POLYTECHNIC INSTITUTE OF LEIRIA
SCHOOL OF TECHNOLOGY AND MANAGEMENT
DEPARTMENT OF INFORMATICS ENGINEERING

ACTIVITIES AND INTENTS

The goal of this exercise sheet is to implement a contacts app. You will now learn some fundamentals of Android app design, including how to build a simple user interface and handle user input. Also, while implementing a Contacts app, you will learn what intents are about and how to start other activities. You will also experiment with setting up the action bar, using adapters and List Views, and using resources among other things.

Index

1) Define the main activity's initial layout

Create a new Android Studio project with the name `Contacts`. Set the Minimum SDK to the one suggested by the Create New Project wizard. Also, add an Empty Activity to your project with the name `MainActivity`.

Add new Image Assets. The Asset Studio can be accessed as shown in Figure 1.

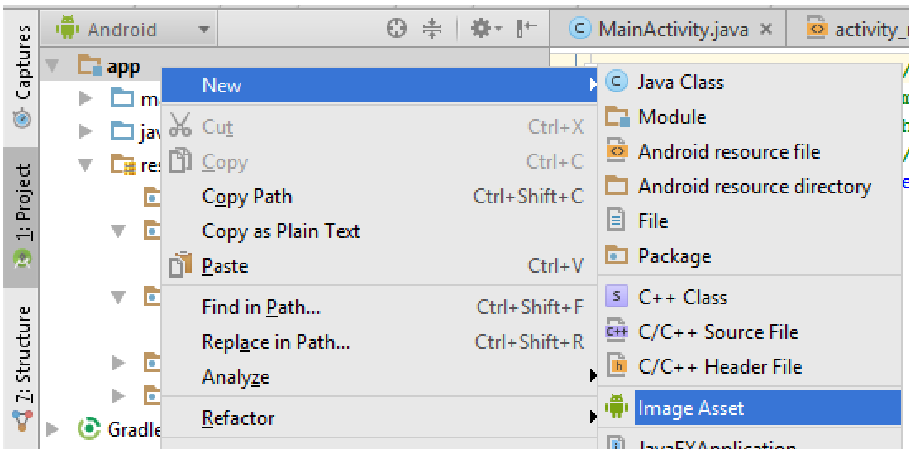
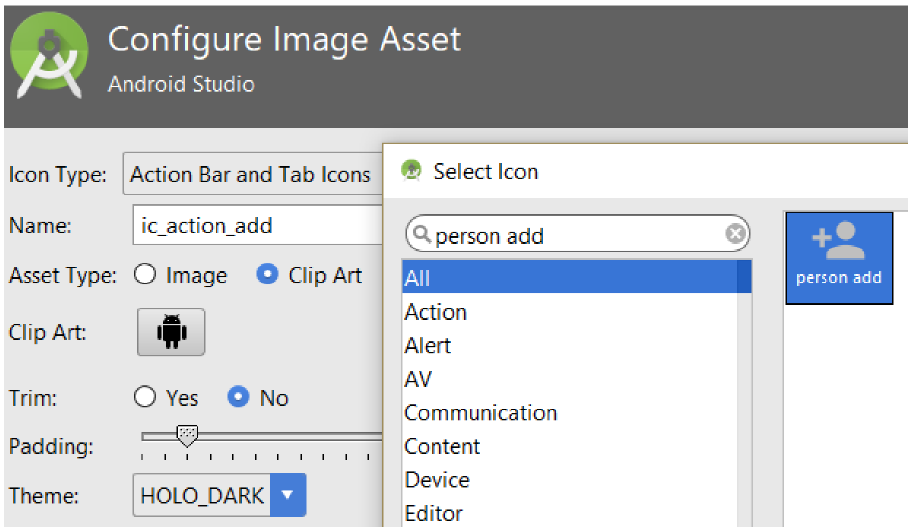


Figure 1

Locate the "person add" and "search" Action Bar and Tab Icons, as shown in Figure 2; name them as `ic_action_add` and `ic_action_search`, respectively. The new icons will be added to the "res/drawable" folder under your project tree.



Icon Type: Action Bar and Tab Icons
Name: ic_action_add
Asset Type: Image [✓ Clip Art]
Clip Art:
Trim: Yes [✓ No]
Padding:
Theme: HOLO_DARK

Figure 2

Add a new "Android Resource File" to your project (right-click on app -> Android -> New Resource File); name it `menu_main.xml`; and don't forget to define the "Resource type" as "menu". The file will be added to the "res/menu" folder.

Edit the `menu_main.xml` file in order to add the icons to the menu bar (Code Block 1). Note that you should extract any hardcoded strings (e.g., the ones set for the titles) to string resources.

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">

    <item android:id="@+id/action_add"
          android:title="Add"
          android:icon="@drawable/ic_action_add"
          android:onClick="onClick_AddItem"
          app:showAsAction="ifRoom" />

    <item android:id="@+id/action_search"
          android:title="Search"
          android:icon="@drawable/ic_action_search"
          android:onClick="onClick_SearchItem"
          app:showAsAction="ifRoom" />

</menu>
```

Code Block 1

Finally, edit `MainActivity`'s code in accordance to Code Block 2 to: override the `onCreateOptionsMenu`, and have it inflate the menu and add the items to the action bar; and create the "onClick" event handler methods for both the add and the search items.

```java
public class MainActivity extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
  }

  public void onClick_AddItem(MenuItem item) {
    Toast.makeText(this, "Add clicked", Toast.LENGTH_SHORT).show();
  }

  public void onClick_SearchItem(MenuItem item) {
    Toast.makeText(this, "Search clicked", Toast.LENGTH_SHORT).show();
  }
}
```

Code Block 2

More info: Action Bar, Defining The ActionBar (CodePath Android Cliffnotes)

2) Create the model's classes

Add a new package named `model` under the existing `pt.ipleiria.contacts` package. Then, create two new classes under the `model` package: `Contact` (Code Block 3) and `Agenda` (Code Block 4). Analyse these classes.

```java
public class Contact {
  private String name;
  private int phone;

  public Contact(String name, int phone) {
    this.name = name;
    this.phone = phone;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public int getPhone() {
    return phone;
  }

  public void setPhone(int phone) {
    this.phone = phone;
  }

  @Override
  public String toString() {
    return "Name: " + name + "; Phone: " + phone;
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    Contact contact = (Contact) o;

    if (phone != contact.phone) return false;
    return name.equals(contact.name);
  }

  @Override
  public int hashCode() {
    int result = name.hashCode();
    result = 31 * result + phone;
    return result;
  }
}
```

Code Block 3

```java
public class Agenda {
  private ArrayList<Contact> contacts;

  public Agenda() {
    this.contacts = new ArrayList<>();
  }

  public void addContact(Contact contact) {
    if (containsPhone(contact.getPhone())) {
      throw new RuntimeException("Cannot add Contact to Agenda; phone
number already exists.");
    }

    contacts.add(contact);
  }

  public boolean containsPhone(int phone) {
    for (Contact c : contacts) {
      if (c.getPhone() == phone) {
        return true;
      }
    }

    return false;
  }

  public ArrayList<Contact> searchContactByPhone(int phone) {
    ArrayList<Contact> res = new ArrayList<>();

    for (Contact c : contacts) {
      if (c.getPhone() == phone) {
        res.add(c);
      }
    }

    return res;
  }

  public ArrayList<Contact> searchContactByName(String name) {
    ArrayList<Contact> res = new ArrayList<>();

    for (Contact c : contacts) {
      if (c.getName().contains(name)) {
        res.add(c);
      }
    }

    return res;
  }

  public ArrayList<Contact> getContacts() {
    return contacts;
  }

  @Override
  public String toString() {
    String res = "";

    for (Contact c : contacts) {
      res += c + "\n";
    }

    return res;
  }
}
```

Code Block 4

Open the `activity_main.xml` file, and make some adaptations to the activity's layout: define the `TextView`'s id as being `textView_contacts`; and have the `TextView` match the parent container's constraints (`layout_width` and `layout_height` properties).

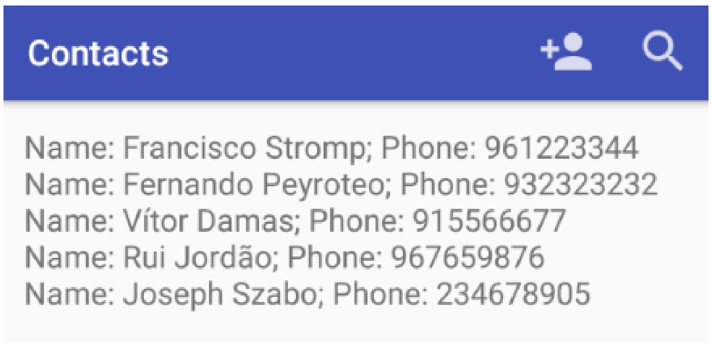
Now go back to the `MainActivity` class file and modify its code in order to: declare an `Agenda` field; instantiate an `Agenda` object inside the `onCreate()` callback method; create some dummy contact data; and show the contact data in the app's `TextView`.

The result of these changes depicted in Code Block 5.

```java
public class MainActivity extends AppCompatActivity {

  private Agenda agenda;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    this.agenda = new Agenda();

    // dummy data
    Contact c1 = new Contact("Francisco Stromp", 961223344);
    Contact c2 = new Contact("Fernando Peyroteo", 932323232);
    Contact c3 = new Contact("Vítor Damas", 915566677);
    Contact c4 = new Contact("Rui Jordão", 967659876);
    Contact c5 = new Contact("Joseph Szabo", 234678905);
    this.agenda.addContact(c1);
    this.agenda.addContact(c2);
    this.agenda.addContact(c3);
    this.agenda.addContact(c4);
    this.agenda.addContact(c5);

    TextView textView_contacts = findViewById(R.id.textView_contacts);
    textView_contacts.setText(this.agenda.toString());
  }

  (…)
}
```

Code Block 5

Run the app. The result should be similar to the one shown in Figure 3.



```
Name: Francisco Stromp; Phone: 961223344
Name: Fernando Peyroteo; Phone: 932323232
Name: Vítor Damas; Phone: 915566677
Name: Rui Jordão; Phone: 967659876
Name: Joseph Szabo; Phone: 234678905
```

Figure 3

3) List Views and Adapters

In your `activity_main.xml` file: replace your `MainActivity`'s `TextView` with a component of type `ListView`; and define its `id` property as `listView_contacts` (Code Block 6).

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="pt.ipleiria.contacts.MainActivity">

    <ListView
        android:id="@+id/listView_contacts"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</LinearLayout>
```

Code Block 6

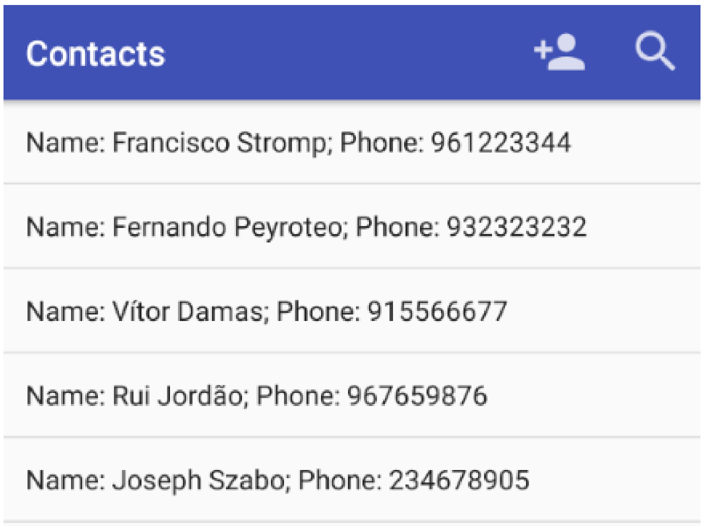
Go back to the `MainActivity`, and edit its `onCreate` method as depicted in Code Block 7. Add some more fake `Contact`s, and add them to the `Agenda`. Then create an `Adapter` – which is an object that retrieves data from an external source and creates a View that represents each data entry.

Specifically, an `ArrayAdapter` should be used because the data source is an `ArrayList` (containing the contacts). Finally, retrieve a reference to the `ListView` using its `id`, and set the adapter you created as the one maintaining the data behind the `ListView` using the `setAdapter` method.

```
public class MainActivity extends AppCompatActivity {

  private Agenda agenda;
  private ArrayAdapter<Contact> adapter;
  private ListView listView;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    this.agenda = new Agenda();

    // dummy data
    Contact c1 = new Contact("Francisco Stromp", 961223344);
    Contact c2 = new Contact("Fernando Peyroteo", 932323232);
    Contact c3 = new Contact("Vítor Damas", 915566677);
    Contact c4 = new Contact("Rui Jordão", 967659876);
    Contact c5 = new Contact("Joseph Szabo", 234678905);
    this.agenda.addContact(c1);
    this.agenda.addContact(c2);
    this.agenda.addContact(c3);
    this.agenda.addContact(c4);
    this.agenda.addContact(c5);

    adapter = new ArrayAdapter<>(
        this,
        android.R.layout.simple_list_item_1,
        agenda.getContacts());
    listView = findViewById(R.id.listView_contacts);
    listView.setAdapter(adapter);
  }

  (…)
}
```

Code Block 7

Run your app. It should look something like the one depicted in Figure 4.



Name: Francisco Stromp; Phone: 961223344

Name: Fernando Peyroteo; Phone: 932323232

Name: Vítor Damas; Phone: 915566677

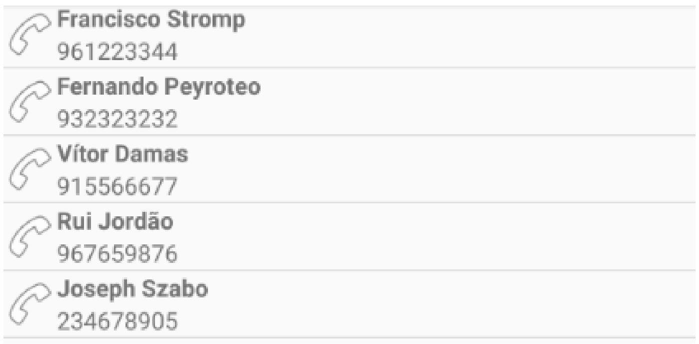
Name: Rui Jordão; Phone: 967659876

Name: Joseph Szabo; Phone: 234678905

Figure 4

More info: Building Layouts with an Adapter, List Views

You can personalize your `ListView`'s items by creating a custom adapter. Let's modify your Contact's app in order to make it look like the one in Figure 5.



Francisco Stromp
961223344

Fernando Peyroteo
932323232

Vítor Damas
915566677

Rui Jordão
967659876

Joseph Szabo
234678905

Figure 5

First you have to define the layout of the `ListView`'s items. Create a new "layout resource file" named `item_contact.xml`, and use the layout available in Code Block 8.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        app:srcCompat="@android:drawable/ic_menu_call"/>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <TextView
            android:id="@+id/textView_name"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Name"
            android:textStyle="bold"/>

        <TextView
            android:id="@+id/textView_phone"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Phone"/>
    </LinearLayout>

</LinearLayout>
```

Code Block 8

Next, you'll have to subclass `ArrayAdapter` and override the `getView` method; this will describe the translation between the data item (i.e., the `Contact`) and the `View` to display (i.e., the `ListView`'s item). Create the `MyAdapter` class and define it as in Code Block 9.

```
public class MyAdapter extends ArrayAdapter<Contact> {
  private final Context context;
  private ArrayList<Contact> contacts;

  public MyAdapter(Context context, ArrayList<Contact> contacts) {
    super(context, R.layout.item_contact, contacts);

    this.context = context;
    this.contacts = contacts;
  }

  @NonNull
  @Override
  public View getView(int position, @Nullable View convertView,
                      @NonNull ViewGroup parent) {

    if (convertView == null) {
      convertView = LayoutInflater.from(context)
          .inflate(R.layout.item_contact, parent, false);
    }

    Contact contact = contacts.get(position);

    TextView name = convertView.findViewById(R.id.textView_name);
```

```
        name.setText(contact.getName());

        TextView phone = convertView.findViewById(R.id.textView_phone);
        phone.setText("" + contact.getPhone());

        return convertView;
    }
}
```

Code Block 9

Finally, you'll have to instantiate `MyAdapter` and attach it to your `ListView`. In order to do so, modify your `MainActivity` using the code snippet available in Code Block 10.

```
listView = findViewById(R.id.listView_contacts);
adapter = new MyAdapter(this, agenda.getContacts());
listView.setAdapter(adapter);
```

Code Block 10

More info: Using a Custom ArrayAdapter (codepath.com)

It should be noted that you should employ the "View Holder design pattern" in order to avoid unconditionally inflating a new layout; this will help make `ListView` scrolling much smoother.

More info: Hold View Objects in a View Holder, Improving Performance with the ViewHolder Pattern (codepath.com)

Also, if your app needs to display a scrolling list of elements based on large data sets (or data that frequently changes), you should consider using `RecyclerView` instead of a `ListView`.

More info: Create a List with RecyclerView

4) Create the Search Contacts and Add Contact activities

Now let's create two new activities: the Add Contact activity and the Search Contact activity. You can do this by pressing the right mouse button on app, and then selecting "New->Activity->Empty Activity".

Name the former `AddActivity` and the latter `SearchActivity`. Define the layout for each of these activities; design them so that they look similar to the ones depicted in Figure 6 and Figure 7.

The ids of the `EditText`, `Button`, and `ListView` components of each activity should be defined as follows:

- *`AddActivity`*: `editText_addName`, `editText_addPhone`, and `button_add`.
- *`SearchActivity`*: `editText_search`, `button_search`, and `listView_search`.

Also, define the `onClick` property of each new button as follows:

- Add button: `button_add_onClick`
- Search button: `button_search_onClick`

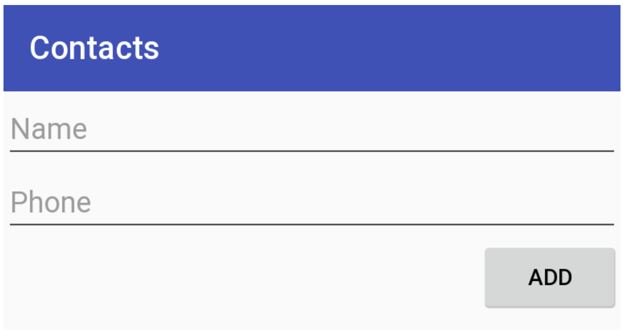


Figure 6

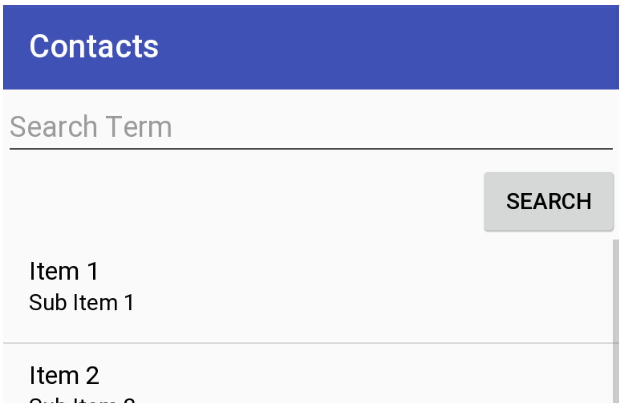


Figure 7

5) Implement the Search Contacts functionality

In order to implement the search contacts functionality, the app must launch the `SearchActivity` when the `action_search` menu item is pressed, and populate the `SearchActivitiy's ListView` with all the contacts containing the string entered in the `EditText` when the `button_search` is pressed.

We must also send all the contact information (i.e., the `Agenda` object) to the `SearchActivity`, so that it can search the contact's names. In order to do this, both the `Agenda` and the `Contact` classes must be *Serializable.*

Serialization is the process of translating object state into a format that can be stored (e.g., in a file, or for being transmitted across a network connection, or for being passed among Android activities) and reconstructed (i.e., *deserialized*) later in the same or another context. Java provides automatic serialization which requires that the object be marked by implementing the `java.io.Serializable` interface. Implementing the interface marks the class as "okay to serialize", and Java then handles serialization internally.

Edit the `Contact` and `Agenda` classes in the `model` package; change both their class declarations in accordance to Code Blocks Code Block 11 and Code Block 12, respectively.

```
import java.io.Serializable;

public class Contact implements Serializable {
(...)
```

Code Block 11

```
import java.io.Serializable;

public class Agenda implements Serializable {
(...)
```

Code Block 12

Proceed to edit the `onClick_SearchItem` method of the `MainActivity` class in order to handle the pressing of the search menu item (see Code Block 13).

When the search menu item is pressed a new `Intent` is created. An `Intent` is an object that provides runtime binding between separate components (such as two activities). The `Intent` represents an app's "intent to do something." You can use intents for a wide variety of tasks, but most often they're used to start another activity.

The constructor used here takes two parameters: a `Context` (`this` is used because the `Activity` class is a subclass of `Context`), and the class to which the system should deliver the `Intent` (in this case, the activity that should be started).

An `Intent` can carry data types as key-value pairs called "extras". The `putExtra()` method takes the key name in the first parameter and the value in the second parameter. For the next activity to query the extra data, you should define the key for your intent's extra using a public constant. It's generally a good practice to define keys for intent extras using your app's package name as a prefix. This ensures the keys are unique, in case your app interacts with other apps. The PT_IPLEIRIA_CONTACTS_AGENDA public constant should be defined in the MainActivity class as shown in Code Block 14.

```
public void onClick_SearchItem(MenuItem item) {
  Intent i = new Intent(this, SearchActivity.class);
  i.putExtra(EXTRA_AGENDA, agenda);
  startActivity(i);
}
```

Code Block 13

```
public static final String EXTRA_AGENDA = "pt.ipleiria.contacts.AGENDA";
```

Code Block 14

Let’s now implement the required code in the SearchActivity class. Start by defining two fields o type Agenda and Listview. Proceed to edit the onCreate method of the SearchActivity class. Make it so that the Intent that started the activity is retrieved and – most importantly – so that the extra included in the Intent – the Agenda – is also stored as a field for use in this class.

```
public class SearchActivity extends AppCompatActivity {

  private Agenda agenda;
  private ListView listView_search;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_search);

    Intent i = getIntent();
    this.agenda =
        (Agenda) i.getSerializableExtra(MainActivity.EXTRA_AGENDA);
    this.listView_search = findViewById(R.id.listView_search);
  }
}
```

Code Block 15

Now define the SearchActivity button’s callback method so that, when clicked, the text entered in the EditText is searched in the contacts’ names using the Agenda’s searchContactByName method. Also, update the ListView with the retrieved contacts, and show and informative message is no contact containing the entered string is found.

```
public void button_search_onClick(View view) {
  EditText editText_search = findViewById(R.id.editText_search);
  String nameToSearch = editText_search.getText().toString();

  ArrayList<Contact> searchedContacts =
      agenda.searchContactByName(nameToSearch);

  ArrayAdapter<Contact> adapter = new ArrayAdapter<Contact>(
      this,
      android.R.layout.simple_list_item_1,
      searchedContacts);
  listView_search.setAdapter(adapter);
```

```
    if (searchedContacts.isEmpty()) {
      Toast.makeText(this, "No Contacts found...",Toast.LENGTH_LONG).show();
    }
  }
```

Code Block 16

Now run your app and search away! Unfortunately the search is not case insensitive… Can you fix that?

More info: Starting Another Activity

6) Implement the Add Contacts functionality

In order to implement the app's this functionality, the AddContact activity you implemented must return the created contact to the MainActivity, so that it can be added to the MainActivity's agenda. This can be achieved if:

- the MainActivity starts the AddActivity with the explicit purpose of receiving a result back;
- the AddActivity is designed to return a result (i.e., the new contact); and
- the MainActivity is able to handle the returned contact, by adding it to the agenda and refreshing the displayed ListView.

In order to achieve this, you will use the startActivityForResult() method (instead of the startActivity method you previously utilized to launch the SearchActivity).

Let's start by handling the users click in the MainActivity's add MenuItem. In this activity, define the onClick_AddItem method (Code Block 17).

```
public class MainActivity extends AppCompatActivity {

  private static final int REQUEST_CODE_ADD = 1;

   (…)

  public void onClick_AddItem(MenuItem item) {
    Intent i = new Intent(this, AddActivity.class);
    startActivityForResult(i, REQUEST_CODE_ADD);
  }
}
```

Code Block 17

Now, if everything goes well, the AddActivity will be launched when you click the action_add MenuItem. Notice that this start operation is identified with a request code (the number 1, defined in the REQUEST_CODE_ADD) which will be useful later on.

Let's now proceed to edit the AddActivity. Modify its onCreate callback method so that is looks like the one depicted in Code Block 18.

```
public class AddActivity extends AppCompatActivity {

  public static final String EXTRA_CONTACT =
"pt.ipleiria.contacts.EXTRA_CONTACT";

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_add);

    Button button_add = findViewById(R.id.button_add);

    button_add.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
          EditText editText_addPhone =
                                      findViewById(R.id.editText_addPhone);
          String phoneAsString = editText_addPhone.getText().toString();
```

```
            int phone = Integer.parseInt(phoneAsString);

            EditText editText_addName = findViewById(R.id.editText_addName);
            String name = editText_addName.getText().toString();

            Contact newContact = new Contact(name, phoneAsString);

            Intent returnIntent = new Intent();
            returnIntent.putExtra(EXTRA_CONTACT, newContact);
            setResult(RESULT_OK, returnIntent);
            finish();
        }
      });

    }
  }
```

Code Block 18

The `AddActivity` now handles the click on the button by: getting the contact's name and id from the `EditText`s; creating a new `Contact`; creating a new `Intent`; adding the new contact to the `Intent` as an extra; setting the result that this activity will return to its caller signalling that everything went ok; and calling the `finish()` method to explicitly close this activity and send the result (and the contact) back to the `MainActivity`.

Now let's get back to the main activity and add the new contact to the agenda. Override `onActivityResult()` as shown in Code Block 19.

```
@Override
protected void onActivityResult(int requestCode, int resultCode,
   Intent data) {
  super.onActivityResult(requestCode, resultCode, data);

  if (requestCode == REQUEST_CODE_ADD) {
    if (resultCode == RESULT_OK) {
      Contact contact = (Contact)
         data.getSerializableExtra(AddActivity.EXTRA_CONTACT);

      try {
        agenda.addContact(contact);

        adapter.notifyDataSetChanged();
      } catch (RuntimeException e) {
        e.printStackTrace();
        new AlertDialog.Builder(this).setMessage(e.getMessage()).show();
      }
    }
  }
}
```

Code Block 19

The `onActivityResult` method is automatically called when an activity started with `startActivityForResult` finished, and receives: the `requestCode` originally supplied to identify who this result came from; the `resultCode`, returned by the previous activity through its `setResult()` method; and an intent containing all included extras – in this case the new `Contact`.

If the `requestCode` and the `resultCode` are as expected, the contact is obtained from the intent, added to the `agenda` field of the `MainActivity`, and the `ArrayAdapter` is notified that its data source has changed and thus it should inform the associated `ListView` to refresh. Note that in order for this code to compile, the `ArrayAdapter` must be declared as a class field.

Now you can run the app and try to add your own (smart, as opposed to dummy) contacts! What happens to the contacts you added when you change the orientation of the screen (e.g., from portait to landscape)?

More info: Starting Activities and Getting Results, Getting a Result from an Activity

7) Exercises

a) Implement an Edit Contact functionality.

If the user clicks on a contact in the `MainActivity`'s `ListView`, the `AddActivity` is launched and its layout is filled with the clicked contact's information; also, `AddActivity`'s button should show the text "Edit". Then, if the user clicks on the "Edit" button the clicked contact's information should be updated and the user should be redirected back to the `MainActivity`; and if the user clicks the back button no changes should be made to the contact.

Tip – code for showing a contact's name when a `ListView` item is clicked: http://pastebin.com/Z33Rm7z6

b) Implement a Delete Contact functionality.

If the user performs a long click on a contact in the `MainActivity`'s `ListView`, a dialog is shown asking if the user really wants to delete the contact. If the user confirms, the contact is deleted from the agenda; otherwise no changes are made.