Lab 4: Create a multi-threaded application

Goals

- Create a multi-threaded application
- Learn how-to create threads
- Have the threads send a message to the main thread for UI operation

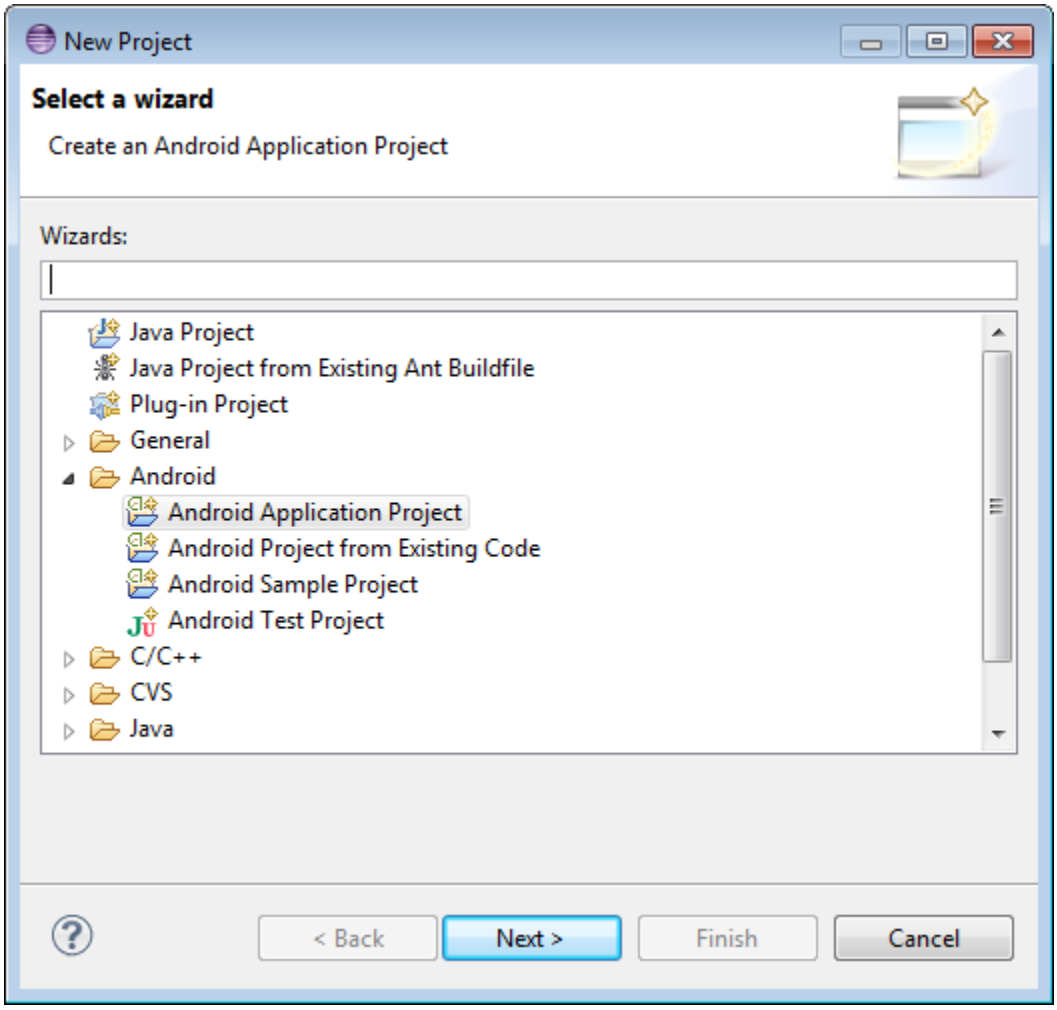
Estimated time: 35 minutes

Part 1: Create a new android application project

- **Creating the project**

1. Open the Eclipse IDE

2. Select **File -> New -> New Project.** Select Android Application Project and click next.

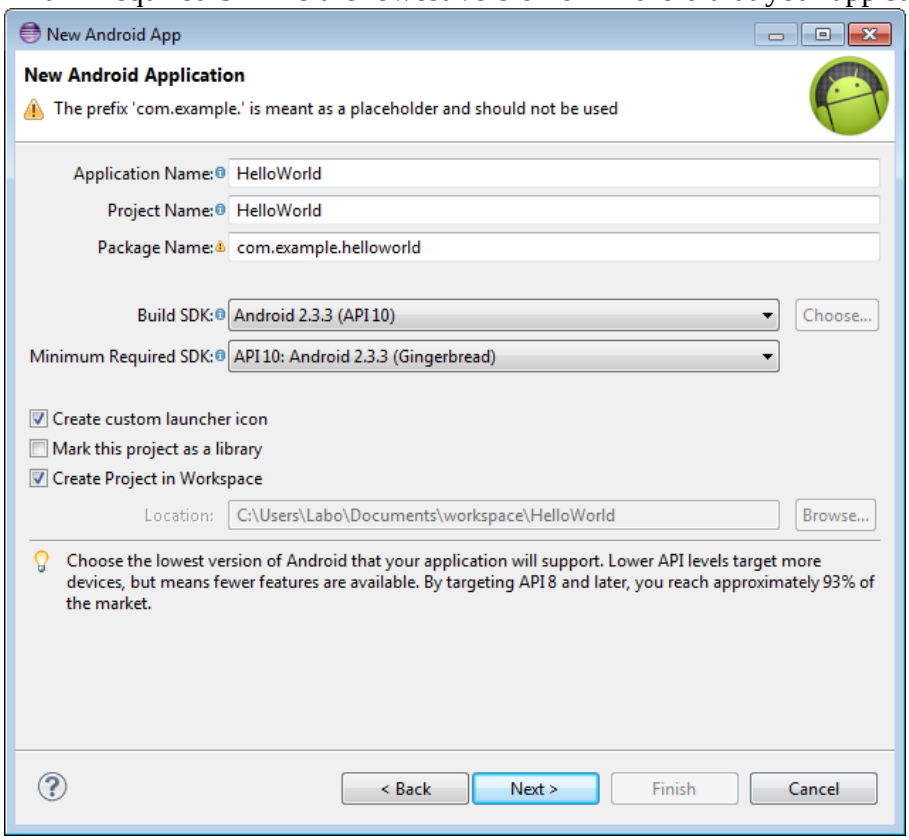


3.

4. Fill in the form in the window that appeared like on the following screenshot and click **Next**.

    - Application name is the name that appears to users (application icon).
    - Project name corresponds to the name of your project directory (visible in Eclipse)
    - Package name corresponds to the namespace for you application
    - Build SDK is the platform version against which you will compile your app.

Minimum Required SDK is the lowest version of Android that your app supports.



5. Now you can select an activity template from which to begin building your app.

    For this project, select BlankActivity and click **Next**.

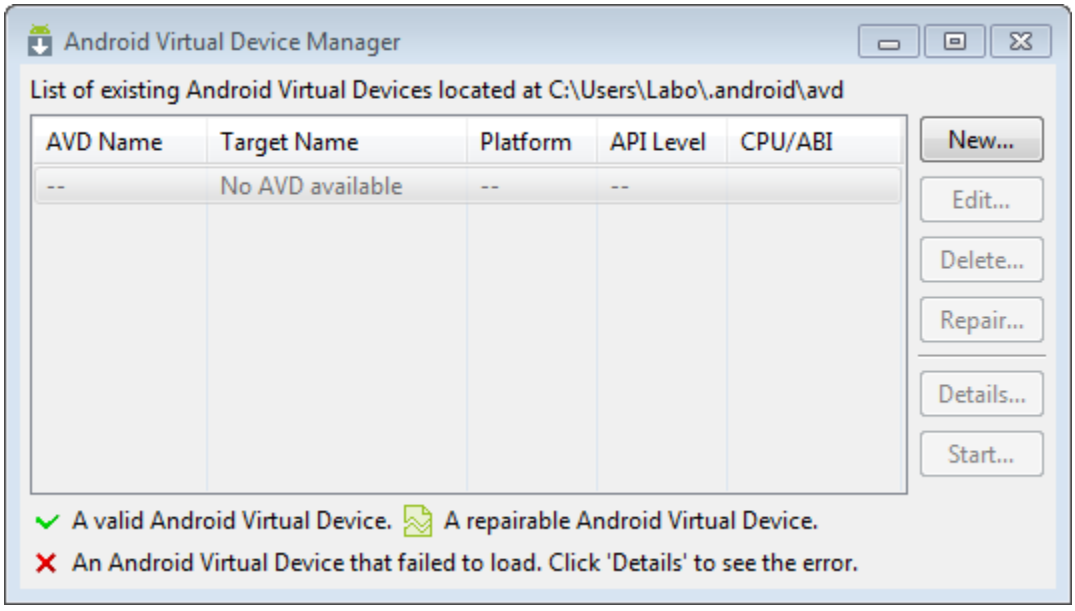
6. Leave all the details for the activity in their default state and click **Finish**.

    You should end un with the following view :

Part 2: Creating an Android Virtual Device (AVD)

- **Create the AVD**

1. Launch the Android Virtual Device Manager via the Eclipse menu toolbar **Window | AVD Manager**.



2. In the Android Virtual Device Manager panel, click **New**.

3. Fill in the details for the AVD. Give it a name, a platform target, an SD card size, and a skin (HVGA is default). Click **Create AVD**.

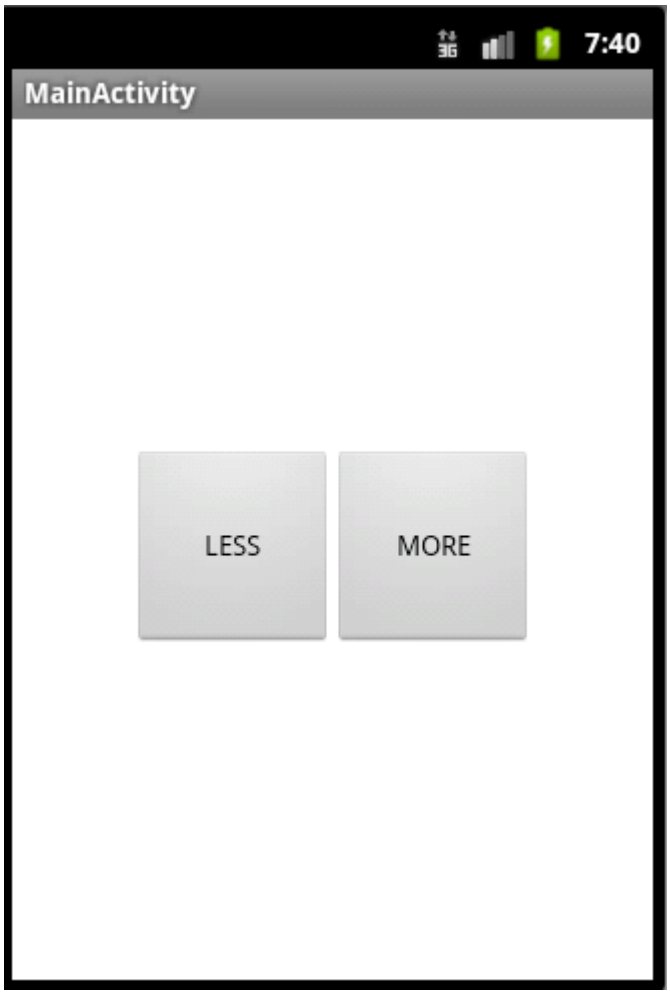
4. Select the new AVD from the Android Virtual Device Manager and click **Start**.

5. After the emulator boots up, unlock the emulator screen.

6. To run the app from Eclipse, open one of your project's files and click Run from the toolbar. Eclipse installs the app on your AVD and starts it.

Part 3: Create the application

The final application should look like this.



It contains:

- Two buttons, to create a new thread or delete the last created thread

Goal:

- Add and remove threads easily, explore them in DDMS

- Create graphics elements

Open the xml file present in folder **res** -> **layout** . You should see something like this:

```
<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"
    android:orientation="vertical"

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/hello_world"
        tools:context=".MainActivity" />

</LinearLayout>
```

First, delete the **TextView** element and add a **Button** component.

```
<Button
            android:id="@+id/btnLess"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:text="LESS"
            android:onClick="btnLessClicked" />
```

id property is used to make a link between GUI and variables in java classes.
layout_width and **layout_height** corresponds of the component's size.
Value **match_parent** allow to the component to use the maximum space according to the size of its parent, here the linear layout.
Value **wrap_content** allow to the component to use only its content to maximum size.
Text property corresponds to the text inside the button.
onClick property allow to detect a click occurred on the button and catch the event inside a method, here, **btnLessClicked.**

Do the same thing for adding a **"**More" button.

Notes: XML language functioning as a tree you can for example add a **LinearLayout** with a horizontal orientation inside another **LinearLayout** with a vertical orientation.

- Create the main activity

Open the main activity of your project present in folder
src-> <package_name> -> MainActivity.java

- Variable declaration

We need to add some variables: an array to store new threads information, a long to store threads ids. Add member variables as private with the same type as XML's components

```
public class MainActivity extends Activity {

	private static ArrayList<Thread> threadArray;
	private static long currentThreadId;
	private static final int MSG_NEW_THREAD = 1;
	private static final int MSG_DELETE_THREAD = 2;
	private final int MAX_THREAD_NUMBER = 10;
	private static Context ctx;
```

Initialize these variables in the **OnCreate** function.

```
threadArray = new ArrayList<Thread>();
ctx = getBaseContext();
```

- Creation of the Class implementing Runnable interface

A thread gets as argument an object implementing the **Runnable** interface

```
private static class RunnableThread implements Runnable {

	public void run() {
		try {
		   //Send message to the handler to change the GUI
		   Message msg = Message.obtain();
		   msg.arg1 = MSG_NEW_THREAD;
		   handler.sendMessageDelayed(msg, MSG_NEW_THREAD);

		   //Sleep thread
		   threadArray.get(threadArray.size()-1).sleep(20000);
		 }
		 catch (Exception e) {
		   e.printStackTrace();
		}
	 }
 }
```

- Create methods allowing to add a thread

    - Instantiate a new thread
    - Add the newly created thread to the array
    - Start the new thread
    - Keep the thread id to delete it later

```
//Add a new thread
public void addNewThread() {
   if (threadArray.size() < MAX_THREAD_NUMBER) {
          //prepare the new thread
          Thread thread = new Thread(new RunnableThread());
          //add the thread to array
          threadArray.add(thread);
          //start the thread
          thread.start();
          //get its id
          currentThreadId = thread.getId();
   }
 }
```

- Create methods allowing to delete a thread

    - Interrupt the last thread of the array
    - Get the last thread id
    - Remove this thread from the array
    - Call the handler

```
 public void deleteThread() {

   int size = threadArray.size();
   if (size > 0) {
          //interrupt the last thread
          threadArray.get(size-1).interrupt();
          //get its id
          currentThreadId = threadArray.get(size-1).getId();
          //remove the thread from the array
          threadArray.remove(size-1);

          //send message to handler to modify the GUI
          Message msg = Message.obtain();
          msg.arg1 = MSG_DELETE_THREAD;
          handler.sendMessageDelayed(msg, MSG_DELETE_THREAD);
   }
}
```

- Create the handler

Because only the main thread is able to modify the GUI, we need to use a handler in order to show a **Toast** message when a thread is created or deleted.

```
//handler creation to be able to modify the GUI
private static Handler handler = new Handler() {

@Override
public void handleMessage(Message msg) {
    switch (msg.arg1) {
        case (MSG_NEW_THREAD): {
            Toast.makeText(ctx, "a new thread has been created id : "+
            currentThreadId , Toast.LENGTH_SHORT).show();
             break;
        }
        case (MSG_DELETE_THREAD): {
            Toast.makeText(ctx, "the last thread added number "+
            currentThreadId + " has been deleted ",
            Toast.LENGTH_SHORT).show();
             break;
        }
    }
 }
};
```

- Create methods to intercept click on buttons

As mentioned in the XML we need to create methods to intercept click on buttons LESS and MORE. These methods will just call the two methods a**ddNewThread()** and **deleteThread().**

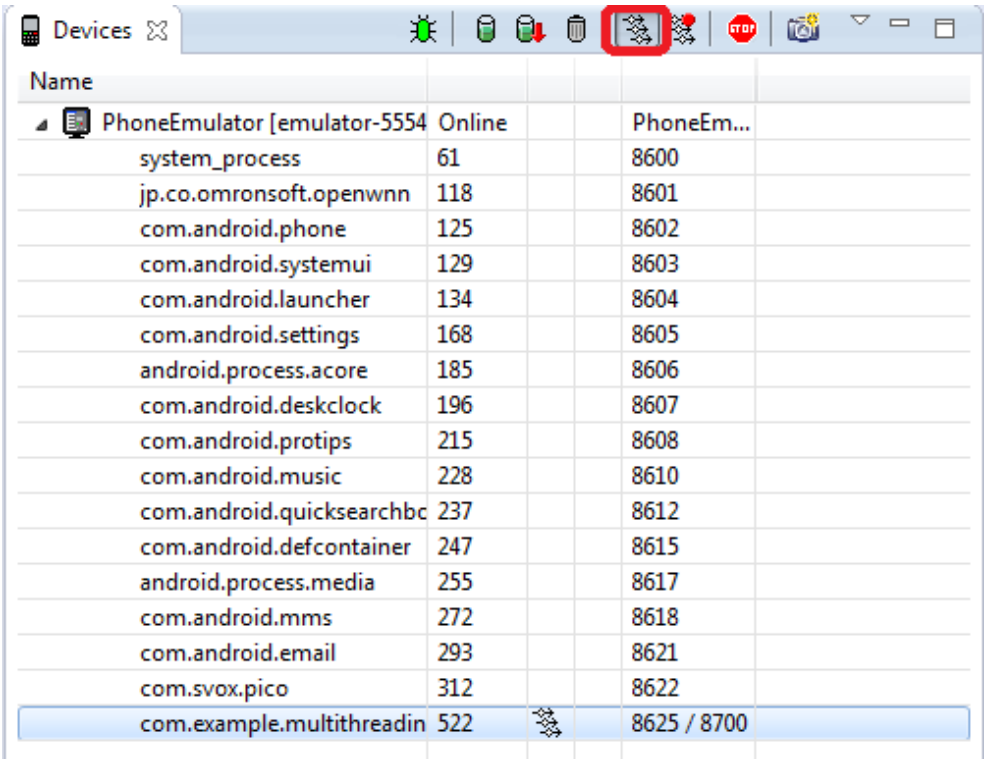
```
//Detect click occurred on the LESS button
public void btnLessClicked(View v) {
      Log.v("USER","btn less pressed by user");
      deleteThread();
}

//Detect click occurred on the MORE button
public void btnMoreClicked(View v) {
      Log.v("USER","btn more pressed by user");
      addNewThread();
}
```

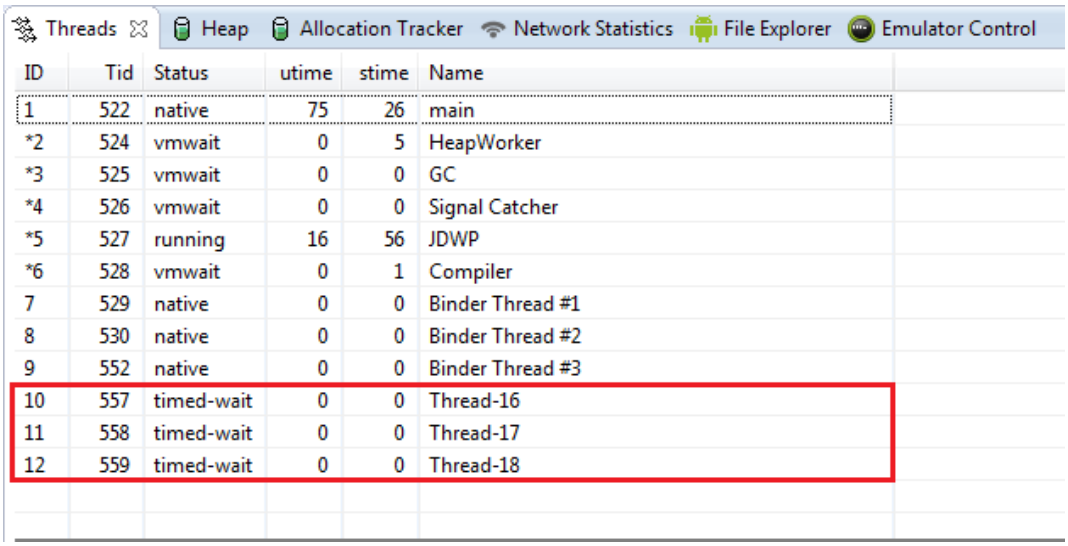
- Observe thread creation/deletion using DDMS

Open the DDMS window in Eclipse by cling **Window-> Open perspective -> Other** then double-click **DDMS**.

Run the application and look it up in the DDMS **Devices** window. Select the corresponding line in the **Devices** window, then click the **Update Threads** button:



Open the **Threads** tab in the DDMS window. Click several times on the "More" button from the application, and observe the newly created threads in the DDMS window.



Lab 5: Create a Twitter-like application

Goals

- Use Http requests
- Use JSON WebServices
- Learn about AsyncTask

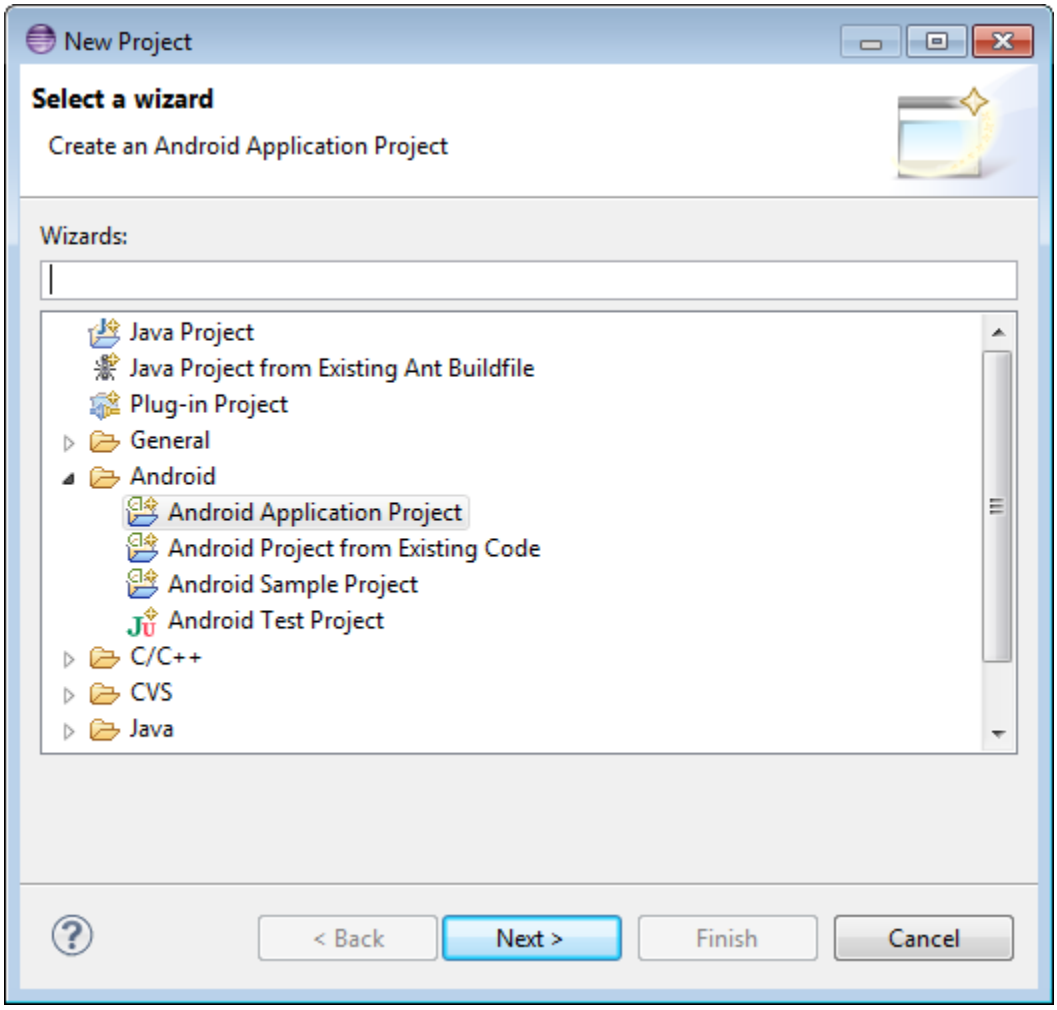
Estimated time: 75 minutes

Part 1: Create a new android application project

- **Creating the project**

1. Open the Eclipse IDE

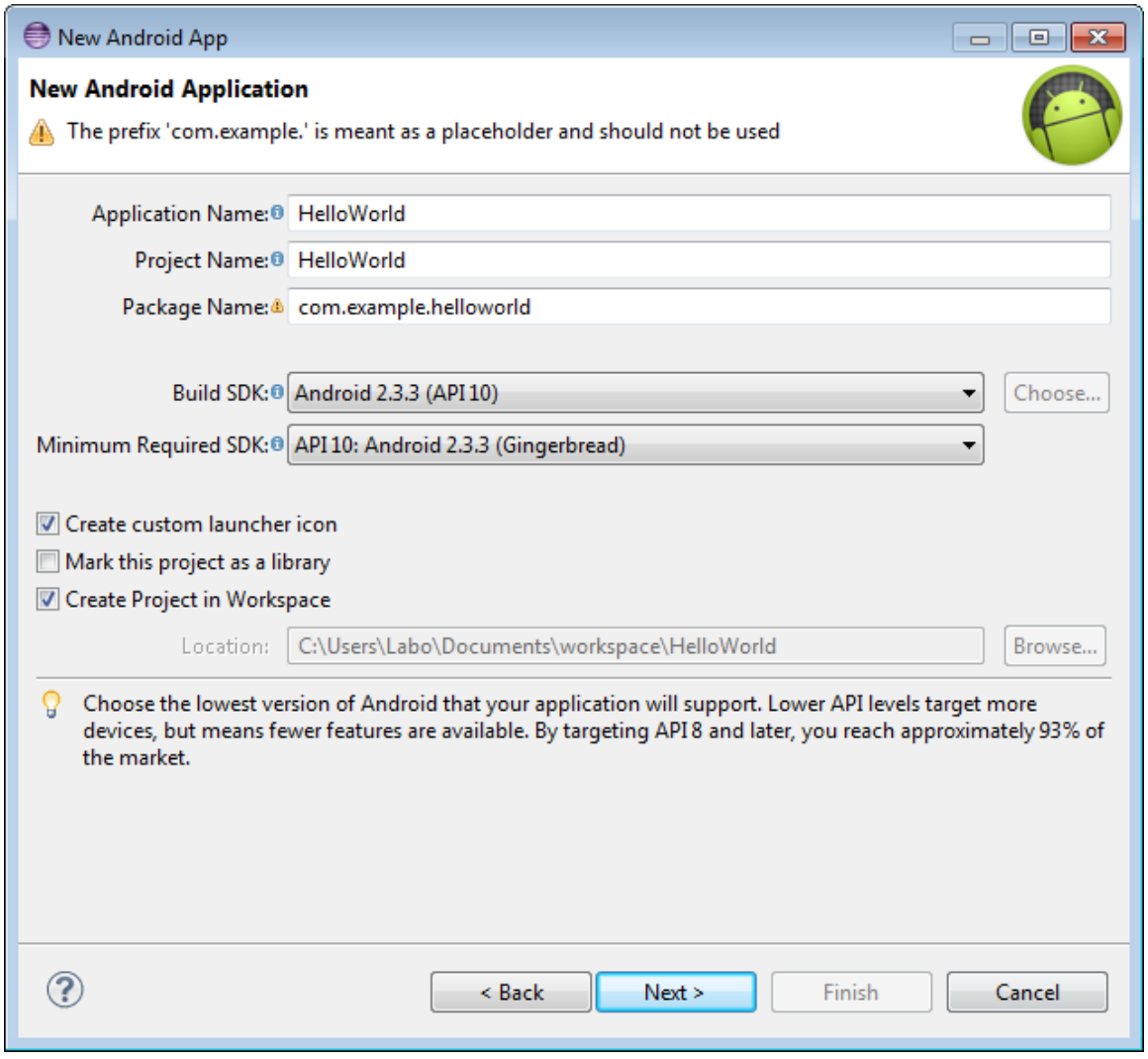
2. Select **File -> New -> New Project.** Select Android Application Project and click next.



3.

4. Fill in the form in the window that appeared like on the following screenshot and click **Next**.

    - Application name is the name that appears to users (application icon).
    - Project name corresponds to the name of your project directory (visible in Eclipse)
    - Package name corresponds to the namespace for you application
    - Build SDK is the platform version against which you will compile your app.
    - Minimum Required SDK is the lowest version of Android that your app supports.



5. Now you can select an activity template from which to begin building your app.

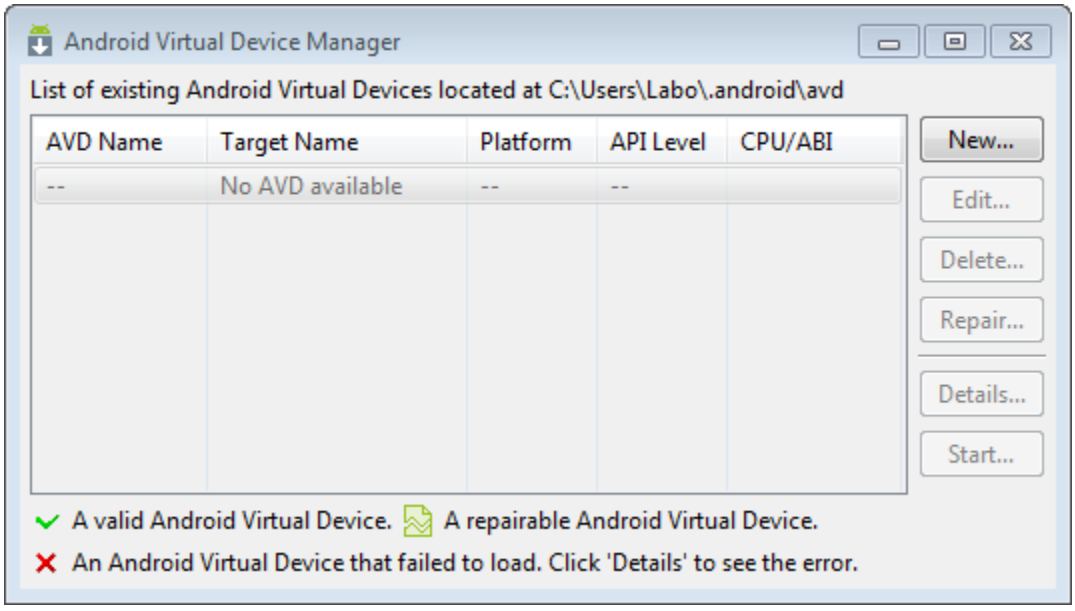
   For this project, select BlankActivity and click **Next**.

6. Leave all the details for the activity in their default state and click **Finish**.

Part 2: Creating an Android Virtual Device (AVD)

- **Create the AVD**

1. Launch the Android Virtual Device Manager via the Eclipse menu toolbar **Window | AVD Manager**.



2. In the Android Virtual Device Manager panel, click **New**.

3. Fill in the details for the AVD. Give it a name, a platform target, an SD card size, and a skin (HVGA is default). Click **Create AVD**.

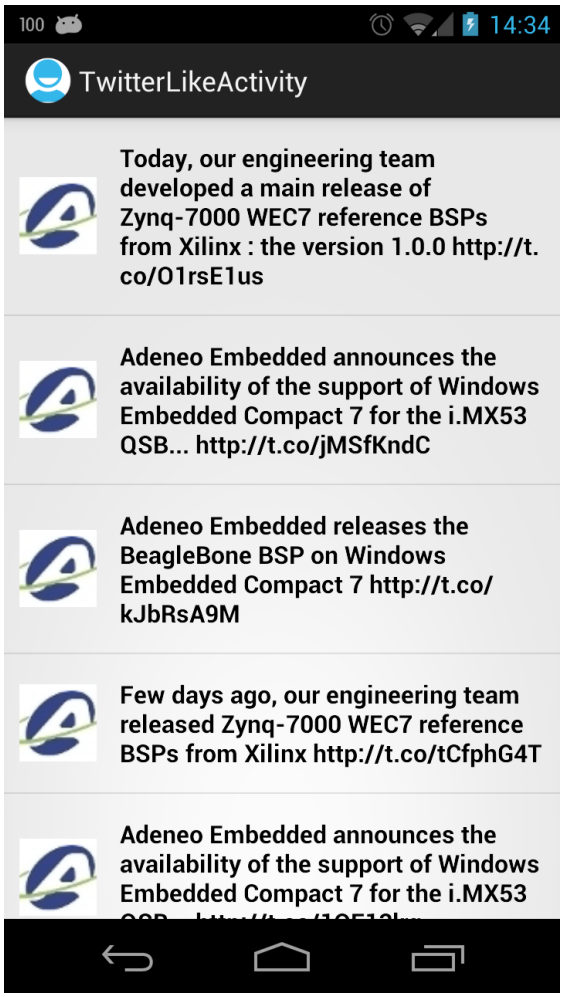
4. Select the new AVD from the Android Virtual Device Manager and click **Start**.

5. After the emulator boots up, unlock the emulator screen.

6. To run the app from Eclipse, open one of your project's files and click Run from the toolbar. Eclipse installs the app on your AVD and starts it.

Part 3: Create the application

The final app should looks like this.



It contains:

- A ListView containing tweets with text and associated avatar.

Goal:

- Display tweets inside a ListView.

- Create graphics elements
    - ListView

Open the xml file present in folder **res** -> **layout**.
You should see something like this:

```
<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"
    android:orientation="vertical"

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/hello_world"
        tools:context=".MainActivity" />

</LinearLayout>
```

First, delete the **TextView** element and add a **ListView** component.

```
    <ListView
        android:id="@+id/list"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
    </ListView>
```

id property is used to make a link between GUI and variables in java classes.
layout_width and **layout_height** corresponds of the component's size.
Value **match_parent** allow to the component to use the maximum space according to the size of its parent, here the linear layout.
Value **wrap_content** allow to the component to use only its content to maximum size.

- Items to display inside the **ListView**

Create another Layout that will contain components that will be displayed inside the **ListView**. We need to create an **ImageView** that will contain the avatar and a **TextView** that will contain the text. Save this new layout as **list_element.xml**. Name the **ImageView** and **TextView** as **avatar** and **tweet**, respectively.

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

     <ImageView android:id="@+id/avatar"
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:layout_gravity="center"
        android:scaleType="fitXY"
        android:layout_marginRight="15dp"
        android:layout_marginTop="2dp"
        android:layout_marginBottom="2dp" />

     <TextView android:id="@+id/tweet"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_vertical"
        android:layout_alignParentTop="true"
        android:layout_alignParentBottom="true"
        android:textStyle="bold"
        android:textSize="16sp"
        android:textColor="#000000"
        android:layout_marginTop="5dp"
        android:layout_marginBottom="5dp" />

</LinearLayout>
```

- Create a **Tweet** class

This class will be used to contain future tweets. It is composed of a **String** object and a **Drawable** object. Do not forget to add getter and setter functions for these private variables.

- Create a constructor for the **Tweet** class

```
      public Tweet(String tweet, Drawable avatar) {
            super();
            this.tweet = tweet;
            this.avatar = avatar;
      }
```

- Create the **TweetAdapter** class for the **ListView**

The L**istView** needs an **adapter** in order to display items. We will create a custom adapter named **TweetAdapter** that will display the avatar and the text of each tweet. This class must extend the **ArrayAdapter<Tweet>** object.

- Create the constructor

Parameters used here are the context of the application and an array which contains tweets. Set local variables with these parameters.

```
Context context;
ArrayList<Tweet> tweetsArray = null;

public TweetAdapter(Context context, ArrayList<Tweet> tweets) {
  super(context, R.layout.list_element, tweets);
  this.context = context;
  this.tweetsArray = tweets;
 }
```

- Override the **getView** method in the **TweetAdapter** class

This method will be in charge of building and loading each item inside the **ListView**. We need to use a **LayoutInflater** which is used to instantiate a layout XML file into its corresponding **View** objects. We have to inflate the item layout with its parent and then set its components, the **TextView** and the **ImageView** objects.

```
@Override
 public View getView(int position, View convertView, ViewGroup parent)
 {
       LayoutInflater inflater = (LayoutInflater)
      context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);

       View rowView = inflater.inflate(R.layout.list_element, parent,
                false);

       ImageView avatar = (ImageView) rowView.findViewById(R.id.avatar);
       TextView tweetText = (TextView) rowView.findViewById(R.id.tweet);

       avatar.setBackgroundDrawable(tweetsArray.get(position).getAvatar(
                 ));
       tweetText.setText(tweetsArray.get(position).getTweet());
       return rowView;
 }
```

  - Create a new class for downloading the tweets

Create a new class called **DownloadTweets** that extends **AsyncTask<Void, Integer, Long>.**

We will use an **AsyncTask** that allows performing background operations and publishing results on the UI thread without having to manipulate threads and/or handlers.

3 overridden methods must be implemented here, called in the following order:

- **onPreExecute** which runs on the UI thread before
- **doInBackground** which perform a computation on a background thread
- **onPostExecute** which also runs on the UI thread when **doInBackGround** finished.

- Fill up the **onPreExecute** method

Notify the user by a toast that the download is running.

```
            @Override
            protected void onPreExecute() {
                  super.onPreExecute();
                  Toast.makeText(getApplicationContext(), "Start downloading
tweets", Toast.LENGTH_LONG).show();
            }
```

- Fill up the **onPostExecute** method

Notify the user by a toast that the tweets have been downloaded.

```
            @Override
            protected void onPostExecute(Long result) {
                  Toast.makeText(getApplicationContext(), "Finished
                        downloading tweets", Toast.LENGTH_LONG).show();
                   tweetsList = (ListView) findViewById(R.id.list);
                   tweetsList.setAdapter(new TweetAdapter(getBaseContext(),
                        tweetsArray));
            }
```

- Create a function that fetches the Twitter feed

```
    public String readTwitterFeed() {
        StringBuilder builder = new StringBuilder();
        HttpClient client = new DefaultHttpClient();
        HttpGet httpGet = new
HttpGet("http://twitter.com/statuses/user_timeline/AdeneoEmbedded.json");
        try {
          HttpResponse response = client.execute(httpGet);
          StatusLine statusLine = response.getStatusLine();
          int statusCode = statusLine.getStatusCode();
          if (statusCode == 200) {
            HttpEntity entity = response.getEntity();
            InputStream content = entity.getContent();
            BufferedReader reader = new BufferedReader(new
InputStreamReader(content));
            String line;
            while ((line = reader.readLine()) != null) {
              builder.append(line);
            }
          } else {
            Log.e(MainActivity.class.toString(), "Failed to download file");
          }
        } catch (ClientProtocolException e) {
          e.printStackTrace();
        } catch (IOException e) {
          e.printStackTrace();
        }
        return builder.toString();
      }
```

- Create functions to download avatar pictures

```
    public Object fetch(String address) throws
MalformedURLException,IOException {
        URL url = new URL(address);
        Object content = url.getContent();
        return content;
    }

    private Drawable ImageOperations(String url) {
        try {
            InputStream is = (InputStream) this.fetch(url);
            Drawable d = Drawable.createFromStream(is, null);
            return d;
        } catch (MalformedURLException e) {
            e.printStackTrace();
            return null;
        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }
    }
```

- Fill up the **doInBackground** method

This method runs in a background thread. We will need to use a **JSONArray** to contain our JSON file converted as a String by the **readTwitterFeed** method.

```
      @Override
      protected Long doInBackground(Void... params) {

            try {
                        JSONArray jsonArray = new JSONArray(readTwitterFeed());

                        for (int i = 0; i < jsonArray.length(); i++) {
                                JSONObject jsonObject =
                                    jsonArray.getJSONObject(i);

                                Tweet tweet = new
                                     Tweet(jsonObject.getString("text"),
ImageOperations(jsonObject.getJSONObject("user").getString("profile_image_url"
)));
                                tweetsArray.add(tweet);
                           }
            } catch (Exception e) {
                  e.printStackTrace();
            }
            return null;
      }
```

- Launch the activity

Inside the **onCreate** method of the main activity, instantiate an object using **AsyncTask** and call the **execute** method. Do not forget to initialize member variables first.

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        this.tweetsList = (ListView) findViewById(R.id.list);
        this.tweetsArray = new ArrayList<Tweet>();

        new DownloadTweets().execute();
    }
```

- Add the required permissions

Edit the application's manifest file to request the following permission: **android.permission.INTERNET**

Launch the application and check that it works!

Lab 6: Create a phone contacts manager

Goals

- Create a phone contact manager
- Learn about the cursor object
- Implements control interface

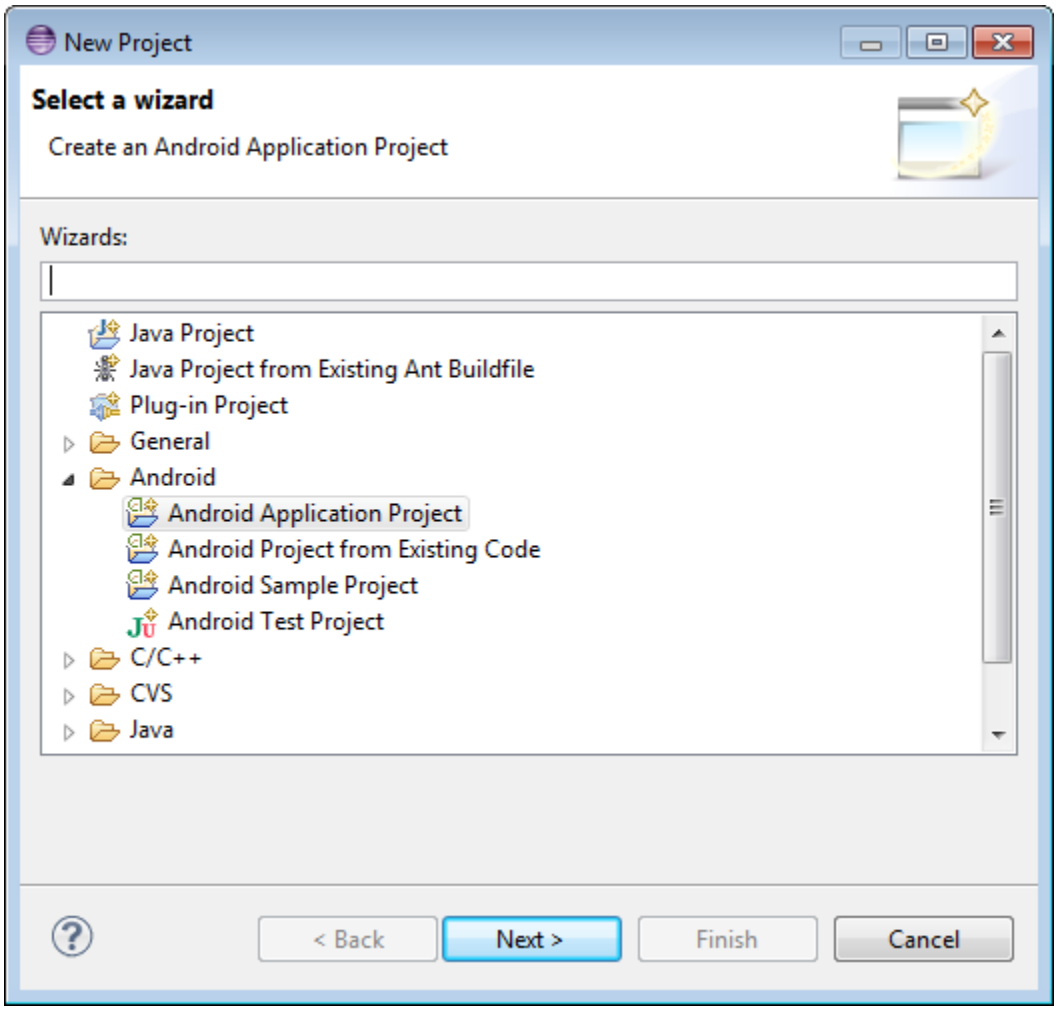
Estimated time: 75 minutes

Part 1: Create a new android application project

- **Creating the project**

1. Open the Eclipse IDE

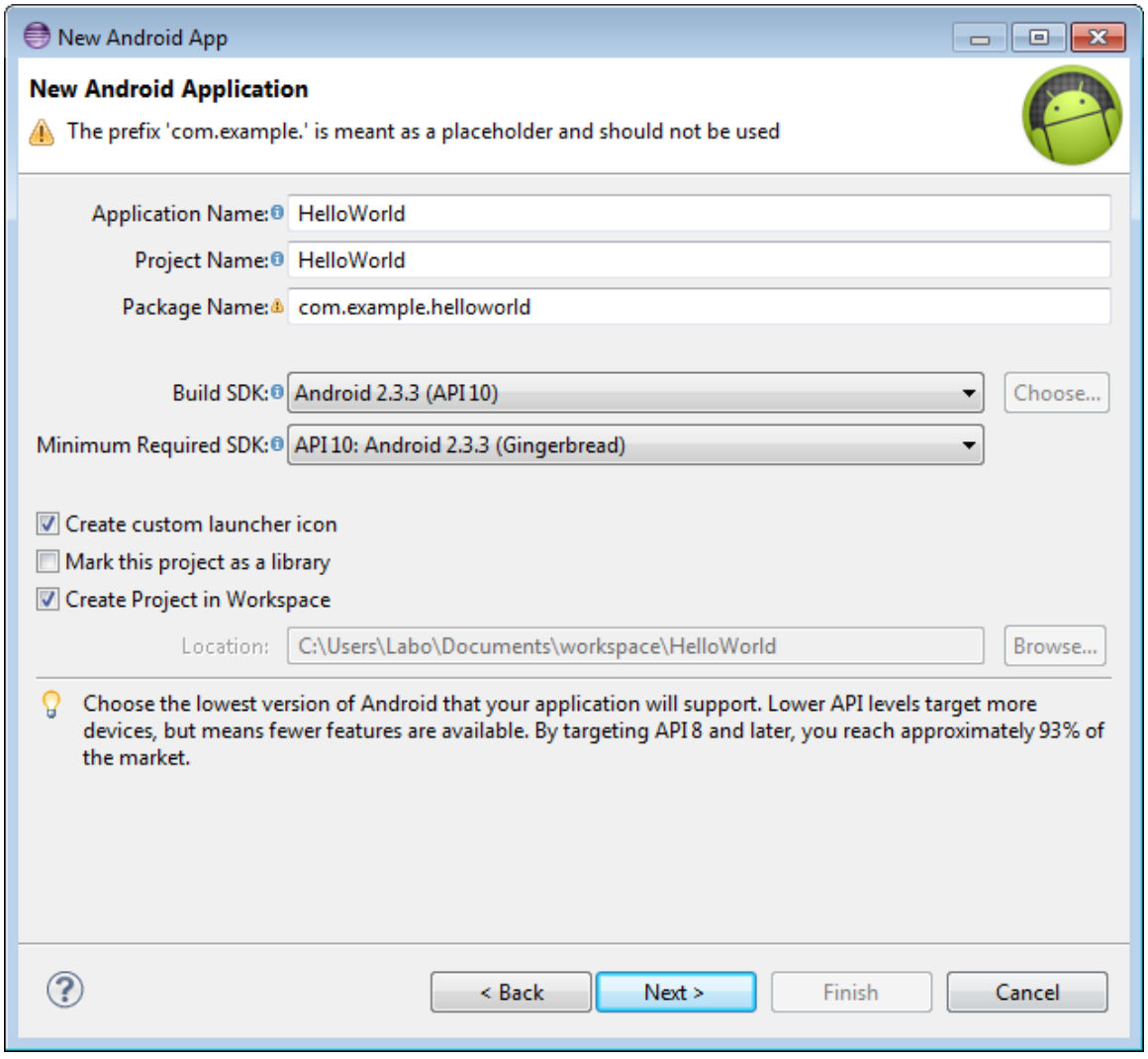
2. Select **File -> New -> New Project.** Select Android Application Project and click next.



3.

4. Fill in the form in the window that appeared like on the following screenshot and click **Next**.

    - Application name is the name that appears to users (application icon).
    - Project name corresponds to the name of your project directory (visible in Eclipse)
    - Package name corresponds to the namespace for you application
    - Build SDK is the platform version against which you will compile your app.
    - Minimum Required SDK is the lowest version of Android that your app supports.



5. Now you can select an activity template from which to begin building your app.

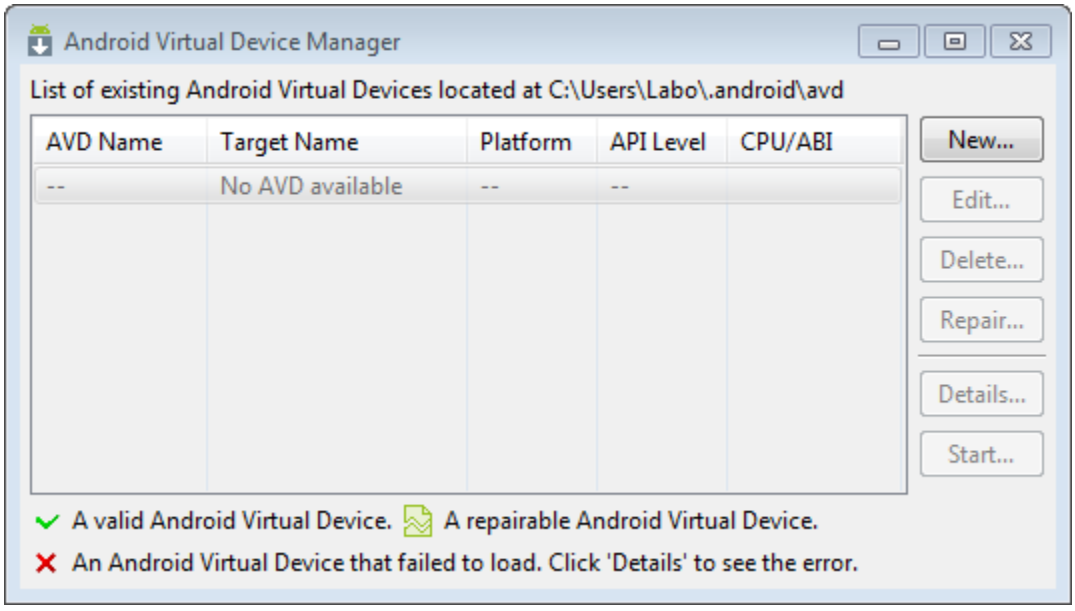
   For this project, select BlankActivity and click **Next**.

6. Leave all the details for the activity in their default state and click **Finish**.

Part 2: Creating an Android Virtual Device (AVD)

- **Create the AVD**

1. Launch the Android Virtual Device Manager via the Eclipse menu toolbar **Window | AVD Manager**.



2. In the Android Virtual Device Manager panel, click **New**.

3. Fill in the details for the AVD. Give it a name, a platform target, an SD card size, and a skin (HVGA is default). Click **Create AVD**.

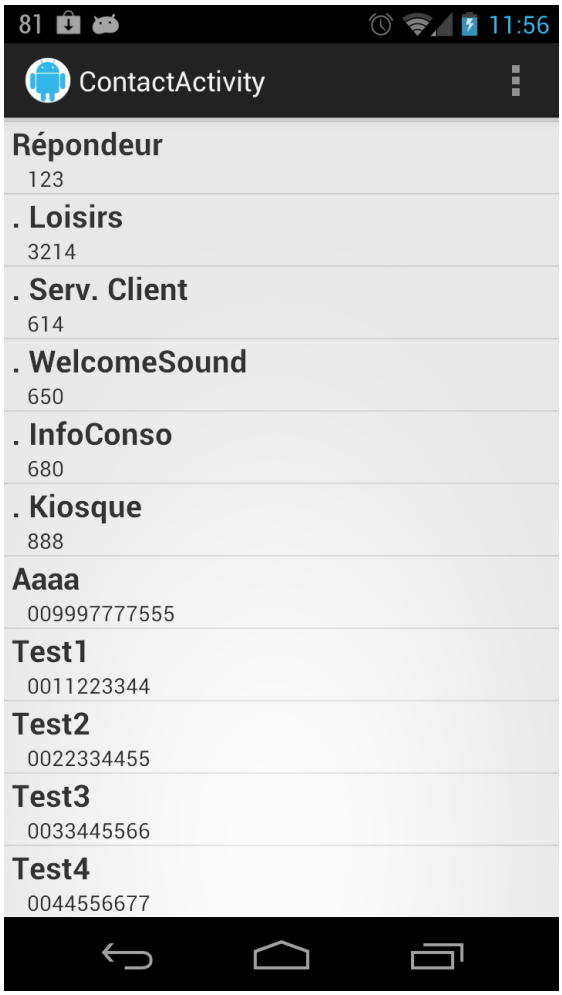
4. Select the new AVD from the Android Virtual Device Manager and click **Start**.

5. After the emulator boots up, unlock the emulator screen.

6. To run the app from Eclipse, open one of your project's files and click Run from the toolbar. Eclipse installs the app on your AVD and starts it.

Part 3: Create the application

The final app should looks like this.



It contains:

- A ListView containing phone's contacts displayed by name and associated number.

Goal:

- Are able to display phone's contact , add and remove contacts

- Create graphics elements
    - ListView

Open the xml file present in folder res -> **layout**.
You should see something like this:

```
<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"
    android:orientation="vertical"

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/hello_world"
        tools:context=".MainActivity" />

</LinearLayout>
```

First, delete the **TextView** element and add a **ListView** component.

```
    <ListView
        android:id="@+id/list"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
    </ListView>
```

id property is used to make a link between GUI and variables in java classes.
layout_width and **layout_height** corresponds of the component's size.
Value **match_parent** allow to the component to use the maximum space according to the size of its parent, here the linear layout.
Value **wrap_content** allow to the component to use only its content to maximum size.

- Items to show inside the **ListView**

Create a layout that contains components which will be displayed inside the **ListView**. We need to create two **TextView** objects. The first containing the contact's name and the second containing the contact's number.

- Create the main activity
  - Variables declaration

```
public class MainActivity extends Activity {

private ListView listView;
private ArrayList<HashMap<String, String>> arrayListContacts;
private HashMap<String, String> mapContact;
private SimpleAdapter listAdapter;
```

  - Fill up the list view using phone contacts

In the main activity's **OnCreate** function, add the following code to populate the **ListView** object with the phone's contact.

```
            listView = (ListView) findViewById(R.id.list);

            arrayListContacts = new ArrayList<HashMap<String, String>>();

            //get the result of the query inside a cursor
            Cursor cursor =
getContentResolver().query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
null, null,null, null);
            while (cursor.moveToNext()) {
                  //get result inside NAME column
                  String contactName =
(cursor.getString(cursor.getColumnIndex(ContactsContract.CommonDataKinds.Phone
.DISPLAY_NAME)));
                  //get result inside NUMBER column
                  String contactNumber =
(cursor.getString(cursor.getColumnIndex(ContactsContract.CommonDataKinds.Phone
.NUMBER)));
                  //put Name and number inside the map
                  mapContact = new HashMap<String, String>();
                  mapContact.put("name", contactName);
                  mapContact.put("number", contactNumber);

                  //add the new map to the array
                  arrayListContacts.add(mapContact);
            }
            cursor.close();

            //create the adapter used by the listView in order to display
items
            listAdapter = new SimpleAdapter(this.getBaseContext(),
                        arrayListContacts, R.layout.contact_list, new
String[] { "name",
                                    "number" }, new int[] { R.id.name,
                                    R.id.number });

            //set the adapter to the listView
            listView.setAdapter(listAdapter);
```

- Add the required permissions

Edit the application's manifest file to request the following permission: **android.permission.READ_CONTACTS**

Launch your application and verify that your contacts are listed. If it is not the case, then your phone most likely has no contact in its phone book!