Issue 8488: 'bitmap size exceeds VM budget' if Activity is restarted [includes test demo!]

303 people starred this issue.
Comments by non-members will not trigger notification emails to users who starred this issue.

Back to list

Status: Declined
Owner: ----
Closed: May 2010

Type-Defect
Priority-Medium
ReportedBy-Developer

Add a comment below

Reported by andreas....@googlemail.com, May 22, 2010

```
Note: This is NOT a requst for assistance!

In my applications, I keep getting the following exception:

E/AndroidRuntime( 1420): java.lang.OutOfMemoryError: bitmap size exceeds VM
budget
E/AndroidRuntime( 1420):        at
android.graphics.BitmapFactory.nativeDecodeStream(Native Method)
E/AndroidRuntime( 1420):        at
android.graphics.BitmapFactory.decodeStream(BitmapFactory.java:459)
E/AndroidRuntime( 1420):        at
android.graphics.BitmapFactory.decodeStream(BitmapFactory.java:515)
E/AndroidRuntime( 1420):        at
de.schildbach.bitmapbug.MyActivity.bitmap(MyActivity.java:38)
E/AndroidRuntime( 1420):        at
de.schildbach.bitmapbug.MyActivity.onCreate(MyActivity.java:23)
E/AndroidRuntime( 1420):        at
android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1047)
E/AndroidRuntime( 1420):        at
android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2459)
E/AndroidRuntime( 1420):        at
android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2512)
E/AndroidRuntime( 1420):        at
android.app.ActivityThread.handleRelaunchActivity(ActivityThread.java:3625)
E/AndroidRuntime( 1420):        at
android.app.ActivityThread.access$2300(ActivityThread.java:119)
E/AndroidRuntime( 1420):        at
android.app.ActivityThread$H.handleMessage(ActivityThread.java:1867)
E/AndroidRuntime( 1420):        at
android.os.Handler.dispatchMessage(Handler.java:99)
E/AndroidRuntime( 1420):        at android.os.Looper.loop(Looper.java:123)
E/AndroidRuntime( 1420):        at
android.app.ActivityThread.main(ActivityThread.java:4363)
E/AndroidRuntime( 1420):        at java.lang.reflect.Method.invokeNative(Native
Method)
E/AndroidRuntime( 1420):        at java.lang.reflect.Method.invoke(Method.java:521)
E/AndroidRuntime( 1420):        at
com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:860)
E/AndroidRuntime( 1420):        at
com.android.internal.os.ZygoteInit.main(ZygoteInit.java:618)
E/AndroidRuntime( 1420):        at dalvik.system.NativeStart.main(Native Method)

I've boiled this down to a simple Testcase, contained in an Eclipse
project. Deploy to your device, watch the log and run. A bitmap of about 4
MB in size is extracted 10 times. The activity always overwrites the same
reference, so this is no problem for the 24 MB of heap the app has.

Now, switch the orientation of your device. The activity is being
restarted. Again, the bitmap is extracted 10 times. But this time, it
chokes at number 4. It always chokes at number 4, regardless of how many
extractions were successful in the first activity instance.

This is on a Nexus One with 2.1-update1.
```

bitmap-bug.tar.gz
795 KB Download

#1 romaingu...@gtempaccount.com May 23, 201

Your app needs to use less memory.

Status: Declined

#2 andreas....@googlemail.com May 24, 201

Hi Romainguy, I don't understand your comment.

My app uses only 4 MB of memory at a time, how can this be too much for 16 MB of heap?

How much less memory does my app have to use in order to fit 4 MB into 16 MB?

#3 dag.kris...@gmail.com Jun 11, 201

Agreed. All my apps suffer from this problem. There seems to be a bug in the garbage collection o bitmaps somewhere. If not, please post an example of how to properly assign and dereference bitmaps.

#4 swinefea...@gmail.com Aug 18, 201

why was this issue closed without any solution or guidance from the google team?

#5 scott.st...@gmail.com Aug 18, 201

I would also like a solution to this by the Google team as I am experiencing the same problem. I have tried scaling images to limit memory consumption, but still same problem.
Help Google !

#6 swinefea...@gmail.com Aug 18, 201

i found a workaround - one that at least works for my particular case.

at the end of my onPause(), i force a garbage collection like so:

```
  @Override
  public void onPause()
  {
    super.onPause();

    // yada yada yada...

    System.gc();
  }
```

and then in onDestroy() i null all of my member variables and do the same thing:

```
  public void onDestroy()
  {
    super.onDestroy();

    m_timerHandler.removeCallbacks(m_updateTimeTask);
    m_timerHandler = null;
    m_someWeirdInstanceThatDoesStuff = null;

    System.gc();
  }
```

hope this helps!

#7 iwan.mem...@gmail.com Aug 20, 201

This is a very weird response from the team to say the least.
The app is obviously not out of all possible memory, it only uses a portion.

If we were allocating arrays of bytes, this would not happen, so obviously it's a Bitmap issue.
I'm afraid whoever declined this bug did not read it well.

#8 iwan.mem...@gmail.com Aug 20, 201

As a workaround, I do Bitmap.recycle() for every bitmap, then null references and call GC manually. Seems to work, but stinks, especially since the docs say recycle() is not really necessary to call..

#9 22dec2012@gmail.com Aug 21, 201

Hi everybody,

I have a solution of this ... please checkout it here...I have tried to explain it in detail, inputs are welcomed...

check it here...
http://mobi-solutions.blogspot.com/2010/08/how-to-if-you-want-to-create-and.html

- Sandeep Choudhary
http://mobi-solutions.blogspot.com

#10 andreas....@googlemail.com Aug 21, 201

@22dec2012: I'm sorry to say that your suggestion does not solve this problem. It also happens with Android managed bitmaps (e.g. images referenced in XML files) and you won't get hold of thei references easily and of course you can't null foreign references either.

btw. Your blog does not accept comments.

#11 andreas....@googlemail.com Aug 21, 201

@22dec2012: Also I am not quite happy with your explaination of the cause of the problem (memory allocation done "fast than" garbage collection). How comes that in my example the loop will run endlessly on the first invocatation of the Activity? The bug only surfaces only when Activity switches occur.

Did you try to slow down the example with some Thread.sleep()s to confirm your theory? The garbag collector should be able to keep up in this case, if your theory is right.

#12 22dec2012@gmail.com Aug 21, 201

@Andreas : i am not aware about your other foreign references.
Whatever i have explained in blog is to solve a problem when we allocate & deallocate bitmaps frequently (as you referred in your source code attached here 'bitmap-bug.tar.gz')

And my solution works every time if you try to start the activity first time or any n number of time restart (changed orientation or any other) it or how fast you do alloc or dealloc.

My blog accepts comments!

- Sandeep Choudhary
http://mobi-solutions.blogspot.com

#13 iwan.mem...@gmail.com Aug 21, 201

Agree with Andreas (partially).
We are not expected to be dependent on the speed of GC cleaning up objects.
When memory is lacking, GC should explicitly clear everything that it can before throwing out of memory.
Now, Sandeep is right in terms of his solution to the problem, which is more important :)

#14 swinefea...@gmail.com Aug 21, 201

Sandeep - how does your solution apply to oom exception occurring due to using drawables to skin views in an activity? If i follow your approach, should I be calling recycle() on every view that has a drawable in onPause()?

Thanks

#15 22dec2012@gmail.com Aug 22, 201

@iwan: If we don't call recycle on the bitmap then it is GC's responsibility to free that, but in actually the timing for automatically call of GC to do this is uncertain and that make things worst. It is not bitmap's or any other object's responsibility to say hey i am done now they call or trigger GC. in other words it is not a bitmap or any other object's bug but...we can say that it is a GC optimization algorithm's bug while CPU is fully in use(that for loop) and missed the right timing ... and that is too late for that app. so as a partial responsibility we should call recycle or gc in our code, in that case recycle is best to reduce the side effects of the GC.

@swinefeaster: can you share your code sample, so i can help in that. you can post here or send m
@ 22dec2012@gmail.com

#16 andreas....@googlemail.com Aug 26, 201

I just modified the testcase a bit:

1) Made Froyo compatible. The image was too big to fit one instance of it into memory in Froyo. I scaled it a bit smaller.

2) Added sleep of two seconds between each image instantiation. The GC, which runs on a separate thread, should now have enough time to run. Being "too fast" is no excuse any more. On my Nexus One, the same crash happens at the same loop iteration of the second activity life.

Note that I do _not_ call System.gc() or Bitmap.recycle(), as this should not be needed as per documentation. Of course, you could do all your memory management on your own. I appreciate that the described workaround is possible in some(!) situations. However, the bug is present and shoul at least be confirmed.

bitmap-bug2.tar.gz
594 KB Download

#17 cyt...@gmail.com Aug 31, 201

I think the issue was, VM should not crash when out of memory. If it's a catchable exception, it's acceptable.

#18 blahblah676@gmail.com Sep 8, 201

We had this problem too. Solution is twofold:

[1] Ensure you null all references to the bitmap and any Canvas objects that the bitmap is selected into. Do this in your onDestroy and/or at the point you don't need the bitmap any more. If there are still any references to the bitmap, the bitmap will never be freed - no many how man times you call gc! Also, Bitmap.recycle() seems to do absolutely nothing, so don't rely on it.

[2] Put a System.gc() at the end of your onDestroy. For some reason this is not done automaticall in onDestroy, and if you don't force a gc() then the memory usage tends to just keep increasing each time you restart the activity (I guess garbage collection doesn't happen very often).

Romain asserts that this isn't a bug, but I disagree. In my opinion there are 2 android bugs here First, Android should automatically do a gc() after destroying the activity. I mean, apart from bitmaps there could be a whole host of other "stuff" that the activity has allocated during its lifetime and that needs to be garbage collected before starting up a new instance of the activity It shouldn't be up to the developer to do this. Second, Bitmap.recycle() really should do what it says - at the moment it seems to do absolutely nothing as far as I can tell.

#19 rehan.va...@gmail.com Sep 9, 201

I was struggling with this same problem yesterday and I found another "solution". Well actually I came across it while looking at a one finger zoom tutorial over at:

http://blogs.sonyericsson.com/developerworld/2010/05/26/android-one-finger-zoom-tutorial-part-2/

I realise that this solution won't apply to all situations but I thought I would add it anyway. While I was storing my (large) image in the res/drawable directory, mBitmap.recycle() would not work and one would have to call System.gc() as mentioned in the previous posts. When the I stored

my image in res/drawable-nodpi (as they have done in the Sony Ericsson tutorial) mBitmap.recycle(would actually work and I was able to continuously open and close my application without it crashing.

#20 iwan.mem...@gmail.com Sep 9, 201

Rehan's solution sounds interesting keeping in mind that I am also facing this issue when dealing with generated Bitmaps rather than ones loaded from resources.

#21 pimd...@gmail.com Sep 13, 201

I tested andreas' project in te first post on three different devices: SE X10 pro mini (1.6), HTC Wildfire (2.1) and Nexus One (2.2). The app runs fine on the X10 but gets an out of memory error on the Wildfire and the Nexus One.

#22 pimd...@gmail.com Sep 13, 201

Also tested on HTC Desire (2.1), OOM error.

#23 blahblah676@gmail.com Sep 13, 201

Also see ~~issue 11089~~ for a test case that should trigger the bug every time. Note that sometime you need to change screen orientation a number of times - it depends how big the bitmap is and ho much memory your device allows to be allocated (16 or 24MB).

#24 kf6nvr Sep 29, 201

I can easily run in to this problem by setting an ImageVIew to a URI returned by the Gallery imag picker. From the logs, I can see that it's the Bitmap causing the problem during the internal BitmapFactory call that the ImageView performs.

Hopefully the suggestions of forcing a gc() will help. In my case, this happens when bouncing between the picker and my own Activity. Resizing the bitmap in advance would only slow down the occurrence, not remove it completely.

#25 yoge...@gmail.com Oct 27, 201

At least exception it throws should be catchable so we can do the needful before activity Closes/Crashes.

#26 kf6nvr Oct 27, 201

@yogengg: From what I've seen, and others on this thread, it's not catchable: the entire VM is terminated. That's part of the issue.

I think this issue needs to be opened again since this particular one is declined and, thus, it's likely being ignored even with 20+ comments since...

#27 bigairso...@gmail.com Oct 28, 201

I have the same issue!!! This does need to be reopened. I create a bitmap in onCreate of my activity, and recycle it immediately. I can pop out of the activity and restart a few times before getting an out of memory. I have tried nulling, GCing, and more with no success.

#28 blahblah676@gmail.com Oct 28, 201

sam: check out my comment 18 above. As long as you null all references to the bitmap AND the canvas that it is selected into, gc() should free up the memory. If you don't null the canvas the gc() will not free the bitmap memory because the canvas still has an internal reference to it.

#29 swinefea...@gmail.com Dec 11, 201

This is a huge bug and should be fixed. Though I think even if they fix it in 2.3, we we still be stuck working around it in the meantime.

Maybe I'm crazy but IMHO:

[1] A garbage collected system should ALWAYS garbage collection before throwing an out of memory

exception.

[2] There should be an OutOfMemoryException so that we can catch it and call System.gc() :)

I will try Comment #18 again as one of my list activities is crashing the app this way...

Cheers,
swine

#30 swinefea...@gmail.com Dec 11, 201

Comment #18 doesn't apply to me because I am loading images from resource inside ListView items. These are created by my list adapter derived class.

--> Is there any way of detecting that a list item should be destroyed so that I can unhook callbacks and such? finalize() won't be it because the garbage collector won't call it until are "references are nulled"...

Thanks for any insight!

S:

#31 blahblah676@gmail.com Dec 11, 201

You will be notified in getView() that one of your list's views has gone off-screen (the convertView parameter).

#32 swinefea...@gmail.com Dec 11, 201

@blahblah676 - thanks but that wont work because it wont get called when the list view is being destroyed. I am using convertView already for reusing the view.

Or am I supposed to iterate through the adapter and manually tell all views to destroy themselves

If I set the number of items in the datamodel to 0 and then do a datasetChanged() I still wantsa get notified to destroytye view (set its callbacks to null)

#33 swinefea...@gmail.com Dec 12, 201

So I tried creating a fake datamodel that holds 40 data items that don't need bitmaps. I switched the adapter to this datamodel in the activitie's onPause() routine. This invokes the getView() as blahblah676 mentioned and seems to properly release the bitmaps after System.gc() is called.

Despite this being an ugly hack, one problem is that the list activity refreshes and shows the 40 blank list items before it transitions away, which detracts from the user experience.

I tried adding another hack to this by using a runnable that I post 100 ms after the onPause() like so:

```
  private void swapInFakeHackModelForDestruction()
  {
    if(m_listView != null)
    {
      // Do it a bit later.
      Runnable later = new Runnable()
      {
        @Override
        public void run()
        {
          if(m_adapter != null && m_context != null)
          {
            PickLeafModel fakeModel =
              PickLeafModel.createFakeHackModelForDestruction(m_context);
            m_adapter.setModel(fakeModel);
          }

          System.gc();
        }
      };
      m_listView.postDelayed(later, 100);
    }
  }
```

```
This fixes the refresh problem because it happens once the activity is in the background. However
putting the 100ms delay there causes all sorts of other issues such as the one below. Not 100%
sure I fixed the issue though... :(


12-11 23:38:42.568: ERROR/AndroidRuntime(4261): FATAL EXCEPTION: main
12-11 23:38:42.568: ERROR/AndroidRuntime(4261): android.view.InflateException: Binary XML file
line #11: Error inflating class <unknown>
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.view.LayoutInflater.createView(LayoutInflater.java:513)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
com.android.internal.policy.impl.PhoneLayoutInflater.onCreateView(PhoneLayoutInflater.java:56)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.view.LayoutInflater.createViewFromTag(LayoutInflater.java:563)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.view.LayoutInflater.rInflate(LayoutInflater.java:618)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.view.LayoutInflater.inflate(LayoutInflater.java:407)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.view.LayoutInflater.inflate(LayoutInflater.java:320)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at xxx.PickLeafCell.<init>
(PickLeafCell.java:38)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
xxx.CueFileListAdapter.getView(CueFileListAdapter.java:108)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.widget.HeaderViewListAdapter.getView(HeaderViewListAdapter.java:220)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.widget.AbsListView.obtainView(AbsListView.java:1315)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.widget.ListView.makeAndAddView(ListView.java:1727)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.widget.ListView.fillSpecific(ListView.java:1272)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.widget.ListView.layoutChildren(ListView.java:1558)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.widget.AbsListView.onLayout(AbsListView.java:1147)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at android.view.View.layout(View.java:7035)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.widget.RelativeLayout.onLayout(RelativeLayout.java:909)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at android.view.View.layout(View.java:7035)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.widget.FrameLayout.onLayout(FrameLayout.java:333)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at android.view.View.layout(View.java:7035)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.widget.LinearLayout.setChildFrame(LinearLayout.java:1249)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.widget.LinearLayout.layoutVertical(LinearLayout.java:1125)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.widget.LinearLayout.onLayout(LinearLayout.java:1042)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at android.view.View.layout(View.java:7035)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.widget.FrameLayout.onLayout(FrameLayout.java:333)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at android.view.View.layout(View.java:7035)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.view.ViewRoot.performTraversals(ViewRoot.java:1045)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.view.ViewRoot.handleMessage(ViewRoot.java:1727)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.os.Handler.dispatchMessage(Handler.java:99)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at android.os.Looper.loop(Looper.java:123)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.app.ActivityThread.main(ActivityThread.java:4627)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
java.lang.reflect.Method.invokeNative(Native Method)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
java.lang.reflect.Method.invoke(Method.java:521)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:858)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
com.android.internal.os.ZygoteInit.main(ZygoteInit.java:616)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at dalvik.system.NativeStart.main(Native
Method)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261): Caused by:
java.lang.reflect.InvocationTargetException
```

```
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at android.widget.ImageView.<init>
(ImageView.java:108)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
java.lang.reflect.Constructor.constructNative(Native Method)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
java.lang.reflect.Constructor.newInstance(Constructor.java:446)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.view.LayoutInflater.createView(LayoutInflater.java:500)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     ... 34 more
12-11 23:38:42.568: ERROR/AndroidRuntime(4261): Caused by: java.lang.OutOfMemoryError: bitmap siz
exceeds VM budget
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at android.graphics.Bitmap.nativeCreate(Nativ
Method)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.graphics.Bitmap.createBitmap(Bitmap.java:468)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.graphics.Bitmap.createBitmap(Bitmap.java:435)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.graphics.Bitmap.createScaledBitmap(Bitmap.java:340)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.graphics.BitmapFactory.finishDecode(BitmapFactory.java:488)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.graphics.BitmapFactory.decodeStream(BitmapFactory.java:462)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.graphics.BitmapFactory.decodeResourceStream(BitmapFactory.java:323)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.graphics.drawable.Drawable.createFromResourceStream(Drawable.java:697)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.content.res.Resources.loadDrawable(Resources.java:1709)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at
android.content.res.TypedArray.getDrawable(TypedArray.java:601)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     at android.widget.ImageView.<init>
(ImageView.java:118)
12-11 23:38:42.568: ERROR/AndroidRuntime(4261):     ... 38 more
```

#34 blahblah676@gmail.com Dec 12, 201

I'm not sure I fully understand what you are doing without seeing your code, but here are a few general points:

- You never need to destroy a view - that will happen automatically
- Your listview (and all its views) will automatically get garbage collected as long as you don't have any references pointing to it.
- Sometimes memory can get leaked and will be missed by the garbage collector if you have two objects pointing to each other, e.g. object A has a reference to object B and object B has a ref to object A. Sometimes the reference can be inside the Android code, e.g. if you have a Bitmap selected into a Canvas then the Canvas will have an internal reference to the Bitmap.

If you are finding that your ListView items aren't being garbage collected then I think you eithe need to check your custom View object for circular references and remove/redesign them, or maybe just ditch ListView altogether and roll your own solution.

#35 swinefea...@gmail.com Dec 12, 201

Thanks for the clarifications... I think I made some progress with this.

Question - if I have some object that has a reference to the context (which is really just the parent activity), does its context reference also need to be nulled before the activity is paused so that it can get garbage collected properly?

#36 blahblah676@gmail.com Dec 12, 201

Its only an issue if your activity has a ref to the object (or to another object which itself has a ref to it i.e. a circular dependency). Just google java memory leaks for more info. I think mos java programmers dont even realise its possible to leak memory until they start using android.

#37 swinefea...@gmail.com Dec 12, 201

I'm coming from the C++ world so I'm frustrated that I can't just call delete! Haha.

Thanks for the explanation. So in my case:

MyActivity -> MyAdapter, ListView -> MyCellView -> Context (which is the activity)

So then I do need to go back and null all the context references in onPause()?

I will do some more research as well.

Thanks

#38 brstr...@gmail.com Dec 20, 201

@blahblah676 comment 34: As far as I know, Java garbage collection has handled cyclic references since at least version 1 (1995 or so). Those forms of (some would say deficient) garbage collection which don't handle cyclic references are things like reference counting, as used in, say, perl(*).

Or perhaps am I misunderstanding; are you saying that the Dalvik vm specifically has deficient gc

(*) Aside from the final perl vm shutdown.

#39 blahblah676@gmail.com Dec 21, 201

Yes, you're right - my mistake, circular refs aren't a problem. The problem is that "The Canvas>Bitmap>pixels chain takes several GC cycles to be collected because of the finalizers" (according to Romain Guy) so nulling any Canvas/Bitmap references in your onDestroy and calling gc() just speeds up the process.

#40 swinefea...@gmail.com Dec 22, 201

Greetings! I have greatly improved my memory issues (and hopefully fixed the problem) in my app, so I would like to share with you some of the things that I have done. I hope you may find these useful.

1. If you use any threads, and they get passed Context variables which are really a reference to your activity, make sure you convert them into the application context so that they do not refer to the activity and thus leak it. An example of a constructor:

```
  public MyBelovedThread(Context context)
  {
    m_context = context.getApplicationContext();
  }
```

...So after doing the above, and of course putting System.gc() in places like onPause(), onDestroy(), onResume(), and null references to callbacks etc, I still had the same issue, even though it happened much more rarely. I investigated further:

2. To easily get the active native memory usage in your app (meaning bitmap allocations), use the following code. It puts the memory usage into your activity's title bar.

```
  int usedMegs = (int)(Debug.getNativeHeapAllocatedSize() / 1048576L);
  String usedMegsString = String.format(" - Memory Used: %d MB", usedMegs);
  getWindow().setTitle(usedMegsString);
```

3. Now that I had the memory info easily visible, I could see that just showing my list activity (one where each list item had an icon bitmap) ate up 15 megs of native memory! I removed the setting of the icon label in code, the memory went down to 7 megs, and I realized that, interestingly enough, each item had a default bitmap that I had set in the xml like so:

```
  <ImageView
    ...
    android:src="@drawable/some_default_icon_that_we_dont_see_cause_it_gets_changed_later"
    />
```

I removed the line in the xml, and the memory still stayed at 7 megs. Then I readded back the cod that set the icon to the correct bitmap drawable and THE MEMORY STILL SHOWED 7 MEGS!

Observation: Since I was inflating each list item from xml, the default ImageView bitmap used for the icon, even though I was immediately setting it to something different, was doubling my memory usage!

Conclusion: Don't leave any "android:src="@drawable/xxx" in your xmls if you do not need that particular drawable.

That's it for now. Hope this helps someone!

Cheers,
swine

#41 joseph.isaac Jan 6, 201

Wow, swine, I am running into this issue now and can't believe I'll have to swap my xml files to Java for UI components. Seems really inefficient for separation of UI from app logic, but I've found no solution to it. Thanks for posting.

#42 swinefea...@gmail.com Jan 6, 201

@joseph.isaac - No no no! That's not what I meant. I had the bitmap referenced in xml as a "default" and was setting it dynamically to something else during creating in code. This was resulting in it being loaded twice. I'm sure that you are ok if you have static images via xml an then don't change them.

sorry for the confusion.

swine

#43 joseph.isaac Jan 7, 201

Hey Swine,

Actually what I'm doing is I have an xml where the background image of a FrameLayout is a png. Inside that FrameLayout is a LinearLayout with an ImageView with its src attribute set to a different png. This is my "base UI component" in XML.

In my base activity, I setContentView() to that "base UI component". Now, this base activity is extended by say "loginActivity". LoginActivity programmatically adds more views/components to th base UI component.

The problem is, when I kick off the application, the loginActivity is created and the UI/app load fine. If I press the back button and go back to the home screen and then immediately launch the app again, no problem. It is the third time where I press the back button and then attempt to launch the app again that I get a runtime exception where the XML is failing to inflate the "base UI component" view because the bitmaps/drawables associated with that view have been recycled (I force the recycle in the OnDestroy() method of the base activity).

Does that make sense? I've tried keeping a weak reference to the drawables, but still no luck.

#44 swinefea...@gmail.com Jan 7, 201

Joseph - I bet what's going on is that your onDestroy() is never called because the OS hasn't decided to destroy it yet. You may want to do the System.gc() in the onPause() method. If that doesn't work you could also try setting the drawable to null in the onPause() before you do the System.gc(), but then you will have to set it back in the onResume() of course.

Hope this helps,

Swine

#47 kamui.ji...@gmail.com Jan 27, 201

Just be sure to manually call recycle() for every bitmap you've created.
Worked for me.

#49 ilan.ca...@gmail.com Jan 28, 201

@blahblah76
I figured that one out already thx, I'll remove the previous comment because it has nothing to do with this discussion

#50 swinefea...@gmail.com Feb 2, 201

WOW I just discovered something that is really contrary to what I expected: If you create a simpl activity with an ImageView in it that references a png drawable through xml, whenever you switch orientations (between portrait and landscape), the memory footprint goes up slightly each time. The Debug.MemoryInfo.getTotalPss() and getTotalSharedDirty()!!!!

After some trial and error, I found the following code makes this problem go away completely:

```
  @Override
  protected void onDestroy()
  {
    super.onDestroy();

    ImageView view = (ImageView)findViewById(R.id.whatever_your_image_view_id);
    view.setImageDrawable(null);
  }
```

So it seems that to fix memory issues, I am going to have to manually set all my drawables to null! Even those that I just create in xml and don't touch. This is a pretty bad bug. :(

Looks like Android 2.3 has concurrent garbage collection. I wonder if this will make this bug manifest less frequently.

#51 swinefea...@gmail.com Feb 2, 201

Here's a drop-in solution if anyone is interested. Just derive from BetterActivity instead of Activity. This will clean up all the drawables on orientation change, and likely on activity switches etc...

```
public abstract class BetterActivity extends Activity
{
  @Override
  protected void onResume()
  {
    System.gc();
    super.onResume();
  }

  @Override
  protected void onPause()
  {
    super.onPause();
    System.gc();
  }

  @Override
  public void setContentView(int layoutResID)
  {
    ViewGroup mainView = (ViewGroup)
      LayoutInflater.from(this).inflate(layoutResID, null);

    setContentView(mainView);
  }

  @Override
  public void setContentView(View view)
  {
    super.setContentView(view);

    m_contentView = (ViewGroup)view;
  }

  @Override
  public void setContentView(View view, LayoutParams params)
  {
    super.setContentView(view, params);

    m_contentView = (ViewGroup)view;
  }

  @Override
  protected void onDestroy()
  {
    super.onDestroy();

    // Fixes android memory ~~issue 8488~~:
    // http://code.google.com/p/android/issues/detail?id=8488
```

```
    nullViewDrawablesRecursive(m_contentView);

    m_contentView = null;
    System.gc();
  }

  private void nullViewDrawablesRecursive(View view)
  {
    if(view != null)
    {
      try
      {
        ViewGroup viewGroup = (ViewGroup)view;

        int childCount = viewGroup.getChildCount();
        for(int index = 0; index < childCount; index++)
        {
          View child = viewGroup.getChildAt(index);
          nullViewDrawablesRecursive(child);
        }
      }
      catch(Exception e)
      {
      }

      nullViewDrawable(view);
    }
  }

  private void nullViewDrawable(View view)
  {
    try
    {
      view.setBackgroundDrawable(null);
    }
    catch(Exception e)
    {
    }

    try
    {
      ImageView imageView = (ImageView)view;
      imageView.setImageDrawable(null);
      imageView.setBackgroundDrawable(null);
    }
    catch(Exception e)
    {
    }
  }

  // The top level content view.
  private ViewGroup m_contentView = null;
}
```

#52 blahblah676@gmail.com Feb 2, 201

It's not actually leaking - it just takes Android a few gc cycles to actually free the bitmaps, and Android doesn't do any garbage collection when destroying your activity. Putting a gc() and nulling any references into your onDestroy speeds up this process, but probably isn't necessary unless you are using large bitmaps.

#53 swinefea...@gmail.com Feb 2, 201

@blabla676 - what constitutes "a few"? time or some other unit? also, what constitutes "large bitmaps"?

to further this experiment, i put three 64x64 bitmaps on my empty text.xml activity, derived from BetterActivity as defined above.

1. switching back and forth between landscape and portrait repeatedly with the BetterActivity cod showed stable mem usage, as expected.

2. commenting out all the System.gc() calls *still* shows stable mem usage

3. commenting out the call to nullViewDrawablesRecursive() in onDestroy() and switching repeatedl
caused the mem to keep growing. after doing this 50 or so times, i observed the following:

a) my hand got tired
b) the memory kept increasing slowly
c) as i type up this email, it's been sitting there for over 5 minutes and still showing high mem
usage (not even a scent of gc in sight!)
d) i bet if i was a robot and did this exercise another 200 times i could get it to crash, just
like the crashes reported above.

i know it seems crazy. but you should try it if you don't believe me! btw i am running this on a
nexus1 with 2.2.2.

cheers,
swine

#54 blahblah676@gmail.com Feb 2, 201

I have no idea what 'several' gc cycles constitutes - you'd need to ask google or look at the
code. See ~~issue 11089~~.

Regarding your scenario above, if you put the gc() back in but take out your nulling, does the
memory still keep increasing? My guess is that Android simply hasn't bothered doing a gc() yet (a
it is very expensive).

My definition of a 'large' bitmap is anything more than a few MB.

#55 swinefea...@gmail.com Feb 2, 201

Putting back the gc() but taking out the nulling. Result: memory usage increases and never
decreases. So it seems you need both and unless someone finds a better solution, my app will be
shipping with the above code.

I looked at ~~issue 11089~~. Wow I am really losing respect for this Romain Guy guy. He must be
really in denial if he things there is no issue with the Android GC.

#56 patrick....@gmail.com Feb 5, 201

For me, the real fix was handling the orientation change myself and not letting the system destro
the activity.

So in your layout XML add android:configChanges="orientation"

```
<activity android:label="@string/app_name" android:configChanges="orientation"
        android:name=".your.package">
```

Then in your activity override onConfigurationChanged, Then set the contentview again.

```
//--------
@Override
onConfigurationChanged(Configuration config)
{
    setContentView(layoutResID);
    //get references again to buttons/etc
    //do your onCreat/onResume stuff
}
```

//-----------
It will use the layout-land version or layout version as appropriate, and all the memory tricks
become less of an issue.

I honestly think this is a bug in the Android system and NOT memorey leaks in the apps (though
maybe a mem leak in the system).

An app dev could null out all your references until his/her eyes bleed, but something else is
holding a reference in the api that devs have no control over.

And while Romain Guy is an *excellent* engineer, his declining of this bug and blog post on the memory management were not helpful, and further confused the matter -- I hope the Android engineers reconsider looking into this. But please dont get me wrong they are superstars doing awesome work, I just think they may want to double check this bug.

#58 newapp...@gmail.com Feb 5, 201

I'm getting Bitmap exceeds VM budget but only on the emulator not on my device. I tried to handle the config change my self but it only force closed in portrait mode.

#59 thorins...@gmail.com Feb 17, 201

I have been looking at exactly this issue for many weeks, without any hope of resolving it. My questions are:

1) What is the VM Bitmap Budget? Can it be changed? Can we influence it for applications that nee to absolutely display large bitmaps, or many bitmaps?

2) Why does bitmap.recycle() not give back the required memory so that further allocations of bitmaps can succeed? In my case I have an application that allocates ONE large bitmap and several smaller ones that are blitted onto that large bitmap. The problem is when my activity is closed, even if I do a bitmap.recycle() eventually by entering the activity and allocating a bitmap, I have been able to exceed the VM bitmap budget. Why?

3) Is there any way we can get statistics from the framework to tell us what the currently allocated bytes of bitmaps are so we might be able to manage this more closely?

Please, Google, spend some time correctly answering this question. It is quickly encountered by every single developer writing Android applications. Give us some answers that will definitively help us with this situation instead of quick dismissals.

Thanks in advance.

#60 8enm...@gmail.com Feb 17, 201

1) The Bitmap budget seems to be phone-dependent. On my Droid X it's about 25 MB, whereas stock android seems to be 16 (Froyo). Pretty sure it can't be changed.

2) After you call bitmap.recycle(), you should probably also set bitmap = null and then call System.gc(). This will ensure that the whole object gets garbage collected and isn't continuing t take up space. Make sure anything using that bitmap has also been nulled. Otherwise, the stuff may still be hanging around in memory. Recycle marks as recycled, doesn't call the garbage collector, presumably for performance reasons.

3) I don't think there's specifically a way to find out how much has been allocated for Bitmaps, but you can certainly find out how much memory your application is using and how much there is free, though it's a little confusing since it can grow the size on demand. Google it.

There is in fact a solution to this problem. The Java JVM is limited to a small number of MB, bu there is pretty much no limit on how much memory you can use in native code. If you're comfortable using JNI and C/C++, you should be doing all of the heavy lifting in there and only returning the final result to the JVM. If you're not, it's definitely something to look into. Native code is MUCH faster for data intensive operations like image processing.

For instance, let's say you have a large image, but only a small part of it will be displayed at given time on the screen. You could load the image into native memory and then return a pointer to only its relevant parts in your Java code.

#61 je...@apptitude.dk Feb 20, 201

I've run into this problem when using medium-size images (400x300-ish) in a Gallery. The problem exists because Gallery does not recycle views and the bitmap/external gc-algorithm seems not to b run when additional memory is required, as mentioned in previous posts.

A simple workaround that seems to resolve the problem for me when using ImageViews in a Gallery i to create a subclass of ImageView which nulls the drawable references and calls gc when the imageview is detached from the window.

```
public class SingleShotImageView extends ImageView {
```

```
        public SingleShotImageView(Context context) {
                super(context);
        }

        public SingleShotImageView(Context context, AttributeSet attrs) {
                super(context, attrs);
        }

        public SingleShotImageView(Context context, AttributeSet attrs, int defStyle) {
                super(context, attrs, defStyle);
        }

        @Override
        protected void onDetachedFromWindow () {
                setImageDrawable(null);
                setBackgroundDrawable(null);
                System.gc();
        }

}
```

#62 jj.Sarra...@gmail.com Feb 23, 201

It seems Honeycomb realease could fix our issue : http://developer.android.com/reference/android/graphics/BitmapFactory.Options.html#inBitmap

#63 lutz.ben...@gmail.com Feb 26, 201

I have an even more frustrating version of that bug. We use a mapview that supports rotation and thus is slightly larger than the screen area. When users zoom in a few times they can very easil cause the OOM condition. I have no control over the way the Google Maps API works, do I?

By the way, OOM is catchable. Not that it would help much though...

Here's my current code

```
            public void onClick(View v) {
                v.performHapticFeedback(HapticFeedbackConstants.LONG_PRESS);
                System.gc();
                try {
                    mMapView.getController().zoomIn();
                } catch (OutOfMemoryError e) {
                    e.printStackTrace();
                }
                System.gc();
                zoomLevel = mMapView.getZoomLevel();
                SharedPreferences.Editor editor = settings.edit();
                editor.putInt("zoomLevel", zoomLevel);
                editor.commit();

            }
```

#64 Boris...@gmail.com Feb 26, 201

my code uses imageswitcher to display jpg images not more than 120kb compressed (drawables). The code works fine on most of the devices, but I've got 60-80 reports with that error daily (i have 800 downloads daily). Very frustrating and hurts my app rating!

checked on memory leaks, tried to compress jpegs even no - was not able to reproduce on 3 my devices and emulator but still getting them every day!

#65 agpx...@gmail.com May 7, 201

DECLINED??????? This is a TRUE NIGHTMARE-BUG! It make my customers (and me too) really ANGRY!! He Google Team... Fix That Terrible Bug!!!!!!!!!!!

#66 piotr.b...@gmail.com May 10, 201

Had to deal with this myself. I cached my images using SoftReferences, thinking that it would be the best way to implement image cache. I was assuming that once Dalvik runs out of memory, currently unused Bitmaps will be removed from memory. Apparently I was wrong, no such thing happens. I'm guessing it's because Bitmap holds on to native memory outside Dalvik's knowledge.

And the OOM is thrown when native memory allocation fails.

For my case, the solution was very simple. Catch the OutOfMemoryError and release memory manually by calling Bitmap.recycle() and getting rid of references to the Bitmap. No need to call System.gc().

Something like that:

```
try {
    image = BitmapFactory.decodeByteArray(data, 0, data.length);
} catch (OutOfMemoryError e) {
    // go over unused images and release them
    // and try loading your image once again
    image = BitmapFactory.decodeByteArray(data, 0, data.length);
}
```

#67 winroot...@gmail.com May 18, 201

whenever the application is really out of memory at this point it will not fail correctly and might get stuck :(but yeah it is one of many workarounds

it is sad to see that this bug has been declined so fast and it is really hard to reinstate a bugreport -.-

#70 szatmari...@gmail.com May 30, 201

I think google hates us all.

#71 xxxhuang...@gmail.com Jun 17, 201

Google doesn't care about developers.

#72 Vladys...@gmail.com Jun 24, 201

This looks like the cause of the Gingerbread keyboard crashes as well...

#73 agpx...@gmail.com Jun 29, 201

I have ported my application on iOS and I have no problem at all (also with device with by far less memory quantity than my Android Phone). I think that the way Android handles the memory budget of a process is really poor.

#74 agpx...@gmail.com Jun 29, 201

I have tried to fully replace the BitmapFactory with my custom PNG loader. This reduce the issue on many phones, but the problem still occurs on other. I think that the problem is related to the way Android is modded by some phone manufacturer. Android Team should clarify the concept of "VM budget". It's a fraction of the physical memory or is it fixed? Can this limit changed via software? Android automatically increase the budget for a process if it require more memory? The same application, runned on the same version of Android, with the same quantity of free memory behaves differently. My application run fine on some device with Froyo and with 64 Mb of free RAM and report out of memory on some device with Froyo and 200 Mb of free RAM. This is strange. IMHO some critical "parameters", like the VM Budget, should be definited by Android Team and phone manufacturers must not able to change it!

#75 blahblah676@gmail.com Jun 29, 201

agpxnet: android apps have a limit of either 16MB or 24MB of virtual memory per process (which I agree is ridiculously small). However I can say that after rewriting our app so that the bitmaps are recycled, then nulled, then gced in the onDestroy, we haven't had any reports of this error.

I understand the reasoning behind the memory limit - you don't want any one app hogging all your memory. However it would be nice to give the user the option of granting more memory to an app rather than just crashing.

#76 xxxhuang...@gmail.com Jun 30, 201

The java virtual machine, Davlik, that you're app runs on doesn't actually allocate memory for bitmaps in the heap given to Davlik for your app. From my understanding, calling system.gc() doesn't really do anything to clear those bitmaps.

One way to lesson the impact of bitmaps is to decode them from file using the following code that makes use of temp memory, and purgeable memory:

```
public static Bitmap decodeFile(
    final File f,
    final int suggestedSize) {
    if (f == null) {
      return null;
    }
    if (f.exists() == false) {
      return null;
    }
    // return BitmapFactory.decodeFile(f.getAbsolutePath());
    try {
      // System.gc();
      // decode image size
      final BitmapFactory.Options o = new BitmapFactory.Options();
      o.inJustDecodeBounds = true;
      BitmapFactory.decodeStream(new FileInputStream(f), null, o);
      // Find the correct scale value. It should be the power of
      // 2.
      final int requiredSize = suggestedSize;
      int widthTmp = o.outWidth, heightTmp = o.outHeight;
      int scale = 1;
      while (true) {
        if ((widthTmp / 2) < requiredSize
          || (heightTmp / 2) < requiredSize) {
          break;
        }
        widthTmp /= 2;
        heightTmp /= 2;
        scale *= 2;
      }
      // decode with inSampleSize
      final BitmapFactory.Options o2 = new BitmapFactory.Options();
      o2.inSampleSize = scale;
      o2.inTempStorage = new byte[64 * 1024];
      o2.inPurgeable = true;
      Bitmap bitmap = null;
      try {
        bitmap =
          BitmapFactory
            .decodeStream(new FileInputStream(f), null, o2);
      } catch (final Throwable e) {
        System.gc();
      }
      return bitmap;
    } catch (final Throwable e) {
      Bug.report(e);
      System.gc();
      return null;
    }
  }
```

You should also extend the default ImageView and override the onDetachedFromWindow to automatically null out the bitmaps in your imageviews.

#77 blahblah676@gmail.com Jun 30, 201

There are two heaps: dalvik and native. The dalvik heap is used for memory allocations in your java code (when you do a 'new'), whereas the native heap is used for things like bitmaps. The 16/24MB per-process limit applies to the sum of memory allocations on both heaps. Garbage collection does affect bitmaps AFAIK.

#78 DengRui0...@gmail.com Jul 5, 201

take a look at camera app of capcake,see how Google made a deal with the bitmap object.

#79 emilesw...@gmail.com Jul 8, 201

I had a similar issue that occurred with a custom BaseApdater whos items contained an image retrieved using
BitmapFactory.decodeResource(c.getResources(), resID); where i tried c as the context of the

activity and getApplicationContext(). In both instances the application crashed.

The custom base adapter was assigned to an instance of Gallery. 100mill secs after the activity was created it would start another activity. To and throw it would go between an empty activity and the activity with the gallery.

After a couple of seconds the app would crash with the VM memory errors below

```
07-08 19:32:18.558: ERROR/dalvikvm-heap(22989): 284648-byte external allocation too large for thi
process.
07-08 19:32:18.598: ERROR/GraphicsJNI(22989): VM won't let us allocate 284648 bytes
```

Having read the previous messages i added the following to my activity,

```
@Override
protected void onDestroy() {
        super.onDestroy();
        ((CustomAdapter)coverFlow.getAdapter()).recycle();
}
```

where i added the following to my CustomAdapter

```
public void recycle()
{
        for (int i = 0; i < getCount(); i++) {
                CustomDataItem item = getItem(i);
                item.aBitmap.recycle();
        }
}
```

This fixed the issue for me. Poor design in other areas probably didn't help though. I created th custom adapter every time a new activity was started, so i wonder if there is a better way of doing that.

#80 mrsl...@gmail.com Jul 19, 201

Since it seems a lot of misinformation is being spread in this ticket, I wanted to try and addres a few things that might help people before they are led down the wrong path.

First, you must understand a few things about bitmaps and the garbage collector. A bitmap (as of 2.x and earlier), has memory in both the Dalvik and native heap (with the majority of it stored o the native side). The garbage collector is only capable of freeing memory in the Dalvik heap. Native memory for each bitmap is freed using a finalizer.

The exact steps in dalvik to collect a bitmap is as follows:

1) A bitmap becomes eligible for garbage collection
2) The GC eventually runs and sees that the object requires finalization
3) The bitmap is placed onto the finalizer queue
4) The finalizer eventually runs and native memory is freed (at this point, most of the memory yo assigned to the bitmap is freed)
5) The GC eventually runs again and the bitmap is removed from the Dalvik heap. This "remainder" object is relatively small when compared to the previously in-use native memory

There are a few things you should understand about finalizers. The following is some data I extracted from http://publib.boulder.ibm.com/infocenter/javasdk /v5r0/index.jsp?topic=%2Fcom.ibm.java.doc.diagnostics.50%2Fhtml%2Fmm_gc_coexist.html

Finalizers:
    Are not run in any particular sequence
    Are not run at any particular time
    Are not guaranteed to run at all
    Will run asynchronously to the Garbage Collector

So lets talk about what does happen when you try to allocate a bitmap and there is no memory available:

1) A bitmap or large chunk of memory tries to be allocated
2) The JVM realizes there isn't enough memory available
3) The JVM kicks off a GC to prevent an out of memory error (http://download.oracle.com/javase

/1.4.2/docs/api/java/lang/OutOfMemoryError.html)
4) The GC completes and the allocation is attempted again. If it fails, an out of memory error is thrown, Otherwise, you get the memory

So, the key thing to notice in the above steps is that the finalizer queue is NOT run until its empty or even mentioned at all. The main problem everyone is having is that they are stuck waitin for the finalizer thread to allow the release of the native memory bitmaps are using.

To fix this problem, simply be sure to call recycle() on bitmaps when you are done using them. This will free the large portion of native memory they use, and will prevent the problems everyon has been having.

Fast Tips:

1) NEVER call System.gc() yourself. This has been propagated as a fix here, and it doesn't work. Do not do it. If you noticed in my explanation, before getting an OutOfMemoryError, the JVM already runs a garbage collection so there is no reason to do one again (its slowing your program down). Doing one at the end of your activity is just covering up the problem. It may causes the bitmap to be put on the finalizer queue faster, but there is no reason you couldn't have simply called recycle on each bitmap instead.

2) Always call recycle() on bitmaps you don't need anymore. At the very least, in the onDestroy o your activity go through and recycle all the bitmaps you were using. Also, if you want the bitmap instances to be collected from the dalvik heap faster, it doesn't hurt to clear any references to the bitmap.

3) Calling recycle() and then System.gc() still might not remove the bitmap from the Dalvik heap. DO NOT BE CONCERNED about this. recycle() did its job and freed the native memory, it will just take some time to go through the steps I outlined earlier to actually remove the bitmap from the Dalvik heap. This is NOT a big deal because the large chunk of native memory is already free!

4) Always assume there is a bug in the framework last. Dalvik is doing exactly what its supposed to do. It may not be what you expect or what you want, but its how it works.

#81 je...@apptitude.dk Jul 19, 201

@#80, good summary and description

I've had this problem just changing the image resource on an imageview often with calls to setImageResource on ImageView. I believe the resource system caches bitmaps loaded via the resource system(?), so getting hold of the loaded bitmap via the BitmapDrawable on the ImageView and recycling this is a bad idea, I guess? - or am I getting this wrong?

If I'm right, is my only option then to stop using the resource system for loading bitmaps, doing this explicitly via the BitmapManager and manually recycling these?

#82 andreas....@googlemail.com Jul 19, 201

@#80, your post is not the final truth.

First, how am I supposed to recycle bitmaps that I have not allocated (referenced in XML resource and so on)?

Second, why should I call Bitmap.recycle() when the Javadoc explicitely states NOT TO DO it? -> http://developer.android.com/reference/android/graphics/Bitmap.html#recycle%28%29

Third, why does my example crash nevertheless, even with calling Bitmap.recycle() (both with and without System.gc(), for that matter)?

Fourth, regarding the question if this is a bug or not: I usually look at the specification, and the specification appears to be the JavaDoc in this case. Dalvik does obviously not do what it specified, because my recycled bitmaps are NOT being garbage collected in all cases and because manually recycling bitmaps should NOT be needed in the first place (according to the spec).

Never assume there is no bug in your code, if someone supplies you with a test case.

#83 mrsl...@gmail.com Jul 19, 201

@#81, it may be the best option if you have very large bitmaps. It appears that Resources does some type of internal caching, but it might be best to avoid using it.

@#82

1) In some cases you will have no choice. I am talking about best practices.

2) It does not say that so please stop spreading misinformation. I will copy and paste for you "This is an advanced call, and normally need not be called, since the normal GC process will free up this memory when there are no more references to this bitmap."

I think the flaw in the documentation there is that its implying you shouldn't be calling it on a regular basis (when I would argue that you should). Again, my argument being that if you can free the memory ASAP why wouldn't you? All the documentation is really trying to tell you is that one does not NEED to call this function for the bitmaps memory to be reclaimed. As I mentioned before the finalizer will eventually be run and will eventually free the native memory.

3) I haven't had a chance to review your particular sample but if I get the chance I will take a look at it. Maybe what you are doing is in violation of what recycle() says, which is that recycle() is not a synchronous call. So it may take some time to free the memory.

4) Again, recycle() has nothing to do with garbage collection. Please re-read my post to understand what recycle() actually does. The garbage collector will need to run a minimum of two times to actually remove the bitmap from the dalvik heap. The bitmap's native data, on the other hand, should be freed shortly after a call to recycle().

Finally, Android isn't my code. I'm just trying to prevent the spread of misinformation.

#84 kf6nvr Jul 19, 201

@#80: The simplest case I had for reproducing this issue was a call out to the image chooser and using the result URI with an ImageView. In no case was my code every using a Bitmap object. Yet, this ~5 line app easily runs out of memory since the entire contents of camera phone images is being duplicated during orientation changes.

Here's a small bit of code that demonstrates this. It never uses Bitmap objects. But after a pick rotate to landscape then to vertical and the VM is out of memory and can no longer display the image. To me, that points to an internal issue since it can display the image twice before it run out. The internal objects aren't freeing something in a timely manner.

```
public class ImagePickerActivity extends Activity {
        private static final String LOG_TAG = "ImagePickerActivity";
        private static final int IMAGE_PICK_REQUEST_ID = 1;
        private Uri mSelectedImage;
        ImageView mPickedImage;

        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);
                mPickedImage = (ImageView) findViewById(R.id.picked_image);

                if (savedInstanceState != null) {
                        String uri = savedInstanceState.getString("imageUri");
                        if (uri != null) {
                                mSelectedImage = Uri.parse(uri);
                                mPickedImage.setImageURI(mSelectedImage);
                        }
                }
        }

        public void clickToPick(View view) {
                Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
                intent.setType("image/*");
                startActivityForResult(intent, IMAGE_PICK_REQUEST_ID);
        }

        protected void onActivityResult(int requestCode, int resultCode,
                        Intent imageReturnedIntent) {
                switch (requestCode) {
                case IMAGE_PICK_REQUEST_ID:
                        if (resultCode == RESULT_OK) {
                                mSelectedImage = imageReturnedIntent.getData();
                                mPickedImage.setImageURI(mSelectedImage);
                        }
                }
```

```
        }

        @Override
        protected void onSaveInstanceState(Bundle outState) {
                if (mSelectedImage != null) {
                        outState.putString("imageUri", mSelectedImage.toString());
                }
                super.onSaveInstanceState(outState);
        }
}


 And the corresponding layout:

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button
        android:layout_width="wrap_content"
        android:text="@string/pick"
        android:layout_height="wrap_content"
        android:id="@+id/button1"
        android:layout_gravity="center_horizontal"
        android:onClick="clickToPick"></Button>
    <ImageView
        android:src="@drawable/icon"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:id="@+id/picked_image"
        android:adjustViewBounds="true"
        android:scaleType="centerInside"></ImageView>
</LinearLayout>
-----
```

#85 kf6nvr Jul 19, 201

And yes, I know that there are ways to do this particular use case. But the point is if I'm never creating a Bitmap object myself, I wouldn't expect to be the one to have to call recycle() on it. And if the internal code is calling recycle() on it, then it continues to be a case of the memory not being freed up quickly enough, even though it's been marked as free. And to that, there seems to be no simple solution in a simple case like this. You effectively have to rewrite to manage th bitmaps separately, resizing them if they're too big, and so on. It's a lot more code when something _this_ simple seems like it should be possible.

#86 mrsl...@gmail.com Jul 19, 201

I don't have too much time, but I ran with my theory (assuming andreas test case still didn't wor with a recycle() in the for loop).

My theory was that if the recycle() call is asynchronous, I suspected the native memory would be freed upon next message pump cycle. Here's the code:

It ran for over 300 iterations fine for me.

bitmap-bug.zip
600 KB Download

#87 kf6nvr Jul 19, 201

#86: That's interesting. Your bitmap should be using about 7MB each decode, so a reasonable test.

Though, it's not assigning the Bitmap to an ImageView and doing so complicates matters. The Bitma here is never being used at all, so would be easy for the next collection to cleanup.

In my example, however, I never even deal with the Bitmap directly. Just an ImageView, which should be destroyed at each orientation change. The particular phone I was just testing with returns a URI to images that decode to about 10MB each.

Several of the work arounds here have talked about having to do things like clearing the internal ImageView bitmap, etc.

#88 kf6nvr Jul 19, 201

Real quickly: ImageView does not return a Bitmap. Thus, there's no way to call Bitmap.recycle() i you're just dealing with an ImageView object. Calling setImageDrawable(null) does not work around the problem. No drawing cache is returned from getDrawingCache().

Waiting for gc or other background freeing doesn't work. Once it runs out of memory, it never get it again*.

*Curiously, never doesn't mean never. When on the debugger, I can click the Cause GC button, and then it works again for a couple of orientation changes. Clicking between each orientation change does, in fact, keep it working indefinitely.

But.... never call System.gc(). Which I do agree with. It's just that, well, that works around th problem. Even waiting minutes, gc never seems to be automatically called.

Something is amiss somewhere.

#89 mrsl...@gmail.com Jul 19, 201

@#88: I understand the awkwardness of your situation, but in the end, you are still indirectly telling Android to create a bitmap for you. For your case, I would imagine the best thing to do would be in onDestroy:

1) Cast the drawable returned from the ImageView to a BitmapDrawable
2) Get the bitmap from the drawable
3) Recycle the bitmap

You probably know that, though, and are looking for a more "automated" solution. Unfortunately, I can't think of a way to do that. It would be really nice if ImageViews kept track of the way thei drawables were populated and made an intelligent decision about how / when to recycle bitmaps, but, for now, it doesn't. I suspect the situations that it would actually be able to do this reliably are limited.

In the end, it all boils down to the notion that the only reliable way to free native memory is with a finalizer. The only alternative is that you would have to be sure to call recycle() on every bitmap created otherwise (or leak native memory).

#90 kf6nvr Jul 19, 201

For those following along with #89, here's what #89 is saying in code:

```
@Override
protected void onDestroy() {
        BitmapDrawable bd = (BitmapDrawable)mPickedImage.getDrawable();
        Bitmap bm = bd.getBitmap();
        bm.recycle();
        super.onDestroy();
}
```

What it is showing, and what #89 is basically confirming, is that there are some internal objects that don't necessarily handle things intelligently. An app must work around them in some situations.

And to be clear, what's meant by #89 saying "[ImageView can't] do this reliably" is simple. If I, the app, am holding on to the Bitmap object, I would not want the ImageView to internally call recycle. So, ImageView can't do this every time. Clearly, a conundrum -- one that should probably be documented better (somewhere).

Should ImageView use recycle when _it_ created the Bitmap internally as a side effect of another call (set as set URI)? Probably. Is it a bug? Unclear. Given that forcing a gc through the debugger does clean up the memory, the ImageView isn't necessarily doing anything wrong. It hasn' leaked a reference. Should the gc have run when it first tried to alloc a block of memory and failed? Maybe. Or maybe it is, but there's something different about the debugger-triggered gc. (For example, it might simply be that the finalizers are being triggered by the debugger gc but not the internal auto-gc.)

Without diving into the internals, it's impossible to tell whether it's a misuse, misunderstanding, or a bug -- and at what level that exists.

To #89: Thank you for adding to the conversation here. I suspect it's been very useful to people.

#91 je...@apptitude.dk Jul 19, 201

I have a rather large app that is rich in graphics. This includes different, large (screen-size) bitmap background images on the different activities, referenced as drawable resources in layout xml. If the user quickly launches new, different activities clicks like menu -> activity1 -> back -> activivity2 -> back -> activity3 -> back ..., the system sometimes run of of bitmap memory. I took a quick look that the Resources source, and drawables are cached as expected: (line 1664)

http://grepcode.com/file/repository.grepcode.com/java/ext/com.google.android/android/2.3.4_r1/android/content/res/Resources.java#Resources.loadDrawable%28android.util.TypedValue%2Cint%29

#92 dtoniolo...@gmail.com Jul 26, 201

I ran into this critical issue as well and i think a good workaround is to use the assets folder instead of res/drawable for images.

This is my getView() method of my Adapter class i use for the Gallery component:

```
        public View getView(int position, View convertView, ViewGroup parent) {
            ImageView imgView = new ImageView(cont);
            Bitmap bitmap;
            try {
// images is a String[] array of file names saved in the root of assets.
                InputStream istream = getAssets().open((String)images[position]);
                bitmap = BitmapFactory.decodeStream(istream);
                imgView.setImageBitmap(bitmap);
            } catch (IOException e) {
    Log.d("MY_ASSETS_ERROR", e.getMessage());
            }

            imgView.setDrawingCacheEnabled(true);
            imgView.buildDrawingCache(true);
            imgView.setLayoutParams(new Gallery.LayoutParams(LayoutParams.FILL_PARENT,
LayoutParams.FILL_PARENT));
            imgView.setScaleType(ImageView.ScaleType.FIT_XY);

            return imgView;
        }
```

#93 mrsl...@gmail.com Jul 28, 201

@#92, That is an approach we have been taking as well as it easily allows for the recycling of imagery.

#94 idi...@gmail.com Aug 18, 201

use BitmapFactory.decodeFileDescriptor instead of decodeStream/decodeFile. It worked for me.

I believe there is a bug in the native methods used in decodeStream and decodeFile. BitmapFactory.decodeFileDescriptor uses different native call: nativeDecodeFileDescriptor() than the others.

#95 kilaka@gmail.com Aug 23, 201

I can't replace decodeStream with decodeFileDescriptor, because my stream is not FileInputStream The data comes from the network and file system.

Project Member #96 jr...@android.com Aug 24, 201

The bitmap in the demo isn't 4mb, it is actually 8mb or 16mb depending on the default bitmap config (whether or not it defaults to RGB_545 or ARGB_8888). For a bitmap this large, you should consider explicitly picking a format using BitmapFactory.Options.inPreferredConfig.

When you rotate the device, the old activity instance may still be around while the new one is being created. The GC is unable to free the bitmaps in the first instance, as there is still a

reference to it. Thus, you try to allocate a couple more bitmaps, which then causes the OOM. An easy fix is to just add this to your activity:

```
    @Override
    protected void onDestroy() {
        super.onDestroy();
        bitmap = null;
    }
```

#97 novaho...@gmail.com Aug 29, 201

In my opinion the issue is that your are holding a reference to your activity somewhere so it is never garbage collected when you switch orientations. You should never have to manually recycle a bitmap that your ImageView has created for you.

#96 is correct.

#98 tuan...@gzone.com.vn Sep 5, 201

I'm working on a qr code project which based on ZXing, and I met this problem too. Yes I know the Bitmap.recycle(), and I did it on every Bitmap that I loaded. But I still met this prolem, one in my expandable list where the imageview load resources (in drawable folder), and one after scanner scan qr code successfully. It happened randomly (and rarely), and I've used DDMS to observe the thread heap, and it never go over 5m. How can my phone (N1) which has a 24m heap size crashed wit the Bitmap OOM???
I've read all of comments in this thread but still not found a solution.

#99 TimMesse...@gmail.com Sep 22, 201

Same as @#94 - the FileDescriptor did the job for me.
Works fine now.

#100 draksia Oct 3, 201

In my case I had this problem when using a gallery of images that I loaded from URL.

If I overload the onConfigurationChanged method, instead of reloading all the images again I just set the gallery to already existing adapter.

If I set the adapter to null, setContentView(R.layout.main) and then find the gallery by Id and set it to the old adapter I can rotate pretty much forever.

So by not setting the adapter to null it didn't allow the old gallery to release its 's resources

So if you can overload onConfiguration change and hold on to your bitmaps you might be ok. At least that works for me.

#101 bhadd...@spartechsoftware.com Nov 12, 201

I had the same problem and Comment 19 (move the bitmap out of /res/drawable to /res/drawable-nodp folder) worked for me. No other changes were made. I simply moved the bitmap to a new drawable-nodpi folder and no more crashes when the application was restarted.

#102 rmattis...@gmail.com Dec 15, 201

This may help - keeps it clean --

http://www.ryangmattison.com/post/2011/12/15/AsynchronousImageProvider.aspx

#104 alexey.s...@gmail.com Jan 15, 201

Comment 76 was the right answer for me until now.
I thought the point is that the correct scale value should be the power of 2! When I load bitmaps this way I can load bitmaps 1M pixels and when I try to load with a wrong scale (not power of 2) exception is raised even with a bitmap 0.01M pixels.
Man who shared the secret Thank you!

The code to load bitmaps for my game

```
public Bitmap loadBitmapFromAssets(String path, int suggestedWidth, AssetManager assetManager)
```

```
    {
        AssetManager am = null;
        if(assetManager != null){
                am = assetManager;
        }else{
                am = context.getAssets();
        }
        InputStream is = null;
        try
                {
                is = am.open(path, AssetManager.ACCESS_STREAMING);
                BitmapFactory.Options bo = new BitmapFactory.Options();
                bo.inJustDecodeBounds = true;
                BitmapFactory.decodeStream(is, null, bo);
                int bmpWidth = bo.outWidth;
                int bmpHeight = bo.outHeight;
                int scale = 1;
                while(bmpWidth > suggestedWidth)
                {
                                bmpWidth /= 2;
                                bmpHeight /= 2;
                                scale *= 2;
                }
                bo.inJustDecodeBounds = false;
                bo.inSampleSize = scale;
                return BitmapFactory.decodeStream(is, null, bo);
        }
        catch (IOException e) {
                Log.e(MB.TAG, "bitmap loading failed: " + e.getMessage());
                return null;
        }
    }
```

where 'path' is something like 'images/sprites/800/sprite1.png' (inside 'assets' folder of my Android project)

BUT! Always there is but :) I've decided to load all bitmaps including those for menus using the way described above. And... And VM Budget came back! Again.
So the right answer for me is to buy mac and stay on developing games for iOS instead of getting crazy this VM Budget Crazy Crash. Buy, buy, Android. If android developers don't want to build stable API I don't now why I should spend my time on Android.

#105 krueger....@gmail.com Jan 15, 201

I've been struggling with this same problem for many hours today, and have read dozens of article and posts about how to work around it. The only thing that worked in my case was the suggestion i #90, i.e. extracting the Bitmaps from the ImageView objects in question and recycling them manually in the activity's onDestroy().

For the record, I am using a ListView object where each row view contains at least one ImageView (often more). I was intentionally loading very large images into each Bitmap object (with inSampleSize always being 1), and my app crashed consistently after 2 orientation changes. Note that the images are user-created and are thus being loaded from external storage, not from resources. Nevertheless, updating my onDestroy with the code suggested in #90 fixed this problem. This is what my onDestroy looks like currently:

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

        ListView lv = (ListView)findViewById(R.id.listItems);
        {
            for (int i = 0; i < lv.getChildCount(); ++i)
            {
                View v = lv.getChildAt(i);
                ImageView imgView = (ImageView) v.findViewById(R.id.imageViewId);
                BitmapDrawable bd = (BitmapDrawable)imgView.getDrawable();
                Bitmap bmp = bd.getBitmap();
                bmp.recycle();
            }
        }

        System.gc();
```

```
    }
```

(I should probably check whether the imgView.getDrawable() is an instance of BitmapDrawable befor
casting, but this code works for me so far).

It still boggles my mind that an issue of this magnitude hasn't been fixed or, at the very least,
thoroughly documented yet. I can live with manually releasing resources, but at least wrapping it
around a nice ImageView.clearResources() method (or something equivalent) that is clearly
documented would go a long way towards easing the confusion of many users such as myself who end
up struggling with this problem for an inexcusably long time.

#106 helderga...@gmail.com Jan 27, 201

I also have a ListView filled with Bitmaps in each row and today my published app reported this
problem. I tried the solution from #150, but after restarting an activity A after the onDestroy
from the same activity A, sometimes I get this:

```
01-27 16:45:18.194: E/AndroidRuntime(2876): java.lang.RuntimeException: Canvas: trying to use a
recycled bitmap android.graphics.Bitmap@4503cf58
01-27 16:45:18.194: E/AndroidRuntime(2876):     at
android.graphics.Canvas.throwIfRecycled(Canvas.java:955)
01-27 16:45:18.194: E/AndroidRuntime(2876):     at
android.graphics.Canvas.drawBitmap(Canvas.java:1044)
01-27 16:45:18.194: E/AndroidRuntime(2876):     at
android.graphics.drawable.BitmapDrawable.draw(BitmapDrawable.java:323)
01-27 16:45:18.194: E/AndroidRuntime(2876):     at
android.widget.ImageView.onDraw(ImageView.java:860)
01-27 16:45:18.194: E/AndroidRuntime(2876):     at android.view.View.draw(View.java:6740)
```

I would think that after destroyng an activity, all its resources would be freed and not reused,
but that doesn't seems to be the case here. Sooo I'm back to stage one with this bug. I'm not
entirely sure to if using recycle() won't break something later on now.

#107 tryprasa...@gmail.com Feb 17, 201

#105 , #106 and others having OOM problem when changing orientation (probably when you use your
own bitmaps)
Android's way of handling orientation change is to destroy and recreate the activity,
which is we are encountering the OOM problems.
So when dealing with large data sets like bitmaps or data that is expensive to load like bitmap
decode, we can pass specific resources(bitmaps) and reuse them after orientation gets changed
instead of reloading/ encoding bitmaps again etc.

It is achieved by retaining the resources like bitmaps using the method
"onRetainNonConfigurationInstance()", which gets called only when needed i.e in cases like
orientation change, otherwise it is not called.
The method is actually called between onStop() and OnDestroy() methods.

Once we have done this, after orientation change we can get back the resources that we retained b
calling another method "getLastNonConfigurationInstance()" and use them.

Change the implementation of setting bitmaps to the views (in onCreate/OnStart etc) to make it
look like following.

```
 final MyObject data = (MyObject) getLastNonConfigurationInstance();
    if (data == null)
    {
        //Activity launching for first time, So load resources like usual
        // like Bitmap.decode
        // setContentView
    }
    else
    {
       //Activity is being relaunced due to a orientation/config change
       // and we have got back the resources/bitmaps retained, just reuse them here
       // myImageView.setImageBitmap(data);
       //assuming the type of MyObject to be Bitmap, of which "data" is an instance.

    }
```

MyObject can be as simple as android.graphics.Bitmap or any custom complex object to handle

numerous resources.
This is explained in more detail in the following links:

[1] http://developer.android.com/resources/articles/faster-screen-orientation-change.html

[2] http://developer.android.com/guide/topics/resources/runtime-changes.html#RetainingAnObject

Hope this helps.

#108 shuhao8...@gmail.com Feb 28, 201

I used a jpg picture as background of myspp. And this pic used by two activities in the same app. Crash all the time because of oom. Comment 19 (move the bitmap out of /res/drawable to /res/drawable-nodpi folder) solved my problem.

#109 kjedr...@gmail.com Mar 28, 201

In my situation, I had a collection of ImageViews in LinearLayouts. If the activity was destroye and created too many times (i.e. orientation changes) I would get an OutOfMemory exception.

I resolved this not by recycling the bitmaps or by forcing garbage collection, but rather explicitly removing all the views from my layout in the activity's onDestroy() method:

```
        @Override
        public void onDestroy()
        {
                super.onDestroy();
                if(theLayout != null)
                {
                        int size = theLayout.getChildCount();

                        for(int x=0;x<size;x++)
                        {

((LinearLayout)theLayout.getChildAt(x)).removeAllViews();
                        }
                        theLayout.removeAllViews();
                }
                //System.gc();

        }
```

In my activity, I had a parent LinearLayout (theLayout) followed by a series of child LinearLayouts. The child LinearLayouts contained ImageViews holding the bitmaps. In my solution I stepped through all the layouts on my activity and explicitly removed the views from them, and then removed all views explicitly from the parent.

Note that the line System.gc() is commented out. I initially tried this fix by forcing garbage collection after removing the views. It seems however, this may not be necessary. My testing so far has shown it doesn't seem to be required.

I'm not certain why this solution works. Technically, it shouldn't be necessary. I'm supposing that the removal of the views has somehow correctly flagged the bitmaps for a normal gc operation. I should also note a simple call to gc() in the onDestroy is not sufficient. OOM will still occur. Again, I am surmising that the explicit removal of the views has made the correct adjustments to the bitmap reference counts in order for them to be re-collected.

I would be interested to see what results others would get by performing a similar operation in their own activities.

#110 jauming...@gmail.com May 17, 201

another way to fix android oom:
[1]modify: dalvik/vm/Init.c(near line762:):
search:
```
  gDvm.heapSizeMax = val;
```
replace to:
```
  LOGE("heapSizeMax '%d' \n", val);
  gDvm.heapSizeMax = 64 * 1024 * 1024; // Spec says 75% physical mem
```

[2]rebuild and replace following .so shared library:
/system/lib/libdvm.so
/system/lib/libdvm_assert.so
/system/lib/libdvm_interp.so

```
/system/lib/libdvm_sv.so
```

Init.c
53.2 KB View Download

#111 maur...@everyme.com May 19, 201

```
@swine... if you use a static reference for the bitmap gc will work as expected. the bitmap itsel
has a reference to the activity so by rotating the screen you are recreating an activity but
holding a reference to an old activity not allowing for gc. So setting a static var would allow
the bitmap to be overwritten and lose its reference to the previous activity allowing for proper
gc.
```

#113 amirf...@gmail.com Oct 5, 201

```
In a final stage at comment 110 system lib

/system/lib/libdvm_assert.so
/system/lib/libdvm_interp.so
/system/lib/libdvm_sv.so

This point in int.c stop the works. http://letdld.blogspot.com and i can't work next.

Any suggestion ..?
```

#114 MattiaMe...@gmail.com Nov 6, 201

```
After reading all the comments I try to put together the things that most users said works.


1)I Put pictures in Assets and not in Drawable, so I can upload as a bitmap and recycle more easi

example:

InputStream is = getAssets().open("nomeimmagine.png");
bitmap = BitmapFactory.decodeStream(istream);
imgView.setImageBitmap(bitmap);

// I try to add these lines to speed up loading :
imgView.setDrawingCacheEnabled(true);
imgView.buildDrawingCache(true);


2) Remove all "android:src="@drawable/.." " from xml files if you do not use them.


3) Inside onDestroy():

    a) call .recycle(): ( This will free the large portion of native memory thet Bitmap use)
        - on each Bitmap that I no longer use
        - on the Bitmap of View with images:

        BitmapDrawable bd = (BitmapDrawable)mPickedImage.getDrawable();
        Bitmap bm = bd.getBitmap();
        bm.recycle();

    b) set null: (..if you want the bitmap instances to be collected from the dalvik heap faster,
clears any references to the bitmap)
        - all the resources that I no longer use ( Bitmap , Timer, Canvas , ... = null)
        - all Drawable of Views (imageView.setImageDrawable(null))
        - all Callback (.setCallback(null))

4) I could do the same in "onPause()", but then in "onResume()" I will have to restore what I nee
to reuse

6) I could put the code that causes the OutOfMemoryError in a "try{}catch(OutOfMemoryError e){}"
and try to run again after freeing memory

example:

int n;
int x = 3;
public void method()
```

```
{
    try
    {
       // here code that causes OOM
    }
    catch ( OutOfMemoryError e)
    {
        Log.d("MY_OOM_ERROR_n ° " + n,e.getMessage());

        // Repeat X times
        if (n < X)
        {
            // here code for freeing memory (i.e. recycle Bitmap that i use nomore)

            // try again
            method();
            n++;
        }
        else
        {
            Bug.report ( e);
        }
    }
}
```

7) As an alternative to point (1), I can put images in "/res/drawable-nodpi" instead of "/res/drawable".
For some users only this fix OOM error and others argue that ".recycle()" works "really".

8) Another solution is to remove all images containing View when no longer needed

9) Another solution If you use any threads, and they get passed Context variables which are reall a reference to your activity, make sure you convert them into the application context so that the do not refer to the activity and thus leak it.

Sorry for my bad english, I hope this helps ;)

Add a comment

Vote for this issue and get email change notifications

Enter your comments