Изображение в SQLite

У нас будет использоваться файловый кэш + mem cache (LruCache).  
Начнём с файла приложения.

| **Java**[Выделить код](http://www.cyberforum.ru/) | |
| --- | --- |
| |  | | --- | | **package** com.example.dbImageExample;    **import** android.app.Application;  **import** android.graphics.Bitmap;  **import** android.util.Log;  **import** android.util.LruCache;    **import** java.io.ByteArrayOutputStream;    **public** **class** TheApplication **extends** Application{  **private** **static** TheApplication instance;  **private** **static** LruCache<**String**, Bitmap> mMemoryCache;    **public** **static** TheApplication getInstance() {  **return** instance;      }        @**Override**  **public** **void** onCreate() {  **super**.onCreate();          instance = **this**;          initMemoryCache();      }    **public** **static** LruCache<**String**, Bitmap> getMemoryCache() {  **return** mMemoryCache;      }    **private** **void** initMemoryCache() {  **final** **int** kilo = 1024;  **int** maxMemory = (**int**) **Runtime**.getRuntime().maxMemory() / kilo;  **int** cacheSize = maxMemory / 8;          mMemoryCache = **new** LruCache<**String**, Bitmap>(cacheSize) {              @**Override**  **protected** **int** sizeOf(**String** key, Bitmap bitmap) {  **ByteArrayOutputStream** bao = **new** **ByteArrayOutputStream**();                  bitmap.compress(Bitmap.CompressFormat.JPEG, 100, bao);  **int** size = bao.size();  **try** {                      bao.close();                  } **catch** (**Exception** e) {   Log.e(getClass().getSimpleName(), "Unable to init memoryCache", e);                  }  **return** size / kilo;              }          };      }  } | |  |

Мы используем кэш в памяти для быстрого доступа к изображениям. Чтобы он работал добавляем в тэг application в манифесте следующее:

| **XML**[Выделить код](http://www.cyberforum.ru/) | |
| --- | --- |
| |  | | --- | | **<application** android:name=".TheApplication" | |  |

Также нам потребуется доступ к файловой системе и интернету, значит запрашиваем persmission в манифесте:

| **Java**[Выделить код](http://www.cyberforum.ru/) | |
| --- | --- |
| |  | | --- | | <uses-permission android:name="android.permission.INTERNET"/>      <uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" /> | |  |

Опишем хелпер-класс, который сможет декодировать изображения, а также работать с кэшем.

| **Java**[Выделить код](http://www.cyberforum.ru/) | |
| --- | --- |
| |  | | --- | | **package** com.example.dbImageExample;    **import** android.graphics.Bitmap;  **import** android.graphics.BitmapFactory;  **import** android.util.DisplayMetrics;  **import** android.util.Log;    **import** java.io.File;  **import** java.io.IOException;  **import** java.lang.ref.SoftReference;    **public** **class** FileUtils {  **public** **static** **final** **String** TEMP\_FOLDER = "temp/";    **public** **static** **int** getDimensions(**boolean** isWidth) {          DisplayMetrics m = TheApplication.getInstance().getResources().getDisplayMetrics();  **return** isWidth ? m.widthPixels : m.heightPixels;      }    **private** **static** **int** calculateInSampleSize(BitmapFactory.Options options, **int** reqWidth, **int** reqHeight, **boolean** canBeCropped) {  *// Raw height and width of image*  **final** **int** height = options.outHeight;  **final** **int** width = options.outWidth;  **int** inSampleSize = 1;    **if** (height > reqHeight || width > reqWidth) {  *// Calculate ratios of height and width to requested height and width*  **int** imageMaxSize = reqHeight > reqWidth ? reqHeight : reqWidth;    **if** (height > imageMaxSize || width > imageMaxSize) {                  inSampleSize = (**int**) **Math**.pow(2, (**int**) **Math**.round(**Math**.log(imageMaxSize /                          (**double**) **Math**.max(height, width)) / **Math**.log(0.5)));              }          }  **return** inSampleSize;      }    **private** **static** **int** calculateInSampleSizeOnWidth(BitmapFactory.Options options, **int** reqWidth, **boolean** canBeCropped) {  **return** calculateInSampleSize(options, reqWidth, reqWidth, canBeCropped);      }    **public** **static** Bitmap decodeUri(**String** filePath, **int** reqWidth, **int** reqHeight, **boolean** canBeCropped) {  *// First decode with inJustDecodeBounds=true to check dimensions*          BitmapFactory.Options options = **new** BitmapFactory.Options();          options.inJustDecodeBounds = **true**;          BitmapFactory.decodeFile(filePath, options);    *// Calculate inSampleSize*          options.inSampleSize = calculateInSampleSize(options, reqWidth, reqHeight, canBeCropped);  *// Decode bitmap with inSampleSize set*          options.inJustDecodeBounds = **false**;          options.inPurgeable = **true**;  **SoftReference**<Bitmap> bsoft = **new** **SoftReference**<Bitmap>(BitmapFactory.decodeFile(filePath, options));  **return** bsoft.get();      }      **public** **static** Bitmap decodeUriResizeOnWidth(**String** filePath, **int** reqWidth, **boolean** canBeCropped) {  *// First decode with inJustDecodeBounds=true to check dimensions*          BitmapFactory.Options options = **new** BitmapFactory.Options();          options.inJustDecodeBounds = **true**;          BitmapFactory.decodeFile(filePath, options);    *// Calculate inSampleSize*          options.inSampleSize = calculateInSampleSizeOnWidth(options, reqWidth, canBeCropped);  *// Decode bitmap with inSampleSize set*          options.inJustDecodeBounds = **false**;          options.inPurgeable = **true**;  **SoftReference**<Bitmap> bsoft = **new** **SoftReference**<Bitmap>(BitmapFactory.decodeFile(filePath, options));  **return** bsoft.get();      }      **public** **static** Bitmap decodeUriResizeOnWidth(**String** filePath, **int** reqWidth) {  **return** decodeUriResizeOnWidth(                  filePath,                  reqWidth,  **false**          );      }    **public** **static** Bitmap decodeUri(**String** filePath, **int** reqWidth, **int** reqHeight) {  **return** decodeUri(                  filePath,                  reqWidth,                  reqHeight,  **false**          );      }    **public** **static** Bitmap decodeUri(**String** filePath) {  **return** decodeUri(                  filePath,                  getDimensions(**true**),                  getDimensions(**false**)          );      }    **public** **static** Bitmap decodeUri(**String** filePath, **boolean** canBeCropped) {  **return** decodeUri(                  filePath,                  getDimensions(**true**),                  getDimensions(**false**),                  canBeCropped          );      }    **private** **static** **String** getStorageDir() {  *//PackageInfo pi = application.getPackageManager().getPackageInfo(application.getPackageName(), PackageInfo.PARCELABLE\_WRITE\_RETURN\_VALUE);*  **File** dir = TheApplication.getInstance().getCacheDir();*//new File(pi.applicationInfo.dataDir);*  **String** storageDirPath = dir.getAbsolutePath() + "/";  **if** (!dir.exists()) dir.mkdirs();          requestFilePermission(dir.getAbsolutePath());            dir = **new** **File**(storageDirPath + TEMP\_FOLDER);  **if** (!dir.exists()) dir.mkdirs();          requestFilePermission(dir.getAbsolutePath());            dir = **new** **File**(storageDirPath + TEMP\_FOLDER);  **if** (!dir.exists()) dir.mkdirs();          requestFilePermission(dir.getAbsolutePath());  **return** storageDirPath;      }      **public** **static** **void** requestFilePermission(**String** filePath) {  **if** (filePath != **null** && !filePath.trim().isEmpty()) {  *//solution for 2.3.3. problem with reading cache dir in file system*  **try** {  **Runtime**.getRuntime().exec("chmod 755 " + filePath);              } **catch** (**IOException** e) {                  Log.e("Utils.getStorageDir", e.getMessage(), e);              }          }      }    **public** **static** **String** getFilePath(**String** fileName) {  **return** getStorageDir() + fileName;      }      */\*------------------mem\_cache---------------\*/*    **public** **static** **synchronized** **void** addBitmapToMemoryCache(**String** key, Bitmap bitmap) {  **if** (bitmap != **null** && key != **null** && !key.trim().isEmpty() && getBitmapFromMemoryCache(key) == **null** && !bitmap.isRecycled()) {              TheApplication.getMemoryCache().put(key, bitmap);          }      }    **public** **static** Bitmap getBitmapFromMemoryCache(**String** key) {  **if** (key != **null** && !key.trim().isEmpty()) {              Bitmap b = TheApplication.getMemoryCache().get(key);  **if** (b != **null** && b.isRecycled()) {                  removeBitmapFromMemoryCache(key);              } **else** {  **return** b;              }          }  **return** **null**;      }    **public** **static** Bitmap removeBitmapFromMemoryCache(**String** key) {  **try** {  **if** (key != **null** && !key.trim().isEmpty()) {  **return** TheApplication.getMemoryCache().remove(key);              }          } **catch** (**IllegalStateException** e) {              Log.e("FileUtils", "Utils.removeBitmapFromMemoryCache", e);          }  **return** **null**;      }    } | |  |

Android откровеено хреново работает с bitmap, и чтобы декодировать bitmap необходимо его ужимать.  
  
  
Приступим к базе данных. Опишем хэлпер синглтон класс:

| **Java**[Выделить код](http://www.cyberforum.ru/) | |
| --- | --- |
| |  | | --- | | **package** com.example.dbImageExample.database;    **import** android.content.Context;  **import** android.database.Cursor;  **import** android.database.sqlite.SQLiteDatabase;  **import** android.database.sqlite.SQLiteOpenHelper;  **import** com.example.dbImageExample.TheApplication;  **import** com.example.dbImageExample.database.entity.ImageObject;  **import** com.example.dbImageExample.database.utils.ImageTable;    **import** java.util.LinkedList;  **import** java.util.List;    **public** **class** DBHelper **extends** SQLiteOpenHelper {    **public** **static** **final** **int** DB\_VERSION = 1;  **public** **static** **final** **String** DB\_NAME = "ImageExampleDB";  **private** **static** DBHelper instance;  **public** **static** **final** **String** CREATE\_TABLE\_PREFIX = "CREATE TABLE IF NOT EXISTS ";    **public** **static** DBHelper getInstance() {  **if** (instance == **null**) {              instance = **new** DBHelper();          }  **return** instance;      }    **public** DBHelper() {  **super**(TheApplication.getInstance().getApplicationContext(), DB\_NAME, **null**, DB\_VERSION);      }    **private** **void** initDataTables() {          getWritableDatabase().rawQuery(CREATE\_TABLE\_PREFIX + ImageTable.TABLE\_NAME + " ("         + ImageTable.IMAGE\_ID + ", " + ImageTable.IMAGE\_PATH + ");", **null**);      }    **public** ImageObject getImageObjectById(**int** id) {          ImageObject model = **null**;  **String** sql = "SELECT \* FROM " + ImageTable.TABLE\_NAME + " WHERE " + ImageTable.IMAGE\_ID + " = " + id;  **Cursor** cursor = getWritableDatabase().rawQuery(sql, **null**);  **if** (cursor != **null** && cursor.moveToFirst()) {              model = ImageTable.parseCursor(cursor);          }          closeCursor(cursor);  **return** model;      }    **public** **List**<ImageObject> getImageObjects() {  **List**<ImageObject> imageObjects = **new** **LinkedList**<ImageObject>();  **String** sql = "SELECT \* FROM " + ImageTable.TABLE\_NAME;  **Cursor** cursor = getWritableDatabase().rawQuery(sql, **null**);  **if** (cursor != **null** && cursor.moveToFirst()) {  **do** {                  imageObjects.add(ImageTable.parseCursor(cursor));              } **while** (cursor.moveToNext());          }          closeCursor(cursor);  **return** imageObjects;      }    **public** **long** saveImageObject(ImageObject imageObject) {  **String** sql = "SELECT \* FROM " + ImageTable.TABLE\_NAME + " WHERE " + ImageTable.IMAGE\_ID + " = " + imageObject.getId();  **Cursor** cursor = getWritableDatabase().rawQuery(sql, **null**);  **long** pointer;  **if** (cursor == **null** || !cursor.moveToFirst()) {  *//insert new raw*              pointer = getWritableDatabase().insertWithOnConflict(ImageTable.TABLE\_NAME, **null**, ImageTable.getCV(imageObject), SQLiteDatabase.CONFLICT\_REPLACE);          } **else** {  *//update existing*              pointer = getWritableDatabase().updateWithOnConflict(ImageTable.TABLE\_NAME, ImageTable.getCV(imageObject), ImageTable.IMAGE\_ID + " = " + imageObject.getId(), **null**, SQLiteDatabase.CONFLICT\_REPLACE);          }          closeCursor(cursor);  **return** pointer;      }    **private** **void** closeCursor (**Cursor** cursor) {  **if** (cursor != **null** && !cursor.isClosed()) {              cursor.close();          }      }        @**Override**  **public** **void** onCreate(SQLiteDatabase db) {          initDataTables();      }        @**Override**  **public** **void** onUpgrade(SQLiteDatabase db, **int** oldVersion, **int** newVersion) {      } | |  |

Сущность с которой мы будем работать:

| **Java**[Выделить код](http://www.cyberforum.ru/) | |
| --- | --- |
| |  | | --- | | **package** com.example.dbImageExample.database.entity;    **public** **class** ImageObject {  **private** **int** id;  **private** **String** path;    **public** ImageObject(**int** id, **String** path) {  **this**.id = id;  **this**.path = path;      }    **public** ImageObject() {      }    **public** **int** getId() {  **return** id;      }    **public** **void** setId(**int** id) {  **this**.id = id;      }    **public** **String** getPath() {  **return** path;      }    **public** **void** setPath(**String** path) {  **this**.path = path;      }  } | |  |

и утилита, которая нам будет помогать парсить cursor и создавать contentValues для работы с БД

| **Java**[Выделить код](http://www.cyberforum.ru/) | |
| --- | --- |
| |  | | --- | | **package** com.example.dbImageExample.database.utils;    **import** android.content.ContentValues;  **import** android.database.Cursor;  **import** com.example.dbImageExample.database.entity.ImageObject;    **public** **class** ImageTable {  **public** **static** **final** **String** TABLE\_NAME = "image\_table";  **public** **static** **final** **String** IMAGE\_ID = "image\_id";  **public** **static** **final** **String** IMAGE\_PATH = "image\_path";    **public** **static** ImageObject parseCursor(**Cursor** cursor) {          ImageObject model = **new** ImageObject();          model.setPath(cursor.getString(cursor.getColumnIndex(IMAGE\_PATH)));          model.setId(cursor.getInt(cursor.getColumnIndex(IMAGE\_ID)));  **return** model;      }    **public** **static** ContentValues getCV(ImageObject imageObject) {          ContentValues contentValues = **new** ContentValues();          contentValues.put(IMAGE\_ID, imageObject.getId());          contentValues.put(IMAGE\_PATH, imageObject.getPath());  **return** contentValues;      }    } | |  |

Идём дальше. Мы будем скачивать картинку из интернета, если её у нас нет, но перед этим создадим ещё один класс.

| **Java**[Выделить код](http://www.cyberforum.ru/) | |
| --- | --- |
| |  | | --- | | **package** com.example.dbImageExample.remote;    **import** org.apache.http.conn.ClientConnectionManager;  **import** org.apache.http.conn.scheme.PlainSocketFactory;  **import** org.apache.http.conn.scheme.Scheme;  **import** org.apache.http.conn.ssl.SSLSocketFactory;  **import** org.apache.http.impl.client.DefaultHttpClient;  **import** org.apache.http.impl.conn.tsccm.ThreadSafeClientConnManager;  **import** org.apache.http.params.HttpConnectionParams;  **import** org.apache.http.params.HttpParams;    **public** **class** HttpClientFactory {    **public** **static** **final** **int** SOCKET\_TIMEOUT = 25000;  **public** **static** **final** **int** CONNECTION\_TIMEOUT = 15000;  **public** **static** **final** **String** HTTPS = "https";  **public** **static** **final** **String** HTTP = "http";  **public** **static** **final** **int** PORT80 = 80;  **public** **static** **final** **int** PORT443 = 443;  **private** **static** DefaultHttpClient client;    **public** **synchronized** **static** DefaultHttpClient getThreadSafeClient() {  **if** (client != **null**) {  **return** client;          }          client = **new** DefaultHttpClient();          ClientConnectionManager mgr = client.getConnectionManager();          mgr.getSchemeRegistry().register(**new** Scheme(HTTP, PlainSocketFactory.getSocketFactory(), PORT80));          mgr.getSchemeRegistry().register(**new** Scheme(HTTPS, **SSLSocketFactory**.getSocketFactory(), PORT443));          HttpParams params = client.getParams();          HttpConnectionParams.setSoTimeout(params, SOCKET\_TIMEOUT);          HttpConnectionParams.setConnectionTimeout(params, CONNECTION\_TIMEOUT);          client = **new** DefaultHttpClient(**new** ThreadSafeClientConnManager(params, mgr.getSchemeRegistry()), params);  **return** client;      }  } | |  |

Фактически это ни разу не factory, а singleton, но пусть будет так)  
Класс, который будет загружать и кэшировать изображения:

| **Java**[Выделить код](http://www.cyberforum.ru/) | |
| --- | --- |
| |  | | --- | | **package** com.example.dbImageExample.remote;    **import** android.graphics.Bitmap;  **import** android.graphics.BitmapFactory;  **import** android.util.Log;  **import** com.example.dbImageExample.FileUtils;  **import** org.apache.http.HttpEntity;  **import** org.apache.http.client.HttpClient;  **import** org.apache.http.client.methods.HttpGet;  **import** org.apache.http.entity.BufferedHttpEntity;    **import** java.io.File;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  **import** java.io.InputStream;  **import** java.lang.ref.WeakReference;  **import** java.net.URI;    **public** **class** DownloadModule {  **public** **static** **final** **String** TAG = DownloadModule.**class**.getSimpleName();    **public** Bitmap downloadImage(**String** url, **boolean** deleteIfExists) {  **if** (url == **null**) {              Log.e(TAG, "Warning! Url is null", **new** **IllegalArgumentException**());  **return** **null**;          }    **WeakReference**<Bitmap> reference = **new** **WeakReference**<Bitmap>(FileUtils.getBitmapFromMemoryCache(url));  **if** (reference.get() != **null**) { *//Try to find it in the mem-cache*  **if** (!reference.get().isRecycled()) {  **return** reference.get();              } **else** {                  FileUtils.removeBitmapFromMemoryCache(url);              }          }      **URI** uri;  **try** {              uri = **URI**.create(url);          } **catch** (**IllegalArgumentException** e) {              Log.e(TAG, "Warning! Can't decode URI", **new** **IllegalArgumentException**());  **return** **null**;          }  **String**[] split = uri.getPath().split("/");  **if** (split.length < 2) {              Log.e(TAG, "Warning! Incorrect uri? " + uri, **new** **IllegalArgumentException**());  **return** **null**;          }  **String** filePath = split[split.length - 2] + "\_" + split[split.length - 1];  **File** file = **new** **File**(FileUtils.getFilePath(filePath));  **if** (file.exists()) {  **if** (deleteIfExists) {  *//noinspection ResultOfMethodCallIgnored*                  file.delete();              } **else** {                  reference = **new** **WeakReference**<Bitmap>(FileUtils.decodeUri(FileUtils.getFilePath(filePath)));  **if** (reference.get() == **null**) {  *//Something goes wrong*  *//noinspection ResultOfMethodCallIgnored*                      file.delete();                  } **else** {  **return** reference.get();                  }              }          }          reference = processDownload(url, filePath);  **return** reference != **null** ? reference.get() : **null**;      }    **private** **WeakReference**<Bitmap> processDownload(**String** uri, **String** filePath) {          HttpClient client = HttpClientFactory.getThreadSafeClient();          HttpGet httpGet = **new** HttpGet(uri);          HttpEntity httpEntity;  **WeakReference**<Bitmap> reference = **null**;  **try** {  **FileOutputStream** stream = **new** **FileOutputStream**(**new** **File**(FileUtils.getFilePath(filePath)));              httpEntity = client.execute(httpGet).getEntity();              BufferedHttpEntity bufferedHttpEntity = **new** BufferedHttpEntity(httpEntity);  **InputStream** is = bufferedHttpEntity.getContent();              Bitmap bitmap = BitmapFactory.decodeStream(is);              bitmap.compress(Bitmap.CompressFormat.PNG, 100, stream);              reference = **new** **WeakReference**<Bitmap>(bitmap);              is.close();              stream.close();              FileUtils.addBitmapToMemoryCache(filePath, bitmap);          } **catch** (**IOException** e) {              Log.e(TAG, "Unable to execute request", e);  **return** reference;          }  **return** reference;      }  } | |  |

Теперь осталось всё это связать.   
Работа с базой и сетью должна быть асинхронная, а асинхронность предполагает методы обратного вызова.  
  
Опишем generic интерфейс для возврата результата.

| **Java**[Выделить код](http://www.cyberforum.ru/) | |
| --- | --- |
| |  | | --- | | **package** com.example.dbImageExample.remote;    **public** **interface** IDataCallback<T> {  **void** onSuccess(T data);    **void** onError(**String** reason);  } | |  |

Асинхронный абстрактный обработчик:

| **Java**[Выделить код](http://www.cyberforum.ru/) | |
| --- | --- |
| |  | | --- | | **package** com.example.dbImageExample.remote;    **import** android.os.AsyncTask;    **public** **abstract** **class** AbsAsyncTask<T> **extends** AsyncTask<**Void**, **Void**, T> {  **private** IDataCallback<T> callback;  **private** **Throwable** throwable;    **protected** AbsAsyncTask(IDataCallback<T> callback) {  **this**.callback = callback;      }        @**Override**  **protected** T doInBackground(**Void**... params) {  **try** {  **return** doAction();          } **catch** (**Throwable** throwable) {  **this**.throwable = throwable;  **return** **null**;          }      }        @**Override**  **protected** **void** onPostExecute(T t) {  **super**.onPostExecute(t);  **if** (callback != **null**) {  **if** (t != **null**) {                  callback.onSuccess(t);              } **else** **if** (throwable != **null**) {                  callback.onError(throwable.getMessage());              } **else** {                  callback.onError("Something goes wrong!");              }          }      }    **protected** **abstract** T doAction() **throws** **Throwable**;    } | |  |

И за всё у нас будет отвечать менеджер

| **Java**[Выделить код](http://www.cyberforum.ru/) | |
| --- | --- |
| |  | | --- | | **package** com.example.dbImageExample.remote;    **import** android.graphics.Bitmap;  **import** com.example.dbImageExample.FileUtils;  **import** com.example.dbImageExample.database.DBHelper;  **import** com.example.dbImageExample.database.entity.ImageObject;    **import** java.io.File;    **public** **class** RemoteManager {  **private** **static** RemoteManager instance;    **public** **static** RemoteManager getInstance() {  **if** (instance == **null**) {              instance = **new** RemoteManager();          }  **return** instance;      }    **public** **void** getImageById(IDataCallback<Bitmap> callback, **final** **int** id, **final** **String** serverUrl) {  **new** AbsAsyncTask<Bitmap>(callback) {              @**Override**  **protected** Bitmap doAction() **throws** **Throwable** {                  ImageObject imageObjectById = DBHelper.getInstance().getImageObjectById(id);    **if** (imageObjectById != **null** && imageObjectById.getPath() != **null** && !imageObjectById.getPath().isEmpty()) {                      Bitmap bitmapFromMemoryCache = FileUtils.getBitmapFromMemoryCache(imageObjectById.getPath());  **if** (bitmapFromMemoryCache == **null**) {  **String** path = imageObjectById.getPath();  **File** file = **new** **File**(FileUtils.getFilePath(path));  **if** (file.exists() && file.canRead()) {                              Bitmap bitmap = FileUtils.decodeUri(path);                              FileUtils.addBitmapToMemoryCache(path, bitmap);  **return** bitmap;                          } **else** {  **return** getBitmap();                            }                      } **else** {  **return** bitmapFromMemoryCache;                      }                  } **else** {  **return** getBitmap();                  }              }    **private** Bitmap getBitmap() {  *//Download file*  **String**[] split = serverUrl.split("/");  **String** newFilePath = split[split.length - 1];                  Bitmap bitmap = **new** DownloadModule().downloadImage(serverUrl, **true**);  **if** (bitmap != **null**) {                      FileUtils.addBitmapToMemoryCache(newFilePath, bitmap);                      DBHelper.getInstance().saveImageObject(**new** ImageObject(id, newFilePath));                  }  **return** bitmap;              }          }.execute();      }    } | |  |

Финал. Использование:

| **Java**[Выделить код](http://www.cyberforum.ru/) | |
| --- | --- |
| |  | | --- | | **public** **class** MyActivity **extends** Activity {  ***/\*\****  ***\* Called when the activity is first created.***  ***\*/***      @**Override**  **public** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);          setContentView(R.layout.main);          RemoteManager.getInstance().getImageById(**new** IDataCallback<Bitmap>() {              @**Override**  **public** **void** onSuccess(Bitmap data) {                  Toast.makeText(MyActivity.**this**, "Complete!", Toast.LENGTH\_LONG).show();              }                @**Override**  **public** **void** onError(**String** reason) {                  Toast.makeText(MyActivity.**this**, "Oops!**\n**" + reason, Toast.LENGTH\_LONG).show();              }          }, 1, "http://someHost.domain/static/1.jpg");      }  } | |  |

P.S. можно было бы сделать и проще