
Lab 1: Create your first « Hello World » using AVD

Goals

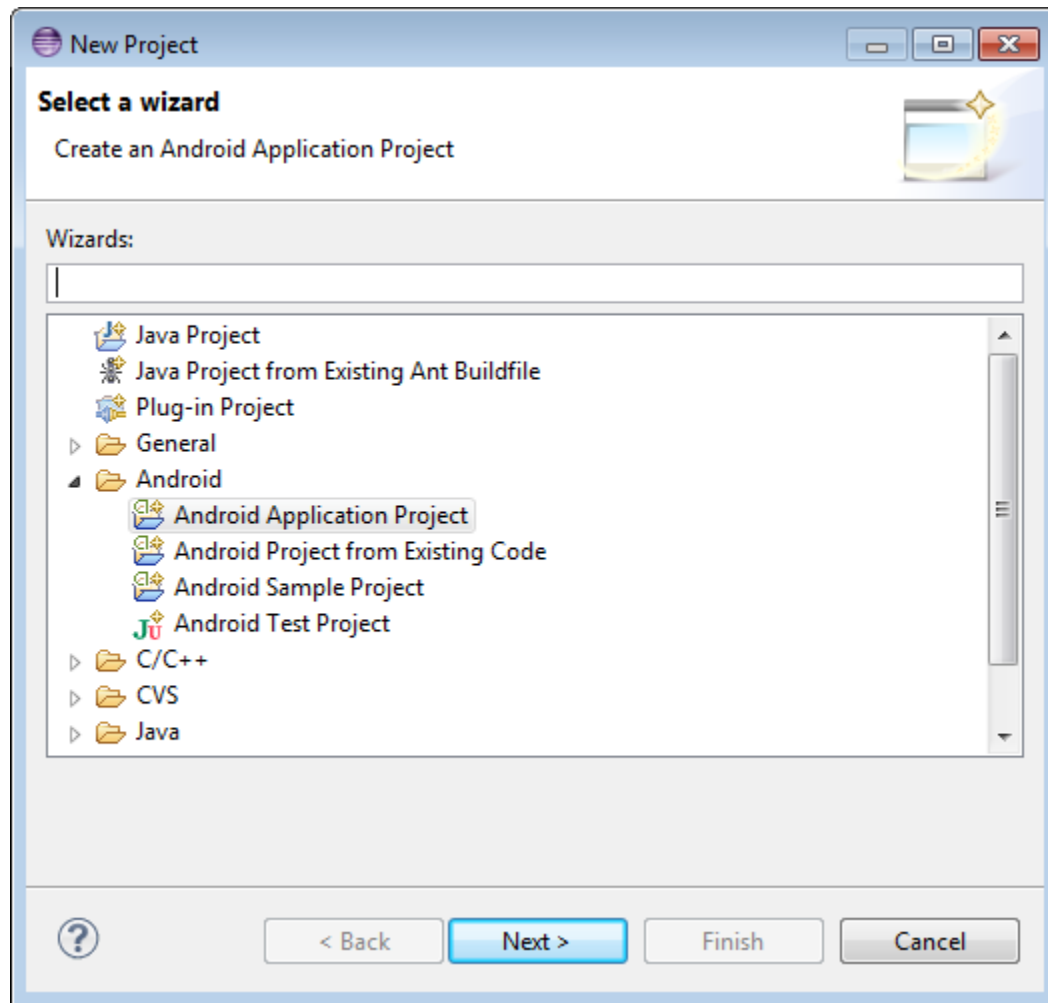
- Create a new hello world project
- Create an android virtual device
- Deploy the application to the virtual device

Estimated time: 30 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. 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.

New Android Application

⚠ The prefix 'com.example.' is meant as a placeholder and should not be used

Application Name: HelloWorld

Project Name: HelloWorld

Package Name: com.example.helloworld

Build SDK: Android 2.3.3 (API 10) Choose...

Minimum Required SDK: API 10: Android 2.3.3 (Gingerbread)

Create custom launcher icon

Mark this project as a library

Create Project in Workspace

Location: C:\Users\Labo\Documents\workspace\HelloWorld Browse...

💡 Choose the lowest version of Android that your application will support. Lower API levels target more devices, but means fewer features are available. By targeting API 8 and later, you reach approximately 93% of the market.

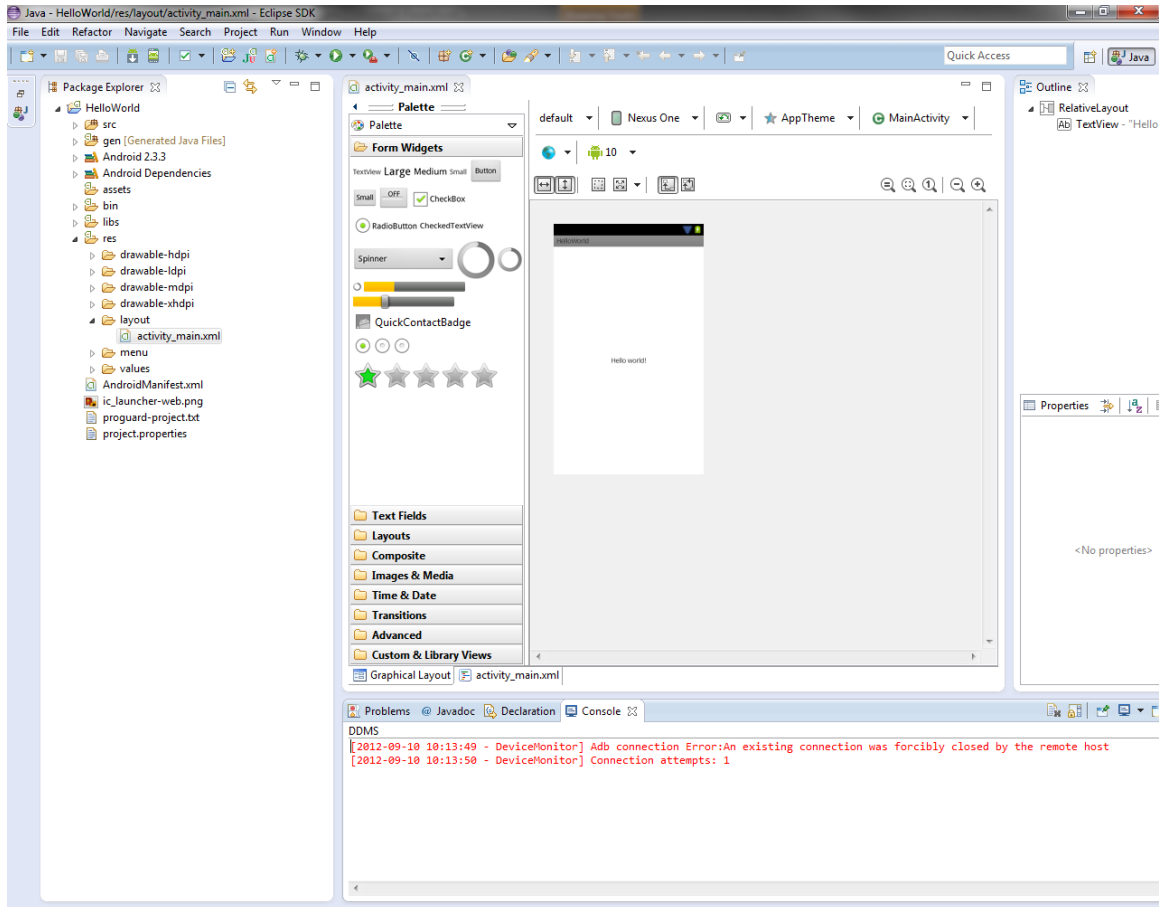
< Back Next > Finish Cancel

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

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

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

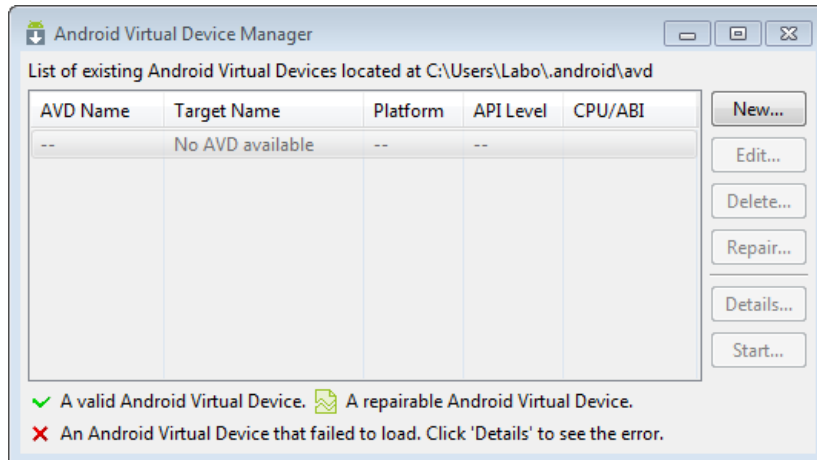
You should end up with the following view:



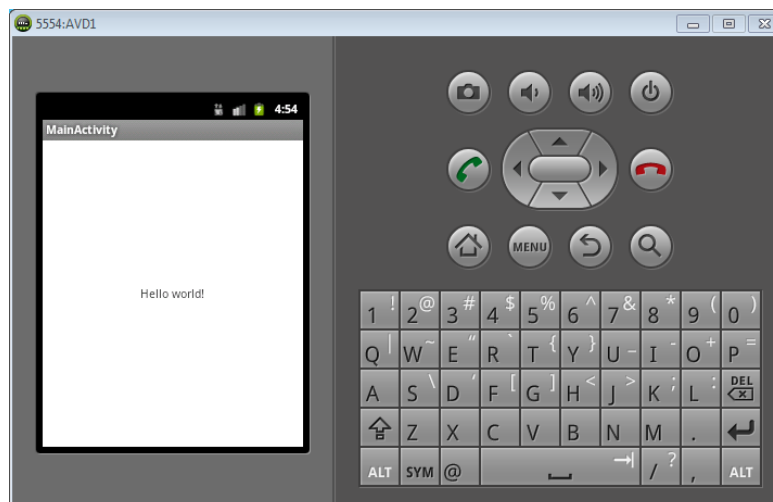
Part 2: Creating an Android Virtual Device (AVD)

➤ 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.



Lab 2: Create an alarm clock with notification reminder

Goals

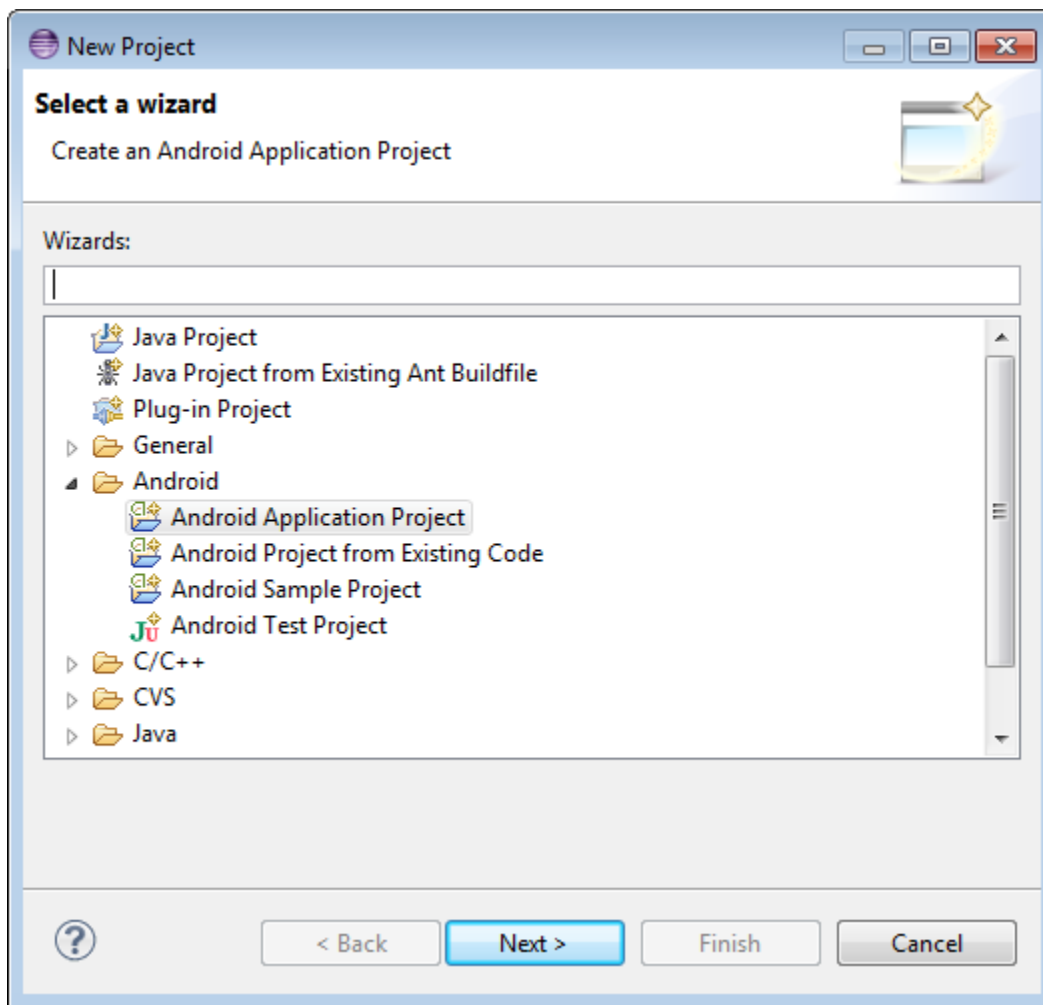
- Use Broadcast receiver
- Use Notifications

Estimated time: 40 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.

The screenshot shows the 'New Android App' dialog box with the following details:

- Title:** New Android Application
- Warning:** The prefix 'com.example.' is meant as a placeholder and should not be used.
- Application Name:** HelloWorld
- Project Name:** HelloWorld
- Package Name:** com.example.helloworld
- Build SDK:** Android 2.3.3 (API 10)
- Minimum Required SDK:** API 10: Android 2.3.3 (Gingerbread)
- Options:**
 - Create custom launcher icon
 - Mark this project as a library
 - Create Project in Workspace
- Location:** C:\Users\Labo\Documents\workspace\HelloWorld
- Buttons:** < Back, Next > (highlighted), Finish, Cancel

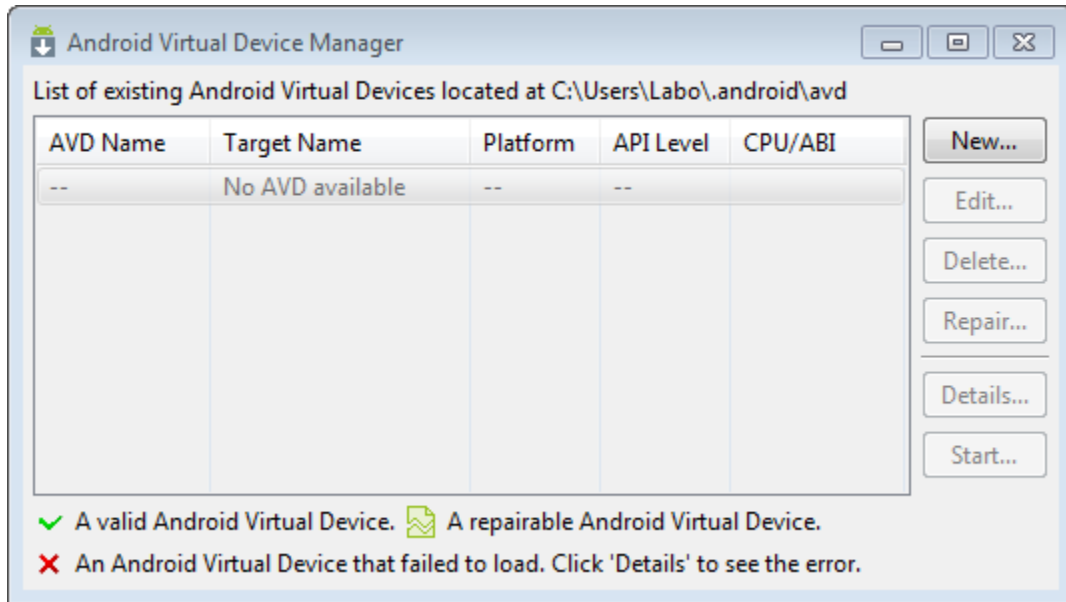
5. Now you can select an activity template from which to begin building your app.

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

Part 2: Creating an Android Virtual Device (AVD)

➤ Create the AVD

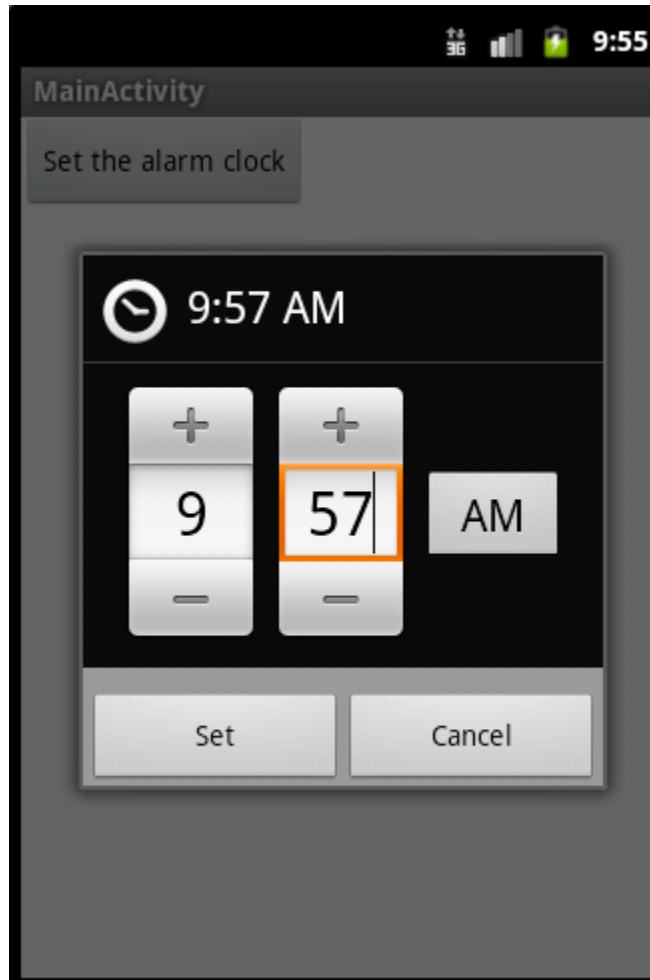
1. Launch the Android Virtual Device Manager via the Eclipse menu toolbar **Window | AVD Manager**.



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. Follow the steps below to reach that goal.



It contains:

- A Button to run a TimerPickerDialog

- 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/btnClock"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:text="Set the alarm clock"
    android:onClick="btnSetClicked"/>
```

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.

➤ Create the `TimerPickerDialog`

To display a `TimerPickerDialog` using `DialogFragment`, we need to define a fragment class that extends `DialogFragment` and return a `TimePickerDialog` from the fragment's `onCreateDialog` method.

- Set the `onCreateDialog` method

We need to use the `Calendar` class in order to get the **current `DateTime`**.

```
final Calendar c = Calendar.getInstance();
int hour = c.get(Calendar.HOUR_OF_DAY);
int minute = c.get(Calendar.MINUTE);
```

Then return an object `TimePickerDialog` using params as the current activity, the context, current hour, current minute and an hour format.

```
return new TimePickerDialog(getActivity(), this, hour, minute,
    DateFormat.is24HourFormat(getActivity()));
```

- Set the `onTimeSet` method

This method is call when the picker has been set and the button “OK” has been pressed.

Create a new intent for our `BroadcastReceiver` getting the context and the java class that extends Broadcast receiver. Then create a Pending receiver using the intent.

```
Intent intent = new Intent(getActivity().getBaseContext(),
    OnetimeAlarmReceiver.class);
PendingIntent pendingIntent =
    PendingIntent.getBroadcast(getActivity().getBaseContext(), 0,
    intent, 0);
```

Set a new `Date` object using `timePicker` params.

```
Calendar calendar = Calendar.getInstance();
Date alarmTime = new Date(System.currentTimeMillis());
alarmTime.setHours(hourOfDay);
alarmTime.setMinutes(minute);
calendar.setTimeInMillis(alarmTime.getTime());
```

Create an `AlarmManager` which provides access to the system alarm services. These allow scheduling an application to be run at some point in the future, here sending the broadcast using the `pendingIntent`.

```
AlarmManager alarmManager = (AlarmManager)
    getActivity().getApplicationContext().getSystemService("alarm");
alarmManager.set(AlarmManager.RTC_WAKEUP,
    calendar.getTimeInMillis(), pendingIntent);
```

➤ Create the Broadcast receiver

Create a new class that extends `BroadcastReceiver`. A new method is created, `onReceive`. This method is called each time that an intent will be send to the broadcast receiver.

First add a constant holding the notification ID:

```
private int NOTIFICATION_ID = 444555666;
```

In this method, create a `NotificationManger`

```
final NotificationManager notificationManager = (NotificationManager)
context.getSystemService(Context.NOTIFICATION_SERVICE);
```

Create a **Notification** with params as the resource that will be set as icon, the text that flows by in the status bar when the notification first activates, and the time to show in the time field

```
final Notification notification = new
Notification(R.drawable.ic_launcher, "Wake up alarm",
System.currentTimeMillis());
```

Create a **pendingIntent** in order to run the main activity when we click on the notification.

```
final PendingIntent pendingIntent = PendingIntent.getActivity(context,
0, new Intent(context, AlarmClockActivity.class), 0);
notification.setLatestEventInfo(context, "Alarm clock", "It's time !",
pendingIntent);
```

Set the **vibrator** when the notification appears.

```
notification.vibrate = new long[] {0,200,100,200,100,200};
```

Then launch it.

```
notificationManager.notify(NOTIFICATION_ID, notification)
```

- Handle the button click event

In the main activity of your application, implement the *btnSetClicked* function that is targeted by the button from the view's xml file. First make sure your activity extends **FragmentActivity** instead of **Activity**.

```
public class MainActivity extends FragmentActivity {  
  
    ...  
  
    public void btnSetClicked(View v) {  
        DialogFragment newFragment = new TimePickerFragment();  
        newFragment.show(getSupportFragmentManager(), "timePicker");  
    }  
}
```

Run your application in the emulator and check if it works as expected.

Lab 3: Create a money converter

Goals

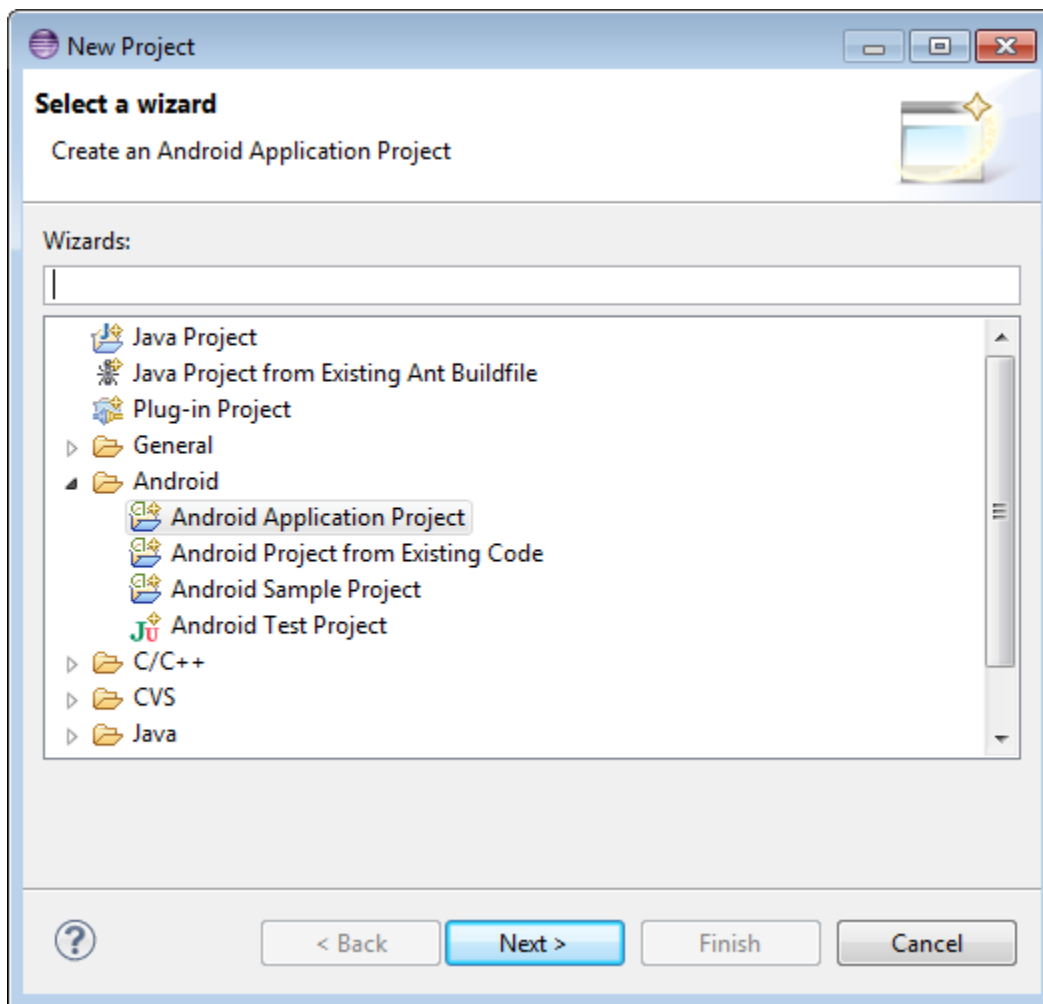
- Create a money converter
- Learn about Layout XML for handling different screen sizes
- Implement a control interface

Estimated time: 45 minutes

Part 1: Create a new android application project

➤ Creating the project

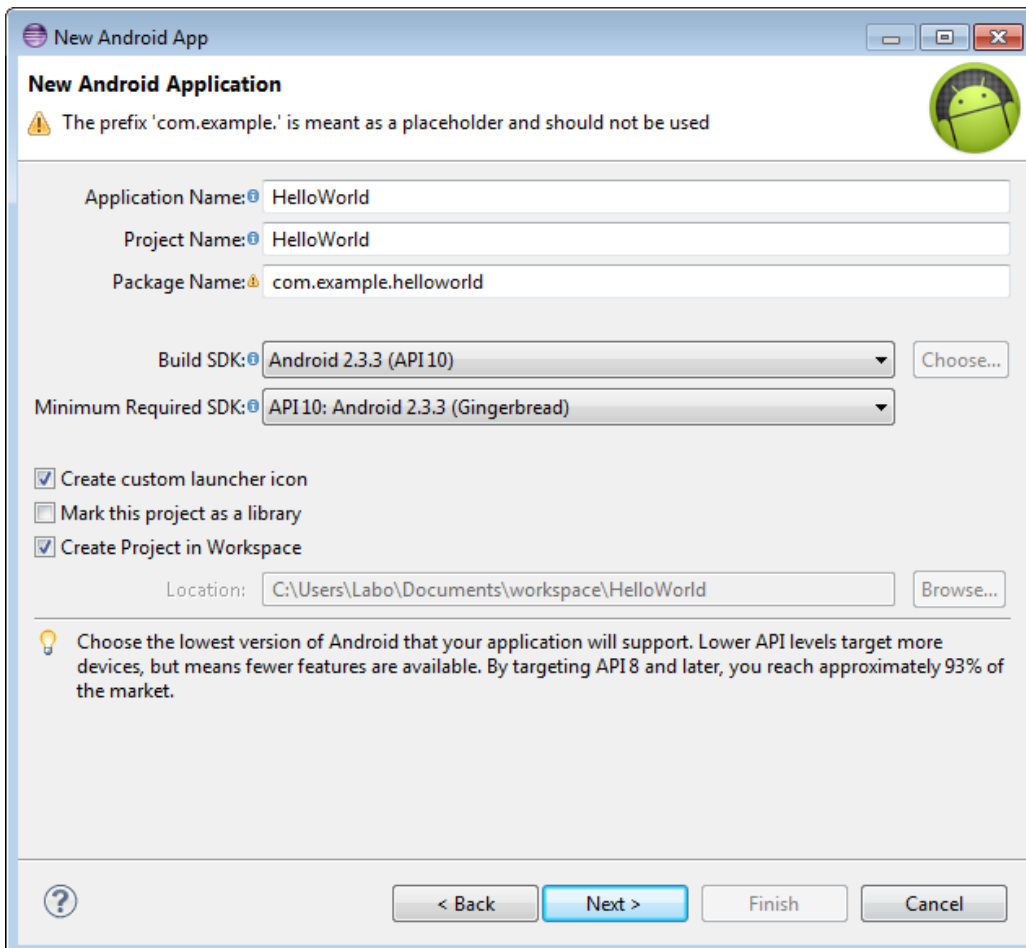
1. Open the Eclipse IDE
2. Select **File -> New -> New Project**. Select **Android Application Project** and click next.



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.



New Android Application

⚠ The prefix 'com.example.' is meant as a placeholder and should not be used

Application Name: HelloWorld

Project Name: HelloWorld

Package Name: com.example.helloworld

Build SDK: Android 2.3.3 (API 10) Choose...

Minimum Required SDK: API 10: Android 2.3.3 (Gingerbread)

Create custom launcher icon

Mark this project as a library

Create Project in Workspace

Location: C:\Users\Labo\Documents\workspace\HelloWorld Browse...

💡 Choose the lowest version of Android that your application will support. Lower API levels target more devices, but means fewer features are available. By targeting API 8 and later, you reach approximately 93% of the market.

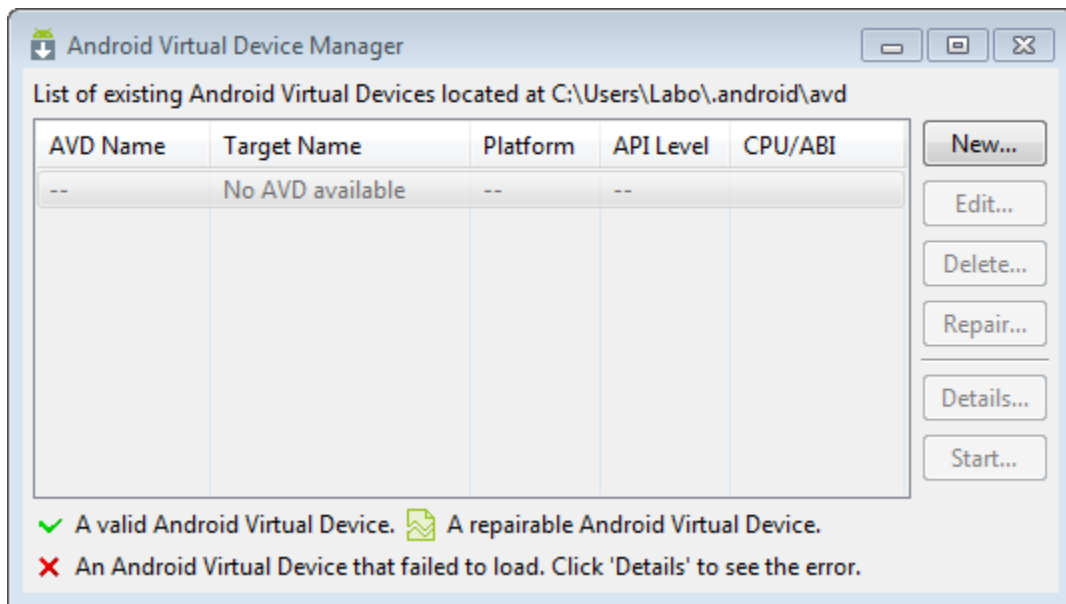
< Back **Next >** Finish Cancel

- Now you can select an activity template from which to begin building your app.
For this project, select **BlankActivity** and click **Next**.
- 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

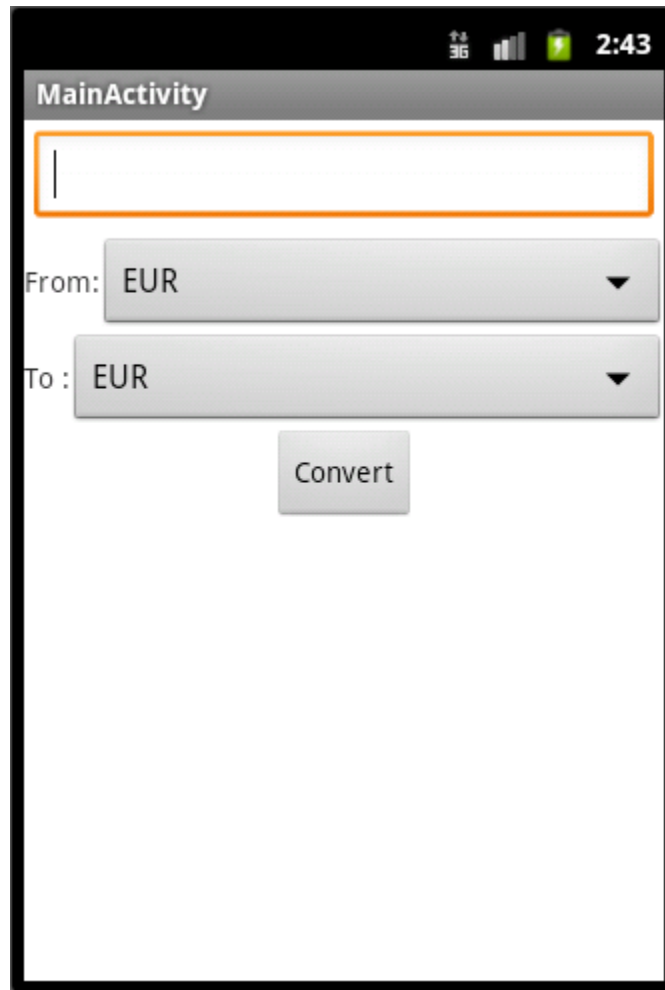
- Launch the Android Virtual Device Manager via the Eclipse menu toolbar **Window | AVD Manager**.



- In the Android Virtual Device Manager panel, click **New**.
- 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**.
- Select the new AVD from the Android Virtual Device Manager and click **Start**.
- After the emulator boots up, unlock the emulator screen.
- 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. Follow the steps below to create this application.



It contains:

- An EditText containing the value to convert and the value which will be converted.
- Two TextView and two Spinners containing currencies values.
- A Button launching the conversion

- 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 an **EditText** component.

```
<EditText
    android:id="@+id/editTextCurrencieValue"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="5dp"
    android:inputType="numberDecimal"
    android:singleLine="true"
/>
```

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.

Do the same thing with others components: **TextView**, **Spinner**, and **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

- Variables declaration

Add member variables as private with the same type as XML's components

```
public class ConvertisseurActivity extends Activity {  
    private EditText editTextCurrencieValue;
```

- Linking between activity's members variables and GUI components

Inside the **onCreate** method, use method **findViewById** in order to link your private variable with your GUI component using its **id**.

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_convertisseur);  
  
    //Link between members variables and GUI Components  
    editTextCurrencieValue = (EditText)  
        findViewById(R.id.editTextCurrencieValue);
```

Do the same thing for the other components.

- Create resources for your spinners

Spinners need to use a String Array containing values to show. Create a new xml file inside folders **res->values** and name it *currencies_array.xml*. Your file should be laid like this:

```
<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <string-array name="currencies_array">  
        <item>EUR</item>  
        <item>USD</item>  
        <item>GBP</item>  
        <item>CAD</item>  
    </string-array>  
</resources>
```

- Create an adapter for spinners containing the array resources

```
ArrayAdapter<CharSequence> adapter =  
ArrayAdapter.createFromResource(this, R.array.currencies_array,  
android.R.layout.simple_spinner_item);  
  
adapter.setDropDownViewResource(android.R.layout.simple_spinner_d  
ropdow_item);
```

- Set the **adapter** to spinners

```
spinnerFrom.setAdapter(adapter);  
spinnerTo.setAdapter(adapter);
```

- Set the interface **onClickListener** to your button

```
buttonCurrencyConvert.setOnClickListener(new OnClickListener() {  
  
    //detect a click occured on the Convert button  
    public void onClick(View v) {  
  
    }  
});
```

Everything should be ok in the **onCreate** method.

- Create an array containing currencies values

```
//Array containing currencies conversions in this order EUR,USD,CAD,AUD  
public double[][] constructDevisArray() {  
    double deviseArray [][] = { {1,1.234,0.788,1.226},  
                                {0.809,1,0.638,0.994},  
                                {1.268,1.565,1,1.556},  
                                {0.815,1.005,0.642,1} };  
  
    return deviseArray;  
}
```

- Create interface **onItemSelectedListener** for spinners

In order to detect which values are selected in spinners we need to create an **onItemSelectedListener** interface.

The new class needs to implements this interface. Add this on the declaration line of the class:

```
implements OnItemSelectedListener
```

Add these unimplemented methods.

```
public void onItemSelected(AdapterView<?> arg0, View arg1, int arg2,  
    long arg3) {  
  
public void onNothingSelected(AdapterView<?> arg0) {  
  
}
```

In this project we will use only the first one.

- Detect which spinner changed

Inside **onItemSelected** we need to detect whose spinner value changed. To do this, we get view's parent which was used inside the interface. Then we can retrieve the index value using the param arg2.

```
if (((Spinner)arg1.getParent()).equals(spinnerFrom))  
    indexSpinnerFrom = arg2;  
  
else if (((Spinner)arg1.getParent()).equals(spinnerTo))  
    indexSpinnerTo = arg2;
```

- Set the **onItemSelectedListener** interface to spinners

In the method **onCreate**, add these two lines:

```
spinnerFrom.setItemSelectedListener(this);  
spinnerTo.setItemSelectedListener(this);
```

Where **this** is a reference of the interface declared inside the main class.

- Perform the conversion

Return inside the **onClick** method and convert **editText**'s value.

To do this:

- get the editText's value
- get the conversion value inside the currencies conversion array
- make calculation
- set the editText with the result value.

```
final double[][] deviseArray = constructDeviseArray();

buttonCurrencyConvert.setOnClickListener(new OnClickListener() {
    DecimalFormat df = new DecimalFormat("#####.00");

    public void onClick(View v) {

        if(!editTextCurrencieValue.getText().toString().equals("")) {

            double valueToConvert =
                Double.parseDouble(editTextCurrencieValue.getText().toString());

            editTextCurrencieValue.setText(String.valueOf(df.format(
                valueToConvert*deviseArray[indexSpinnerFrom][indexSpinnerTo])).
                replace(',', '.'));

        }
    }
}
```

- Build your application and try it in the emulator