stock.getPrice(stock1));
            ticks.append("    ");
          }
       } catch(Exception ex) {}
       return (ticks.toString());
    }
```

```
// Add a new stock to the record store
    // by calling StockDB.addNewStock()
    public void addStock() {
      input = new TextBox("Enter a Stock Name:", "", 5,
      TextField.ANY);
      input.setTicker(ticker);
      input.addCommand(saveCommand);
      input.addCommand(backCommand);
      input.setCommandListener(this);
      input.setString("");
      display.setCurrent(input);
      currentMenu = "Add";
    }

    // Connect to quote.yahoo.com and
    // retrieve the data for a given
    // stock symbol.
    public String getQuote(String input) throws IOException,
NumberFormatException {
      String url = quoteServer + input + quoteFormat;
      StreamConnection c =
(StreamConnection)Connector.open(url, Connector.READ_WRITE);
      InputStream is = c.openInputStream();
      StringBuffer sb = new StringBuffer();
      int ch;
      while((ch = is.read()) != -1) {
        sb.append((char)ch);
      }
      return(sb.toString());
    }

    // List the stocks in the record store
    public void listStocks() {
        choose = new List("Choose Stocks", Choice.MULTIPLE);
        choose.setTicker(
             new Ticker(tickerString()));
        choose.addCommand(backCommand);
        choose.setCommandListener(this);
      try {
         RecordEnumeration re = db.enumerate();
         while(re.hasNextElement()) {
           String theStock =
                   new String(re.nextRecord());
           choose.append(Stock.getName(
                                theStock)+" @ "
          + Stock.getPrice(theStock), null);
         }

      } catch(Exception ex) {}
      display.setCurrent(choose);
      currentMenu = "List";
   }
```

```
// Handle command events
   public void commandAction(Command c,   Displayable d) {
      String label = c.getLabel();
      if (label.equals("Exit")) {
         destroyApp(true);
      } else if (label.equals("Save")) {
          if(currentMenu.equals("Add")) {
              // add it to database
            try {
            new Thread(this).start();
            } catch(NumberFormatException se) {
            }
            mainMenu();
          }
      } else if (label.equals("Back")) {
          if(currentMenu.equals("List")) {
            // go back to menu
            mainMenu();
          } else if(currentMenu.equals("Add")) {
            // go back to menu
            mainMenu();
          }
      } else {
         List down = (List)display.getCurrent();
         switch(down.getSelectedIndex()) {
           case 0: listStocks();break;
           case 1: addStock();break;
         }
      }
  }

  public void run() {
      try{
      String userInput = input.getString();
      pr = getQuote(userInput);
      db.addNewStock(pr);
        ticker.setString(tickerString());
      }catch(Exception e){}
  }
}
```

There is a lot of code, so take a minute to read it all over and understand what is happening. Note the thread that is launched when a user clicks to add a stock. This is because operations that will potentially lock the application should not be run in the commandAction() method.

7. Copy the above code into the ‘src’ folder of your application.

8. Now click on **Build** in the Wireless Toolkit and then **Run**

Explore the application and add a couple of stocks. Show your work to the instructor when you are done.

Experiment 2

For the second part of this lab add some extra features to the application. Allow the user to remove a selected stock from their stock list.