Lab 10: Personal Information Management

Objective

The objective to this lab is to get experience with:

1. Personal Information Management (PIM) APIs
2. BlackBerry personal digital assistant profile (PDAP) APIs
3. Develop an application using these APIs

Experiment 1 Develop an application using PIM

In this experiment, you will learn how to write application using PIM APIs for handheld devices. PIM APIs (javax.microedition.pim) and PDAP APIs (net.rim.blackberry.api.pdap) enable you to access the calendar task, event and address book on the handheld device. These APIs are controlled API. This means that in order for you to be able to load them onto the BlackBerry Wireless Handheld devices, you must obtain a code signature from Research In Motion (RIM) and sign the application using that. However, these APIs can be run in the simulator. In this experiment, you write an application for BlackBerry devices using the PIM objects. In this application, you access the address book. Figures 1 and 2 demonstrate how your application should appear on the screen.

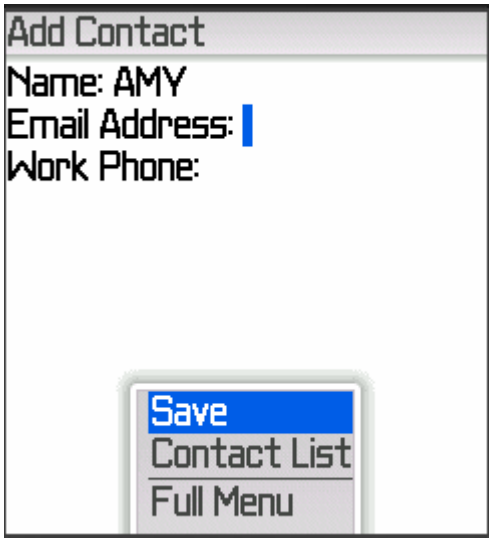

Figure 1- Contact screen

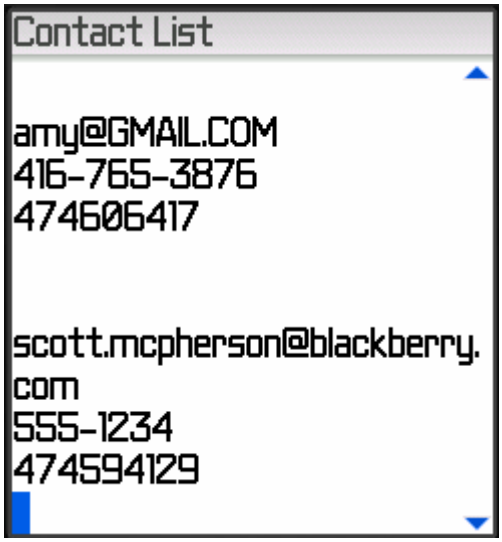

Figure 2- Contact List

To start the experiment, do the following steps:

1. Create a class extended from **net.rim.device.api.ui.UiApplication** class. Name it "Lab10".
2. Write the constructor and main() methods and add the required codes to these methods.
3. Import the required classes.

```
import java.util.*;
import javax.microedition.pim.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
```

4. Create the contact screen class and name it "ContactScreen". Figure 1 depicts how it should look like.

```
public final class ContactScreen extends MainScreen {
        public ContactScreen () {
                super();
         }
}
```

5. In this screen, you can add new contact information, save them and you can also see the list of contacts by choosing the "Contact List" from the menu. Thus, it consists of the three instances of EditFields for entering a name, an email and a phone number. It also includes a menu with two items: "Add Contact" and "Contact List". You create an instance of these objects in the constructor. For this purpose, do the following steps.

6. Set Title to the "Add Contact".

```
setTitle(new LabelField("Add Contact", LabelField.ELLIPSIS |
LabelField.USE_ALL_WIDTH));
```

7. Create three instances of EditFields as follows and add them to the screen:

```
public final class ContactScreen extends MainScreen {
    private EditField _name, _email, _phone;

    public ContactScreen() {
       super();
        setTitle(new LabelField("Add Contact", LabelField.ELLIPSIS |
        LabelField.USE_ALL_WIDTH));
        _name = new EditField("Name: ", "");
        add(_name);
         _email = new EditField("Email Address: ",
         "",BasicEditField.DEFAULT_MAXCHARS, BasicEditField.FILTER_EMAIL);
        add(_email);
```

```
        _phone = new EditField("Work Phone: ",
        "",BasicEditField.DEFAULT_MAXCHARS, BasicEditField.FILTER_PHONE);
        add(_phone);
    }
```

8. The next step is to create the menu. It includes two menuItems. Thus, you should define one subclass for each of these items. Your subclasses must implement the Runnable interface. It is defined as follows:

```
    private class SaveMenuItem extends MenuItem implements Runnable {
        private SaveMenuItem() {
            super(null, 0, 100000, 5);
        }
        public String toString() {
            return "Save";
        }
        public void run() {
            onSave();
        }
    }
    private class ShowMenuItem extends MenuItem implements Runnable {
        private ShowMenuItem() {
            super(null, 0, 100000, 5);
        }
        public String toString() {
            return "Contact List";
        }
        public void run() {
            onList();
        }
    }
```

The first menu item saves the contact information in the handheld address book. When you click on this item, it invokes the onSave() method. In this method, you use PIM APIs to add the data to the address book. You will define it in the next steps.
The second menu item demonstrates the list of contacts. When you click on this menu item, it invokes the onList() method. In this method, you retrieve the contact information in a list. You will define this method in the next steps as well.

9. In the previous step, you defined the classes for the menu items. Now, you should create two instances of these classes and put them in the constructor.

```
public final class ContactScreen extends MainScreen {
        ……..
        private SaveMenuItem _saveMenuItem;
        private ShowMenuItem _showItems;
```

```
        public ContactScreen() {
                super();
                 …..
                 _saveMenuItem = new SaveMenuItem();
                  _showItems = new ShowMenuItem();
                 …….
        }
        ……
}
```

10. Now, you should add the menu items' instances to the menu. This is done using the makeMenu() method.

```
protected void makeMenu(Menu menu, int instance) {
        menu.add(_saveMenuItem);
        menu.add(_showItems);
        super.makeMenu(menu, instance);
 }
```

makeMenu() is a method defined in the Manager class (See the JDE API document). You can override this function with your own menu. To do this, you must call *super.makeMenu(menu, instance).* In this way, your menu items are added to the menu.

11. The next step is to write the onSave() fuction. This is done in the following steps:
    - Get the data from the EditField instances:

```
protected boolean onSave() {
        String name = _name.getText();
        String email = _email.getText();
        String phone = _phone.getText();
}
```

    - Verify that a name and email have been entered.

```
protected boolean onSave() {
        String name = _name.getText();
        String email = _email.getText();
        String phone = _phone.getText();

        if (name.equals("")  || email.equals("")) {
           Dialog.inform("You must enter a name and an email address!");
           return false;
        } else {
                ....
        }
}
```

- Save the contact information. For this purpose, do the following tasks:
  - Open a contact list: you must create a contact list before you can add contacts. To do this, invoke PIM.openPIMList(). As parameters, provide the type of list to open and the access mode (READ_WRITE, READ_ONLY, or WRITE_ONLY).

```
try{
        ContactList contactList =
                (ContactList)PIM.getInstance().openPIMList  (
                 PIM.CONTACT_LIST, PIM. READ_WRITE
                );
catch(Exception e)
{
        return false;
}
```

  - Create a contact: to do this invoke createcontact() on a contact list.

```
        .....
        Contact contact = contactList.createContact();
        ....
```

  - Add contact information: the Contact class defines fields in which you store data, such as Contact.NAME, Contact.ADDR, and Contact.TEL. Each field has a specific data type. Consider, for instance, name and address that are a String_Array or email and phone that are a string. Thus, in order to save the name, you should define an array of string:

```
String[] names =
         new  String [contactList.stringArraySize(Contact.NAME)];
```

    contractList.stringArraySize (given field) returns the size of a string array for a given field.

  - The following line assigns the name entered by the user to the given name of the contact's name array defined above:

```
if (!name.equals(""))
      names[Contact.NAME_GIVEN] = name;
```

  - The following lines add the data to the relative fields in the contact objects.

```
contact.addStringArray(Contact.NAME, Contact.ATTR_NONE, names);
contact.addString(Contact.EMAIL, Contact.ATTR_HOME, email);
contact.addString(Contact.TEL, Contact.ATTR_WORK, phone);
```

  - Save a contact: the data added to contact object is saved to the address book by invoking commit().

```
contact.commit();
```

- Reset UI fields and return true.

```
_name.setText("");
_email.setText("");
_phone.setText("");
return true;
```

12. The last step is to write the onList() fuction. This is done in the following steps:
    - Set the title of the screen to the "Contact List".

```
protected void onList() {
        String tmp;
        try{
           setTitle(new LabelField("Contact List", LabelField.ELLIPSIS |
                LabelField.USE_ALL_WIDTH));

        }catch(Exception e)
        {
        }
```

    - Delete the fields from the screen's manager.

```
           delete(_name);
           delete(_phone);
           delete(_email);
```

    - Create an instance of RichTextField and add it to the screen. For a better performance, you can create a list instead.

```
RichTextField rich1 = new RichTextField("\n");
add(rich1);
```

    - Retrieve contact information: for this purpose, do the following tasks:
        - Open a contact list: this time, set the access mode to Read_Only.

```
ContactList contactList =
        (ContactList)PIM.getInstance().openPIMList (
            PIM.CONTACT_LIST, PIM.READ_ONLY
            );
```

        - Invoke contactList.items() to enumerate the items in the address book.

```
Enumeration enum = contactList.items();
```

        - Enumerate the items. For a particular contact, invoke getFields() to retrieve an array of IDs for fields:

```
while (enum.hasMoreElements()) {
        Contact c = (Contact)enum.nextElement();
      int[] fieldIds = c.getFields();
      int id;
      for(int index = 0; index < fieldIds.length; ++index) {
            id = fieldIds[index];
             if(c.getPIMList().getFieldDataType(id) ==
                                        Contact.STRING) {
```

```
                for(int j=0; j < c.countValues(id); ++j) {
                    String value = c.getString(id, j);
                    tmp = rich1.getText()+"\n";
                    rich1.setText(tmp  + value );
                }
            }
        }
        tmp = rich1.getText()+"\n";
        rich1.setText(tmp );
```

c.countValues(id) returns a value indicating the number of values currently contained within the field.

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

Listing 1 – Lab10.java:

```
import java.util.*;
import javax.microedition.pim.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;

//import net.rim.blackberry.api.pdap.*;

public class Lab10 extends UiApplication{
   private ContactScreen _contactScreen;

   public Lab10() {
      _contactScreen  = new ContactScreen();
      pushScreen(_contactScreen );
   }

   public static void main(String[] args) {
      Lab10 theApp = new Lab10();
      theApp.enterEventDispatcher();
   }

   public final class ContactScreen extends MainScreen {
      private EditField _name, _email, _phone;
      private SaveMenuItem _saveMenuItem;
      private ShowMenuItem  _showItems;


      public ContactScreen() {
         super();
         _saveMenuItem = new SaveMenuItem();
         _showItems = new ShowMenuItem();
         setTitle(new LabelField("Add Contact", LabelField.ELLIPSIS |
                  LabelField.USE_ALL_WIDTH));
```

```
        _name = new EditField("Name: ", "");
        add(_name);
        _email = new EditField("Email Address: ",
                    "",BasicEditField.DEFAULT_MAXCHARS,
                    BasicEditField.FILTER_EMAIL);
        add(_email);

         _phone = new EditField("Work Phone: ",
                    "",BasicEditField.DEFAULT_MAXCHARS,
                    BasicEditField.FILTER_PHONE);
        add(_phone);

    }

    private class SaveMenuItem extends MenuItem {
        private SaveMenuItem() {
            super(null, 0, 100000, 5);
        }
        public String toString() {
            return "Save";
        }
        public void run() {
            onSave();
        }
    }
    private class ShowMenuItem extends MenuItem {
        private ShowMenuItem() {
            super(null, 0, 100000, 5);
        }
        public String toString() {
            return "Contact List";
        }
        public void run() {
            onList();
        }
    }

     protected void makeMenu(Menu menu, int instance) {
        menu.add(_saveMenuItem);
        menu.add(_showItems);
        super.makeMenu(menu, instance);
    }

    protected void onList() {
        String tmp;
        try{
            setTitle(new LabelField("Contact List", LabelField.ELLIPSIS |
                            LabelField.USE_ALL_WIDTH));
            delete(_name);
            delete(_phone);
            delete(_email);
```

```
        RichTextField rich1 = new RichTextField("\n");
        add(rich1);

        ContactList contactList =
                (ContactList)PIM.getInstance().openPIMList(PIM.CONTACT_LIST,
                PIM.READ_ONLY);

        Enumeration enum = contactList.items();
        while (enum.hasMoreElements())
        {
            Contact c = (Contact)enum.nextElement();
            int[] fieldIds = c.getFields();
            int id;
            for(int index = 0; index < fieldIds.length; ++index)
            {
                id = fieldIds[index];
                if(c.getPIMList().getFieldDataType(id) == Contact.STRING)
                {
                    for(int j=0; j < c.countValues(id); ++j)
                    {
                        String value = c.getString(id, j);
                        tmp = rich1.getText()+"\n";
                        rich1.setText(tmp  + value );
                    }
                }
            }
            tmp = rich1.getText()+"\n";
            rich1.setText(tmp );
        }
    }catch(Exception e)
    {
      System.out.println(e);
    }
}

protected boolean onSave() {
    String name = _name.getText();
    String email = _email.getText();
    String phone = _phone.getText();

    if (name.equals("")  || email.equals("")) {
        Dialog.inform("You must enter a name and an email address!");
        return false;
    } else {
        try {
            ContactList contactList =
                 (ContactList)PIM.getInstance().openPIMList(PIM.CONTACT_LIST,
                 PIM.READ_WRITE);
            Contact contact = contactList.createContact();
```

```
                    String[] names = new
                                        String[contactList.stringArraySize(Contact.NAME)];

                    if (!name.equals(""))
                          names[Contact.NAME_GIVEN] = name;

                    contact.addStringArray(Contact.NAME, Contact.ATTR_NONE, names);
                    contact.addString(Contact.EMAIL, Contact.ATTR_HOME, email);
                    contact.addString(Contact.TEL, Contact.ATTR_WORK, phone);

                    // Save data to address book.
                    contact.commit();
                    // Reset UI fields.
                     _name.setText("");
                     _email.setText("");
                     _phone.setText("");
                     return true;
                    } catch(Exception e)
                    {
                       System.out.println(e);
                       return false;
                    }
                }
            }
        }
    }
```

Excersize 1 Develop the experiment #1 using PDAP APIs.

net.rim.blackberry.api.pdap package enables applications to interact with BlackBerry PIM applications. As mentioned earlier, these APIs are controlled APIs. Most of the same functionality is provided by PIM APIs. However, you can use these pdap APIs because they provide additional methods. In the experiment #1, you can cast your contactlist as a BlackBerryContactList. (See other methods in JDE API documents)
Note: to use JDE API document, go to *Help* menu and select *API Reference.* Then go to the net.rim.blackberry.api.pdap package and see its interfaces. [**4 marks**]