Lab 7: Create a picture viewer application

Goals

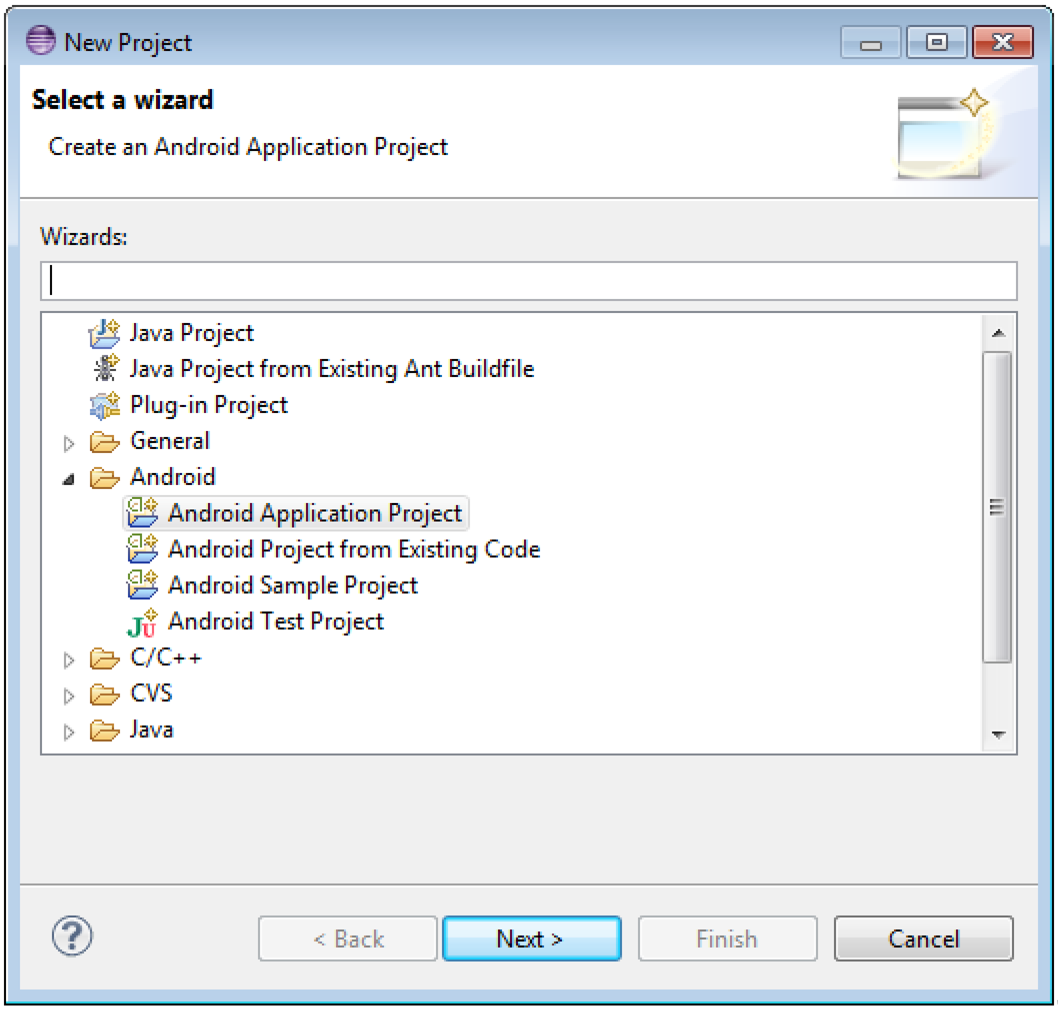
- Learn about custom gallery objects
- Learn about animations

Estimated time: 50 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.



New Project

Select a wizard

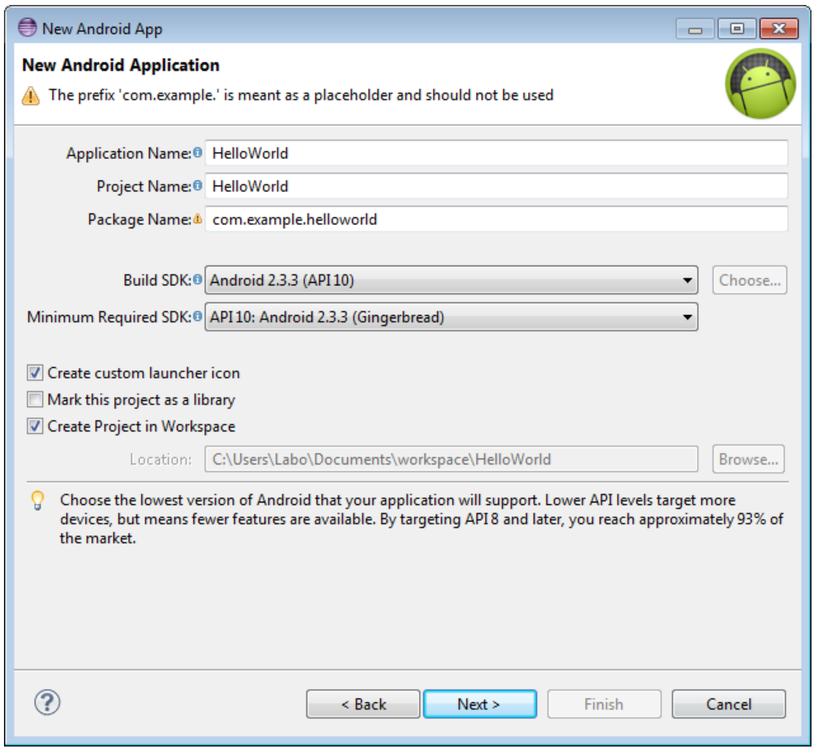
Create an Android Application Project

Wizards:

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)
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

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.

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

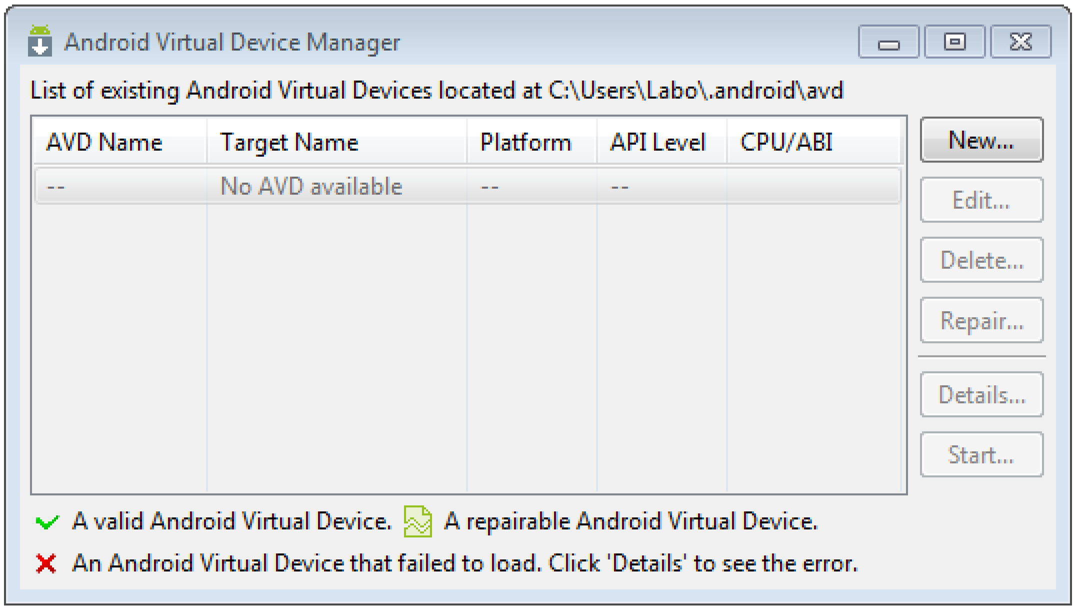
   For this project, select BlankActivity and click **Next**.

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**.



List of existing Android Virtual Devices located at C:\Users\Labo\.android\avd

AVD Name	Target Name	Platform	API Level	CPU/ABI
--	No AVD available	--	--	

✓ A valid Android Virtual Device. A repairable Android Virtual Device.

✗ An Android Virtual Device that failed to load. Click 'Details' to see the error.

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**.

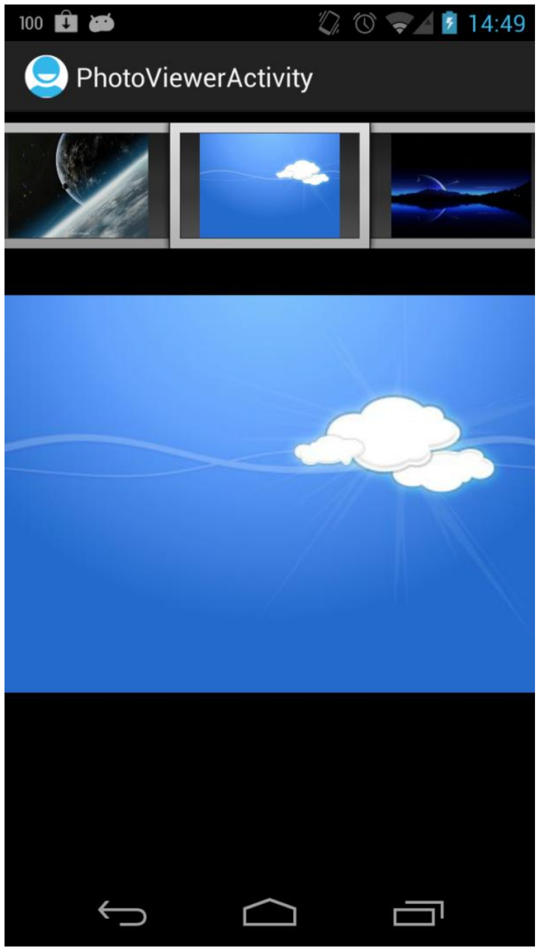
4. Select the new AVD from the Android Virtual Device Manager and click **Start**.

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:

- A Gallery displaying pictures
- An **ImageSwitcher** displaying pictures with transition animations.

Goal:

- Display pictures and use animations as transition

- Create the main activity

    - Get device’s resources

We will get all pictures stored on the devices and display them. We need to use a **Cursor** object, which is used to query read-write access to a database. We then navigate through the database using this object and add all available pictures to a bitmap array. Add the following code to your main activity’s **OnCreate** function.

```
    private ImageSwitcher imageSwitcher;
    private ArrayList<Bitmap> bitmapArray;

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

        Cursor cur =
getContentResolver().query(MediaStore.Images.Media.EXTERNAL_CONTENT_URI, null,
null, null, MediaStore.Images.Media.DEFAULT_SORT_ORDER);
        cur.moveToFirst();
        bitmapArray = new ArrayList<Bitmap>();
            int positionId = cur.getColumnIndex(MediaStore.Images.Media._ID);

            while (!cur.isAfterLast()) {
                try {
                    long id = cur.getLong(positionId);
                    Bitmap b =
MediaStore.Images.Thumbnails.getThumbnail(getContentResolver(), id,
                                       MediaStore.Images.Thumbnails.MINI_KIND,
null);

                    bitmapArray.add(b);
                } catch (Exception e) {
                    System.out.println("out of memory");
                    return;
                }
                cur.moveToNext();
            }
    }
```

- Prepare the layout

Open your main activity's layout and add **ImageSwitcher** and **Gallery** objects:

```
 <ImageSwitcher
     android:id="@+id/switcher"
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:layout_alignParentLeft="true"
     android:layout_alignParentRight="true"
     android:layout_alignParentBottom="true"
     />

  <Gallery
     android:id="@+id/gallery"
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
         />
```

- Extend the main activity

The main activity needs to extend the **ViewFactory** class**.** This interface is used to create views in an **ImageSwitcher** object.

```
public class MainActivity extends Activity implements ViewFactory
```

- Implement the **MakeView** method

The **ViewFactory** requires the implementation of the **MakeView** method. Implement this function into your main activity. It should return the view that will be added to the **ImageSwitcher** object.

```
public View makeView()
{
  ImageView imageView = new ImageView(this);
  imageView.setBackgroundColor(0xFF000000);
  imageView.setScaleType(ImageView.ScaleType.FIT_CENTER);
  imageView.setLayoutParams(new ImageSwitcher.LayoutParams(
                                      LayoutParams.MATCH_PARENT,
                                      LayoutParams.MATCH_PARENT));
  return imageView;
}
```

- Prepare the **ImageSwitcher** object

In the main activity's **OnCreate** function, make the link between the **ImageSwitcher** variable and the XML layout. Then set **Animation** when pictures will appear and disappear.

```
        imageSwitcher = (ImageSwitcher) findViewById(R.id.switcher);
        imageSwitcher.setFactory(this);
        imageSwitcher.setInAnimation(AnimationUtils.loadAnimation(this,
                  android.R.anim.slide_in_left));
        imageSwitcher.setOutAnimation(AnimationUtils.loadAnimation(this,
                  android.R.anim.slide_out_right));
```

- Prepare the **Gallery** object

The gallery needs to use an adapter in order to display pictures. We will create the **adapter** in the next step. We also need to set an **onItemClickListener** interface to the gallery to detect a user click on a picture to display it inside the **ImageSwitcher.** The bitmap needs to be converted into a **Drawable** object to be used by the **imageSwitcher**. Add the following code at the end of the main activity's **OnCreate** function.

```
         Gallery gallery = (Gallery) findViewById(R.id.gallery);
         gallery.setAdapter(new ImageAdapter(this));
         gallery.setOnItemClickListener(new OnItemClickListener() {
             public void onItemClick(AdapterView parent,
             View v, int position, long id)
             {
              if (bitmapArray != null && bitmapArray.size()>0) {
              Drawable drawable =new
BitmapDrawable(getResources(),bitmapArray.get(position));
              imageSwitcher.setImageDrawable(drawable);
              }
             }
         });
```

- Declare the **Gallery** as a styleable resource

Create a new **Android XML values file** to your project. Add the following XML code to declare the **Gallery** as a styleable resource.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <declare-styleable name="Gallery">
        <attr name="android:galleryItemBackground" />
    </declare-styleable>
</resources>
```

- Create the image Adapter

Create an inner class called **ImageAdapter** which extends the **BaseAdapter** class. Fill up the required functions as follows:

```
    public class ImageAdapter extends BaseAdapter
    {
        private Context context;
        private int itemBackground;

        public ImageAdapter(Context c)
        {
            context = c;

            //---setting the style---
            TypedArray a = obtainStyledAttributes(R.styleable.Gallery);
            itemBackground =
a.getResourceId(R.styleable.Gallery_android_galleryItemBackground, 0);
            a.recycle();
        }

        //---returns the number of images---
        public int getCount()
        {
            return bitmapArray.size();
        }

        //---returns the ID of an item---
        public Object getItem(int position)
        {
            return position;
        }

        public long getItemId(int position)
        {
            return position;
        }

        //---returns an ImageView view---
        public View getView(int position, View convertView, ViewGroup parent)
        {
            ImageView imageView = new ImageView(context);
            imageView.setImageBitmap(bitmapArray.get(position));
            imageView.setLayoutParams(new Gallery.LayoutParams(300, 200));
            imageView.setScaleType(ImageView.ScaleType.FIT_CENTER);
            imageView.setBackgroundResource(itemBackground);
            return imageView;
        }
    }
```

Launch your application and play around with the pictures. If you get a black screen, then your device most likely has no picture in its storage!

Lab 8: Create an animated Wallpaper

Goals

- Draw a frame
- Learn about Engine Class
- Learn about Paint Class

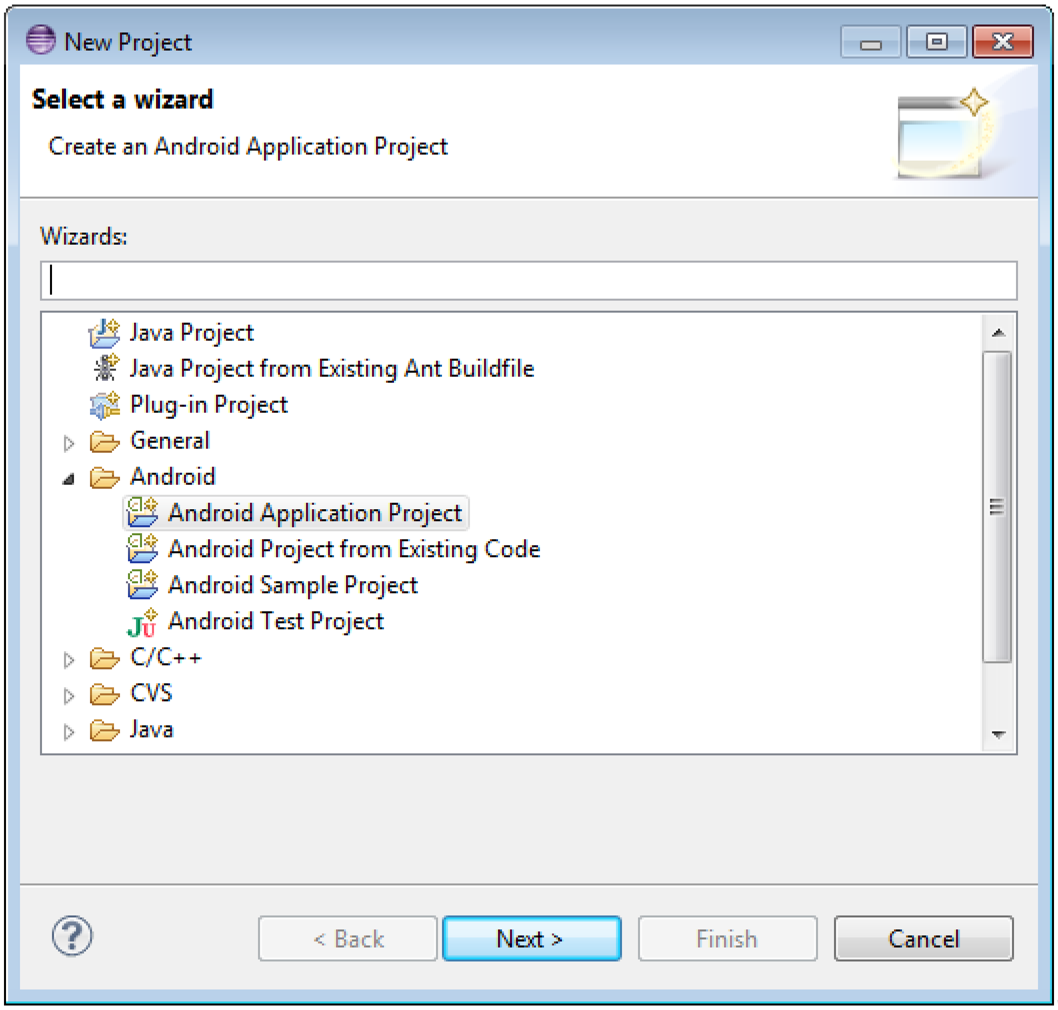
Estimated time: 55 minutes

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.


New Project

Select a wizard

Create an Android Application Project

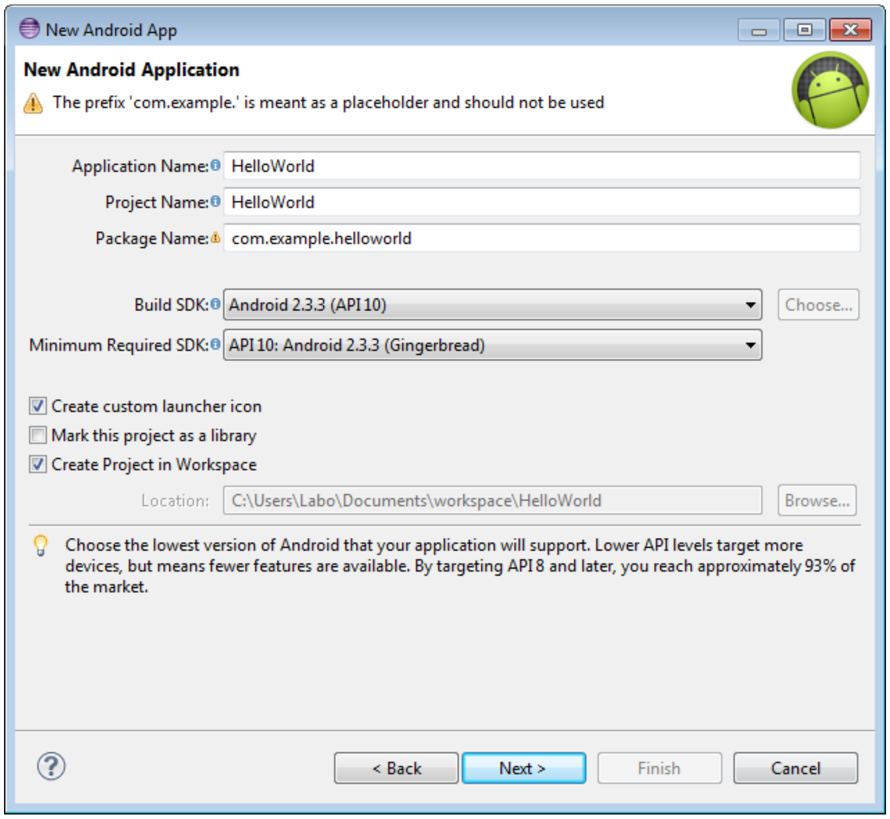
Wizards:

- Java Project
- Java Project from Existing Ant Buildfile
- Plug-in Project
- General
- Android
  - Android Application Project
  - Android Project from Existing Code
  - Android Sample Project
  - Android Test Project
- C/C++
- CVS
- Java

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)

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

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.

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

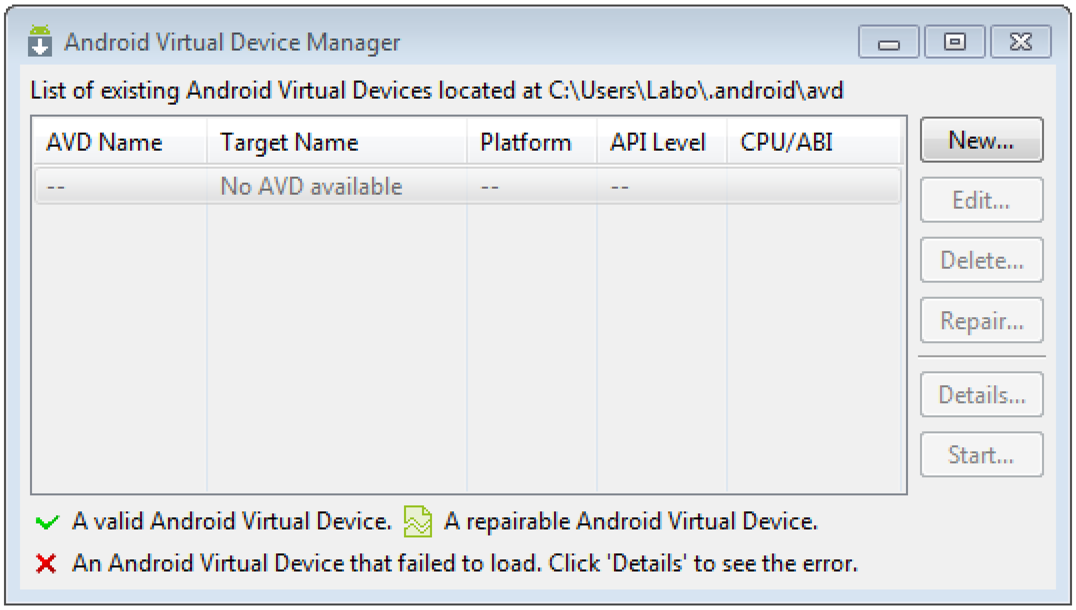
   For this project, select BlankActivity and click **Next**.

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**.



List of existing Android Virtual Devices located at C:\Users\Labo\.android\avd

AVD Name	Target Name	Platform	API Level	CPU/ABI
--	No AVD available	--	--	

✓ A valid Android Virtual Device. A repairable Android Virtual Device.
✗ An Android Virtual Device that failed to load. Click 'Details' to see the error.

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**.

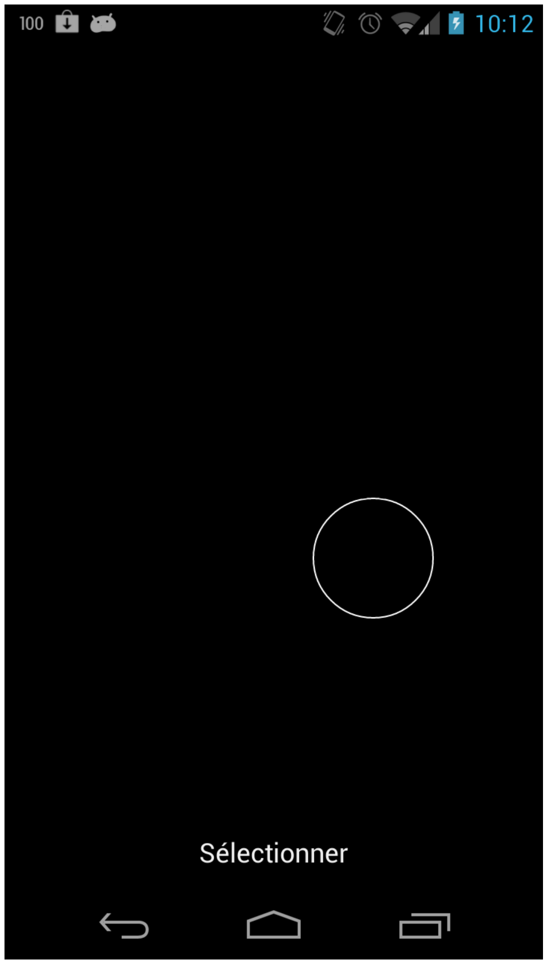
4. Select the new AVD from the Android Virtual Device Manager and click **Start**.

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:

- A circle which is drawn each time the user is touching the screen

Goal:

- Be able to create an animated wallpaper using Paint and Canvas classes

- Prepare the main activity

First, the main activity needs to extend the **WallpaperService** class which is responsible for showing live wallpaper behind applications that would like to sit on top of it.

```
public class MainActivity extends WallpaperService {
```

Three methods need to be implemented: **onCreate**, **onDestroy** and **onCreateEngine**. Not that the default implementation of **onCreate** needs to be deleted first, such as **onCreateOptionsMenu** that is no longer required by this new parent class. Override these three methods in your main activity and fill them up as follows.

```
@Override
public void onCreate() {
    super.onCreate();
}

@Override
public void onDestroy() {
    super.onDestroy();
}

@Override
public Engine onCreateEngine() {
    return new CircleEngine();
}
```

Only the main thread is able to draw on the screen, hence we will need to create a **Handler** object that will be called from inside the **CircleEngine** object.

```
        private final Handler mHandler = new Handler();
```

- Create of the **CircleEngine** Class

Create an inner class which extends the **Engine** Class. This class will be in charge of drawing the circle. Declare the following member variables into the **CircleEngine** class.

```
        private final Paint mPaint = new Paint();
        private float mTouchX = -1;
        private float mTouchY = -1;
        private boolean mVisible;

        private final Runnable mDrawCircle = new Runnable() {
               public void run() {
                      drawFrame();
               }
        };
```

The **mPaint** object holds the style and color information about how to draw shapes, text and bitmaps.

Variables **mTouchX** and **mTouchY** are used to get **X** and **Y** values on the screen, **mDrawCircle** is a **Runnable** object that will used to start the drawing.

- Fill up the constructor

Inside the **CircleEngine** constructor, you need to set the **mPaint** object

```
                final Paint paint = mPaint;
                paint.setColor(0xffffffff);
                paint.setAntiAlias(true);
                paint.setStrokeWidth(2);
                paint.setStrokeCap(Paint.Cap.ROUND);
                paint.setStyle(Paint.Style.STROKE);
```

- Create methods responsible for drawing the frame

We will need to instantiate **SurfaceHolder** class in order to display a surface. Then create a **Canvas** object which holds the shapes to draw. The canvas needs to be **locked** in order to start editing pixels on the surface via the **drawTouchPoint**() method. When you are done drawing objects, you need to unlock the **Canvas**. Implement the following **drawFrame()** and **drawTouchPoint()** functions in the **CircleEngine** class.

```
                void drawFrame() {
                final SurfaceHolder holder = getSurfaceHolder();

                Canvas c = null;
                try {
                       c = holder.lockCanvas();
                       if (c != null) {
                           // draw something
                           drawTouchPoint(c);
                       }
                   } finally {
                       if (c != null) holder.unlockCanvasAndPost(c);
                   }

                   // Reschedule the next redraw
                   mHandler.removeCallbacks(mDrawCircle);
                   if (mVisible) {
                       mHandler.postDelayed(mDrawCircle, 1000 / 50);
                   }
               }

                void drawTouchPoint(Canvas c) {
                      if (mTouchX >=0 && mTouchY >= 0) {
                            c.drawCircle(mTouchX, mTouchY, 80, mPaint);
                      }
                }
```

- Handle the touch events

Override the **OnTouchEvent** function inside the **CircleEngine** class. Implement it as follows to save the last touched position into our coordinates member variables.

```
@Override
public void onTouchEvent(MotionEvent event) {
  if (event.getAction() == MotionEvent.ACTION_MOVE) {
            mTouchX = event.getX();
            mTouchY = event.getY();

  } else if (event.getAction() == MotionEvent.ACTION_DOWN){
            mTouchX = event.getX();
            mTouchY = event.getY();

  } else if (event.getAction() == MotionEvent.ACTION_CANCEL){
            mTouchX = -1;
            mTouchY = -1;
  }
  super.onTouchEvent(event);
}
```

- Finalize the class implementation

Finish up the **CircleEngine** class by implementing the following functions.

```
@Override
public void onCreate(SurfaceHolder surfaceHolder) {
    super.onCreate(surfaceHolder);

    // By default we don't get touch events, so enable them.
    setTouchEventsEnabled(true);
}

@Override
public void onDestroy() {
    super.onDestroy();
    mHandler.removeCallbacks(mDrawCircle);
}

@Override
public void onVisibilityChanged(boolean visible) {
    mVisible = visible;
    if (visible) {
        drawFrame();
    } else {
        mHandler.removeCallbacks(mDrawCircle);
    }
}
```

```
        @Override
        public void onSurfaceChanged(SurfaceHolder holder, int format, int
width, int height) {
            super.onSurfaceChanged(holder, format, width, height);
            drawFrame();
        }

        @Override
        public void onSurfaceCreated(SurfaceHolder holder) {
            super.onSurfaceCreated(holder);
        }

        @Override
        public void onSurfaceDestroyed(SurfaceHolder holder) {
            super.onSurfaceDestroyed(holder);
            mVisible = false;
            mHandler.removeCallbacks(mDrawCircle);
        }

        @Override
        public void onOffsetsChanged(float xOffset, float yOffset,
                float xStep, float yStep, int xPixels, int yPixels) {
             super.onOffsetsChanged(xOffset, yOffset, xStep, yStep, xPixels,
yPixels);
        }
```

- Expose your wallpaper resource

Add an **Android XML file** named **cube.xml** in your project. It should only contain the following lines.

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

- Adapt the application's manifest file

Insert the following **Service** tag inside the **Application** tag in your manifest.

```
        <service
            android:label="@string/app_name"
            android:name="MainActivity"
            android:permission="android.permission.BIND_WALLPAPER">
            <intent-filter>
                <action
android:name="android.service.wallpaper.WallpaperService" />
            </intent-filter>
            <meta-data android:name="android.service.wallpaper"
android:resource="@layout/cube" />
        </service>
```

Modify the Activity tag to look like the following.

```
        <activity
             android:name=".MainActivity"
             android:label="@string/title_activity_main"
             android:theme="@android:style/Theme.Light.WallpaperSettings"
             android:exported="true">
         </activity>
```

Launch your application and see if it installs correctly. You should then be able to select your freshly installed custom wallpaper from the **Wallpaper** settings menu. Play around with the live wallpaper, and imagine what other nice graphics you could implement!

Lab 9: Create an application involving camera, face detection, audio and location

Goals

- Learn how to use the camera
- Learn about face detection
- Use Audio recording and playback
- Use location services

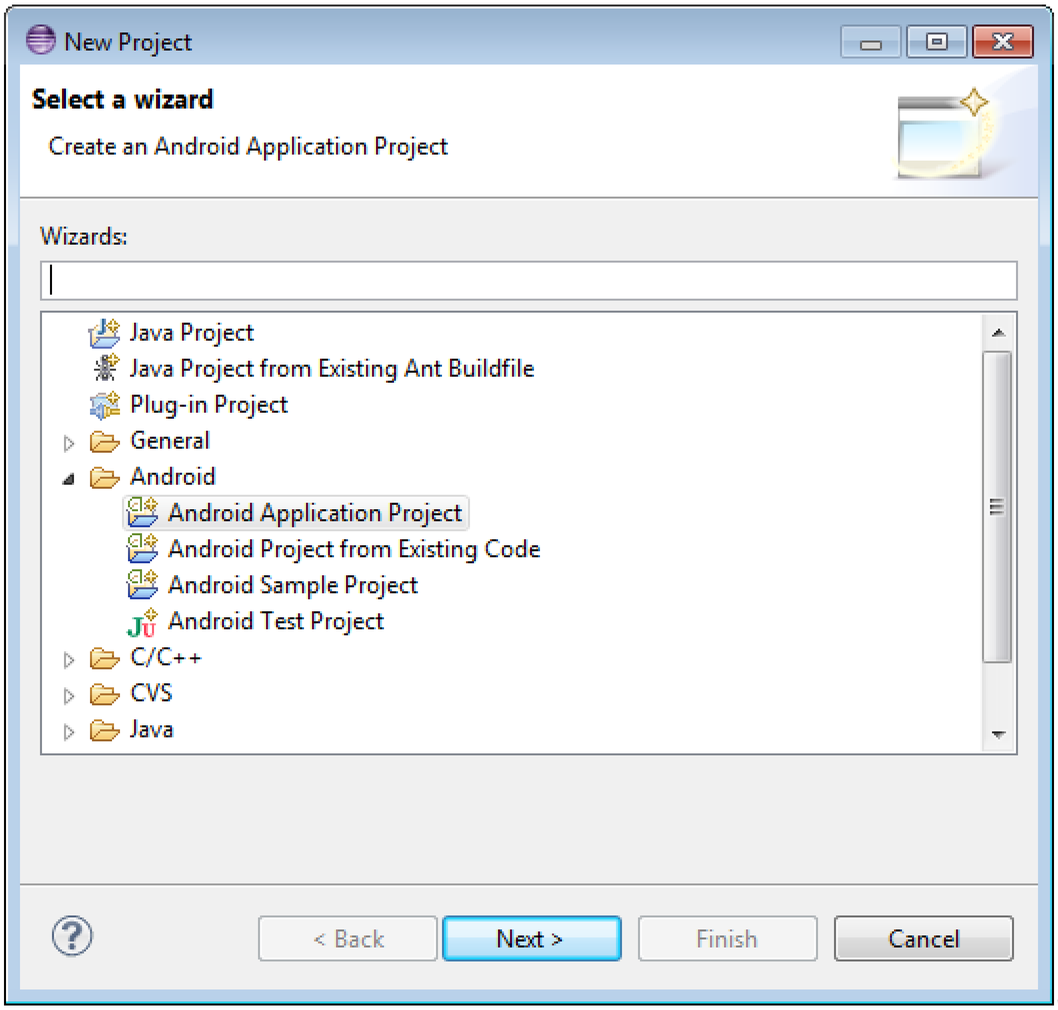
Estimated time: 90 minutes

Part 1: Create a new android application project

➢ **Creating the project**

1. Open the Eclipse IDE

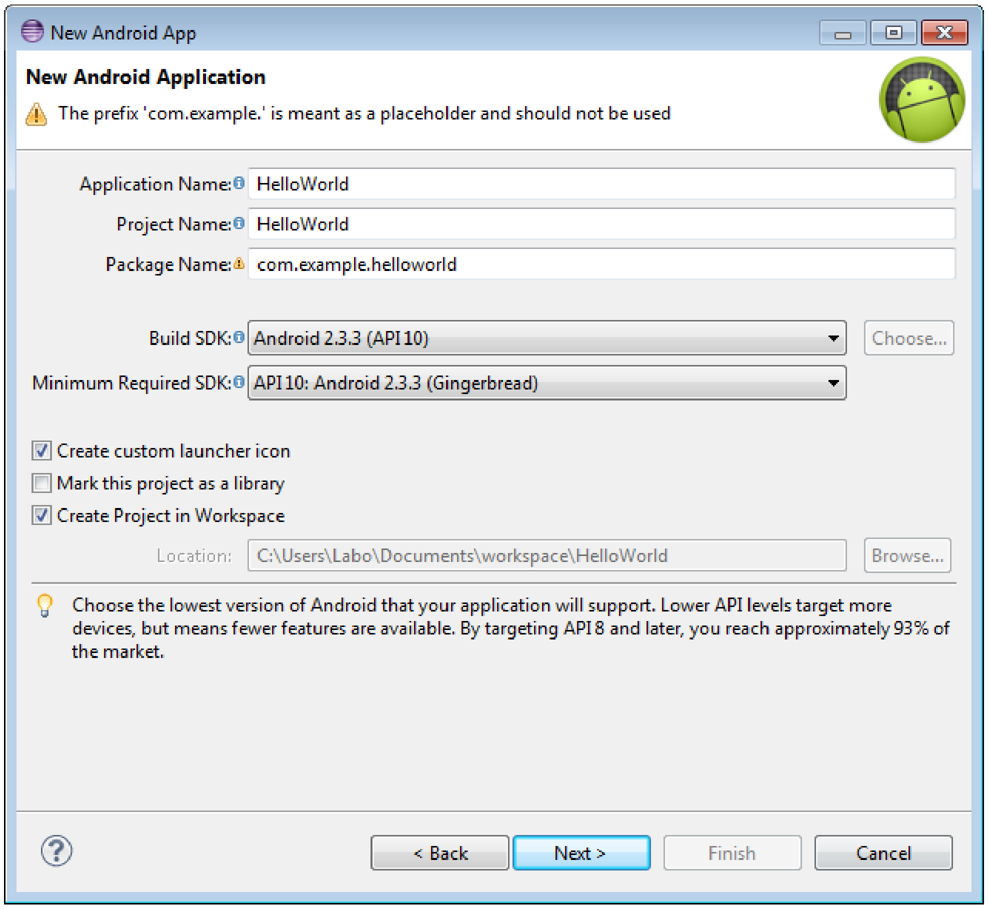
2. Select **File -> New -> New Project.** Select Android Application Project and click next.



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)

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

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.

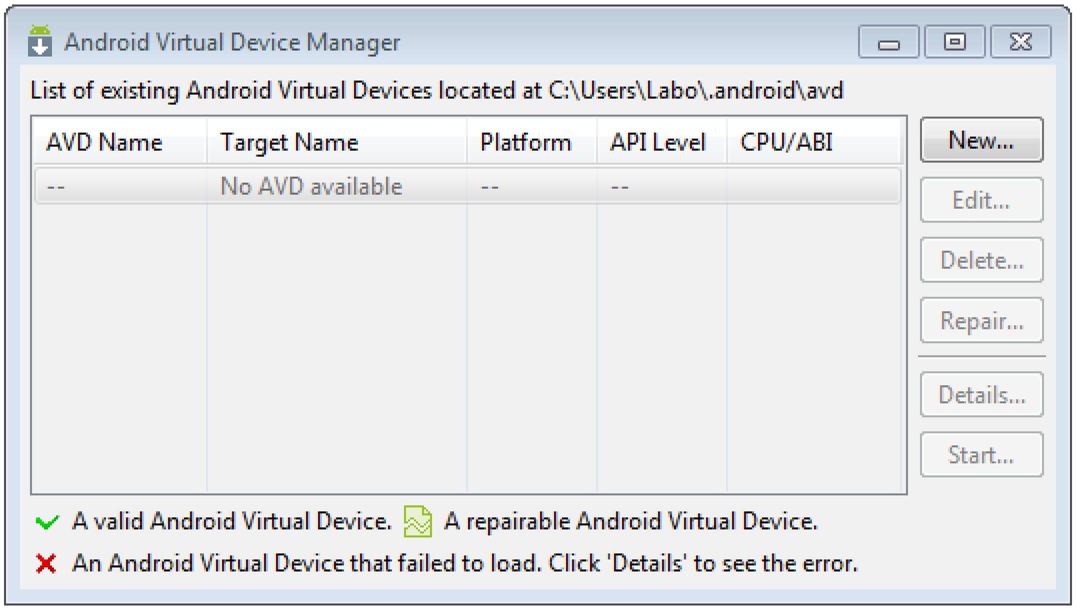
5. Now you can select an activity template from which to begin building your app.

   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**.



List of existing Android Virtual Devices located at C:\Users\Labo\.android\avd

AVD Name	Target Name	Platform	API Level	CPU/ABI
--	No AVD available	--	--	

✓ A valid Android Virtual Device. A repairable Android Virtual Device.

✗ An Android Virtual Device that failed to load. Click 'Details' to see the error.

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**.

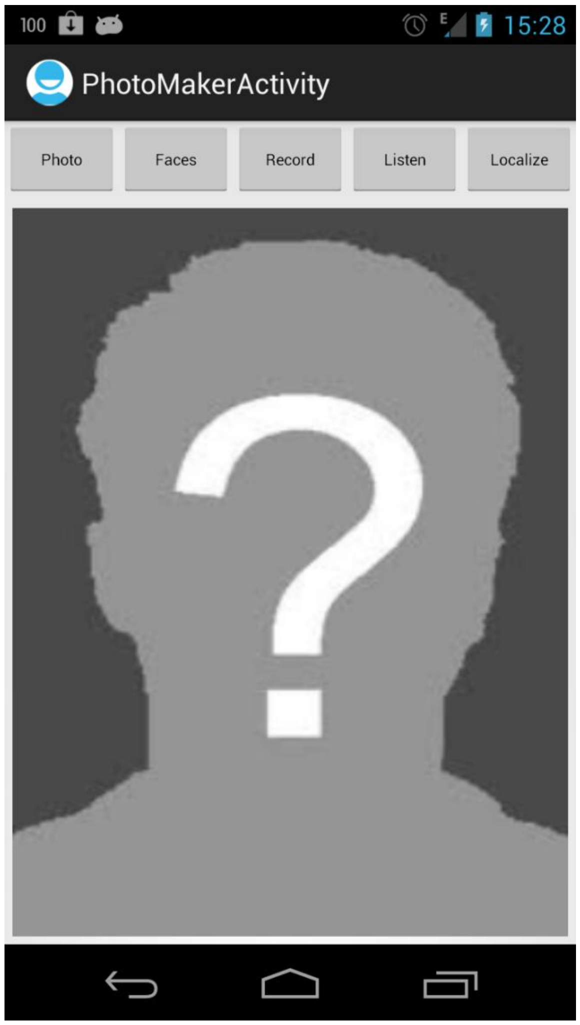
4. Select the new AVD from the Android Virtual Device Manager and click **Start**.

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:

- 5 Buttons for different functionalities, an **ImageView** to display picture back

Goal:

- Using hardware capabilities

- Create the main activity

Create a **PhotoMaker** application and add the following member variables to the project's main activity.

```
    private int PHOTO_CODE = 124578;
    private int NUMBER_OF_FACE = 3;

    private ImageView mImageViewPhoto;
    private Bitmap mBmpPhoto;
    private Uri mImageUri;
    private MediaRecorder mRecorder;
    private MediaPlayer  mPlayer;

    private static String mFileName;

    boolean mStartRecording = true;
    boolean mStartPLaying = true;

    private LocationManager locManager;
    private LocationListener locListener;

    private boolean gps_enabled = false;

    private boolean network_enabled = false;
```

- Prepare the layout

Open the main activity's layout and add 5 buttons (preferably in a **LinearLayout**) and an **ImageView** object.

```
    <LinearLayout
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:orientation="horizontal"
    android:weightSum="5"
    android:gravity="center_horizontal" >

        <Button
            android:id="@+id/btnTakePhoto"
            android:layout_width="wrap_content"
             android:layout_height="wrap_content"
             android:text="Photo"
             android:onClick="btnTakePhotoPressed"
             android:textSize="10sp"
             android:layout_weight="1"
      />

        <Button
            android:id="@+id/btnDetectFaces"
            android:layout_width="wrap_content"
             android:layout_height="wrap_content"
             android:text="Faces"
```

```xml
            android:textSize="10sp"
            android:onClick="btnDetectFacesPressed"
            android:visibility="visible"
            android:layout_weight="1"
    />

     <Button
           android:id="@+id/btnRecord"
           android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Record"
            android:textSize="10sp"
            android:onClick="btnRecordPressed"
            android:visibility="visible"
            android:layout_weight="1"
    />

     <Button
           android:id="@+id/btnListen"
           android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Listen"
            android:textSize="10sp"
            android:onClick="btnListenPressed"
            android:visibility="visible"
            android:layout_weight="1"
    />

     <Button
           android:id="@+id/btnLocalize"
           android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Localize"
            android:textSize="10sp"
            android:onClick="btnLocalizePressed"
            android:visibility="visible"
            android:layout_weight="1"
    />

</LinearLayout>

<ImageView
    android:id="@+id/imageViewPhoto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="5dp"
    android:src="@drawable/johndoe"
    android:scaleType="fitXY"

/>
```

- Add the default “johndoe” picture

Add a picture to your project inside the **drawable-hdpi** folder. This picture will be used as the default picture when the user has not taken a picture yet.

- Write code for taking pictures

Create an intent using **MediaStore.ACTION_IMAGE_CAPTURE** to take a picture and return control to the calling application. Create a temporary file that will hold the picture taken by the user for further processing. The temporary file’s Uri is passed to the intent in order to be filled up with the picture data. This is done by implementing the following functions.

```
    private File createTemporaryFile(String part, String ext) throws Exception
    {
        File tempDir= Environment.getExternalStorageDirectory();
        tempDir=new File(tempDir.getAbsolutePath()+"/.temp/");
        if(!tempDir.exists())
        {
            tempDir.mkdir();
        }
        return File.createTempFile(part, ext, tempDir);
    }

    private void dispatchTakePictureIntent(int actionCode) {
        Intent takePictureIntent = new
Intent(MediaStore.ACTION_IMAGE_CAPTURE);

        File photo = null;

        try
        {
            photo = this.createTemporaryFile("picture", ".png");
            photo.delete();
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        mImageUri = Uri.fromFile(photo);
        takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, mImageUri);

        startActivityForResult(takePictureIntent, actionCode);
      }
```

In order to receive a callback from the activity, override the protected method **onActivityResult**. In this method, we need to test if the result is OK and then fill up the **ImageView** object with the picture. We need to create a **bitmap** from the Uri, resizing the bitmap that can not exceed 2048x2048 pixels.

```
    public Bitmap getResizedBitmap(Bitmap bm, int newHeight, int newWidth) {
        int width = bm.getWidth();
        int height = bm.getHeight();
        float scaleWidth = ((float) newWidth) / width;
        float scaleHeight = ((float) newHeight) / height;
        Matrix matrix = new Matrix();
        matrix.postScale(scaleWidth, scaleHeight);
        Bitmap resizedBitmap = Bitmap.createBitmap(bm, 0, 0, width, height,
matrix, false);
        return resizedBitmap;
    }

    public void grabImage(ImageView imageView)
    {
        this.getContentResolver().notifyChange(mImageUri, null);
        ContentResolver cr = this.getContentResolver();
        Bitmap bitmap;
        try
        {
           bitmap = android.provider.MediaStore.Images.Media.getBitmap(cr,
mImageUri);

            mBmpPhoto = getResizedBitmap(bitmap,
(int)((float)(bitmap.getHeight()*(2f/3f))),
((int)(float)(bitmap.getWidth()*(2f/3f))));
            imageView.setImageBitmap(mBmpPhoto);
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent
data) {

      if (requestCode == PHOTO_CODE && resultCode == Activity.RESULT_OK)
            grabImage(mImageViewPhoto);

      super.onActivityResult(requestCode, resultCode, data);
    }

    public static boolean isIntentAvailable(Context context, String action) {
        final PackageManager packageManager = context.getPackageManager();
        final Intent intent = new Intent(action);
        List<ResolveInfo> list =
                packageManager.queryIntentActivities(intent,
PackageManager.MATCH_DEFAULT_ONLY);
        return list.size() > 0;
    }

    public void btnTakePhotoPressed(View v) {
      if ( isIntentAvailable(this,MediaStore.ACTION_IMAGE_CAPTURE))
            dispatchTakePictureIntent(PHOTO_CODE);
    }
```

- Use the face detector

We need to use **FaceDetector** API that provides a service to identify faces out of a Bitmap graphic object. We will then draw green rectangles around the detected faces. Implement the **btnDetectFacesPressed** function that will be called after a user click on the “Detect faces” button. Implement this function as follows.

```
    public void btnDetectFacesPressed(View v) {
       if (mBmpPhoto != null) {

                 Bitmap bmpTemp = mBmpPhoto.copy(Config.RGB_565, true);

               int imageWidth = bmpTemp.getWidth();
               int imageHeight = bmpTemp.getHeight();
               FaceDetector.Face[] faceArray = new
FaceDetector.Face[NUMBER_OF_FACE];
               FaceDetector faceDetector = new FaceDetector(imageWidth,
imageHeight, NUMBER_OF_FACE);
               int numberOfFaceDetected = faceDetector.findFaces(bmpTemp,
faceArray);

               Canvas canvas = new Canvas(mBmpPhoto);

                Paint myPaint = new Paint();
                myPaint.setColor(Color.GREEN);
                myPaint.setStyle(Paint.Style.STROKE);
                myPaint.setStrokeWidth(5);

                for(int i=0; i < numberOfFaceDetected; i++)
                {
                 Face face = faceArray[i];
                 PointF myMidPoint = new PointF();
                 face.getMidPoint(myMidPoint);
                 float myEyesDistance = face.eyesDistance();
                 canvas.drawRect(
                   (int)(myMidPoint.x - myEyesDistance),
                   (int)(myMidPoint.y - myEyesDistance),
                   (int)(myMidPoint.x + myEyesDistance),
                   (int)(myMidPoint.y + myEyesDistance),
                   myPaint);

                 mImageViewPhoto.invalidate();
                }
       }
    }
```

- Record an audio clip

Implement the **btnRecordPressed** function that will record audio upon user click on the **Record** button, then stop recording after a second press. This function relies on the associated **startRecording** and **stopRecording** functions.

```
   private void startRecording() {
        mRecorder = new MediaRecorder();
        mRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
        mRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
        mRecorder.setOutputFile(mFileName);
        mRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);

        try {
            mRecorder.prepare();
        } catch (IOException e) {
            e.printStackTrace();
        }

        mRecorder.start();
    }

    private void stopRecording() {
        mRecorder.stop();
        mRecorder.release();
        mRecorder = null;
    }

    private void onRecord(boolean start) {
        if (start) {
            startRecording();
        } else {
            stopRecording();
        }
    }

      public void btnRecordPressed(View v) {
             mFileName =
Environment.getExternalStorageDirectory().getAbsolutePath() +
"/audiorecordComment.3gp";

             onRecord(mStartRecording);
             if (mStartRecording)
                    Toast.makeText(this, "Start recording",
Toast.LENGTH_SHORT).show();
             else
                    Toast.makeText(this, "Stop recording",
Toast.LENGTH_SHORT).show();

             mStartRecording = !mStartRecording;
     }
```

- Playback the audio clip

We then need to implement the **btnListenPressed** function that is tied to the **Listen** button. This function simply plays back the last recorded audio clip using the **MediaPlayer** object.

```
    private void onPlay(boolean start) {
        if (start) {
            startPlaying();
        } else {
            stopPlaying();
        }
    }

    private void startPlaying() {
             if (mFileName != null) {
                   mPlayer = new MediaPlayer();
                   try {
                          mPlayer.setDataSource(mFileName);
                          mPlayer.prepare();
                          mPlayer.start();
                   } catch (IOException e) {
                          e.printStackTrace();
                   }
             }
    }

    private void stopPlaying() {
       if (mFileName != null) {
             mPlayer.release();
             mPlayer = null;
       }
    }

    public void btnListenPressed(View v) {
       onPlay(mStartPLaying);
        if (mStartPLaying) {
             Toast.makeText(this, "Start playing", Toast.LENGTH_SHORT).show();
        } else {
             Toast.makeText(this, "Stop playing", Toast.LENGTH_SHORT).show();
        }
        mStartPLaying = !mStartPLaying;
    }
```

- Get the current location

Create a **LocationManager** object which provides access to Android's **System Location Services**. Inside the **onCreate** method of the main activity, call the following function to reach the system location service.

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

        mImageViewPhoto = new ImageView(this);
        mImageViewPhoto = (ImageView) findViewById(R.id.imageViewPhoto);

        locManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);
    }
```

Create a new class that extends **LocationListener**. This interface is used for receiving notifications from the **LocationManager** when the location has changed. Four methods must be implemented but only one is needed for our purposes: **onLocationChanged**

```
    class MyLocationListener implements LocationListener {

       public void onLocationChanged(Location location) {
              if (location != null) {
                     locManager.removeUpdates(locListener);

                     Double lat = location.getLatitude();
                     Double lon = location.getLongitude();

                     Log.v("TAG","location finded : lat =
"+location.getLatitude() + " long = "+location.getLongitude());
                     Toast.makeText(getBaseContext(), "current location : Lon =
"+location.getLongitude() + "Lat = "+location.getLatitude(),
Toast.LENGTH_LONG).show();
              }
       }

       public void onProviderDisabled(String provider) {
              // TODO Auto-generated method stub
       }

       public void onProviderEnabled(String provider) {
              // TODO Auto-generated method stub
       }

       public void onStatusChanged(String provider, int status, Bundle extras)
{
              // TODO Auto-generated method stub
       }
    }
```

Implement the function to start the localization service whenever the user presses the associated button.

```
    public void btnLocalizePressed(View v) {
      if (mBmpPhoto != null) {

            locListener = new MyLocationListener();

            try {
                  gps_enabled =
locManager.isProviderEnabled(LocationManager.GPS_PROVIDER);
                  } catch (Exception e) {
                        e.printStackTrace();
                  }
                  try {
                  network_enabled =
locManager.isProviderEnabled(LocationManager.NETWORK_PROVIDER);
                  } catch (Exception e) {
                        e.printStackTrace();
                  }

                  if (!gps_enabled && !network_enabled) {
                        Toast.makeText(this, "Unable to get current
location, try to start your GPS", Toast.LENGTH_LONG);
                        startActivity(new
Intent(android.provider.Settings.ACTION_LOCATION_SOURCE_SETTINGS));
                        return;
                  }

                  if (gps_enabled) {

      locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,
locListener);
                  }
                  else if (network_enabled) {

      locManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0,
0, locListener);
                  }

      }
    }
```

- Finalize the manifest file

Open your application’s manifest file to add the needed permissions and usage feature information.

```
    <uses-feature android:name="android.hardware.camera" />
    <uses-feature android:name="android.hardware.camera.autofocus" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Launch the application on the device and play around. Image all the cool applications you could write using those ready-to-use components!