codepath / **android_guides**

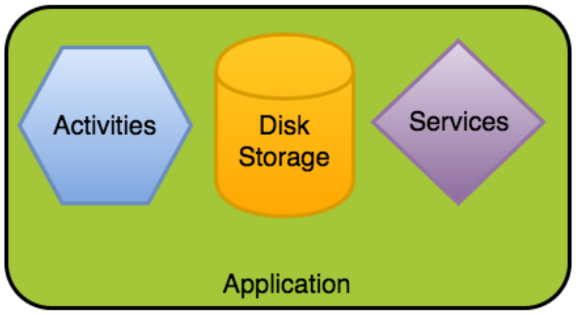
Edit New Page

Understanding the Android Application Class

Alexander Manoharan edited this page on 13 Nov 2017 · 4 revisions

Overview

The `Application` class in Android is the base class within an Android app that contains all other components such as activities and services. The Application class, or any subclass of the Application class, is instantiated before any other class when the process for your application/package is created.



This class is primarily used for initialization of global state before the first `Activity` is displayed. Note that custom `Application` objects should be used carefully and are often **not needed at all**.

Custom Application Classes

In many apps, there's no need to work with an application class directly. However, there are a few acceptable uses of a custom application class:

- Specialized tasks that need to run before the creation of your first activity
- Global initialization that needs to be shared across all components (crash reporting, persistence)
- Static methods for easy access to static immutable data such as a shared network client object

Note that you should **never store mutable shared data** inside the `Application` object since that data might disappear or become invalid at any time. Instead, store any mutable shared data using persistence strategies such as files, `SharedPreferences` or `SQLite`.

Defining Your Application Class

If we do want a custom application class, we start by creating a new class which extends `android.app.Application` as follows:

```
import android.app.Application;

public class MyCustomApplication extends Application {
        // Called when the application is starting, before any other application obj
        // Overriding this method is totally optional!
        @Override
        public void onCreate() {
            super.onCreate();
            // Required initialization logic here!
```

Pages 192

Finding these guides helpful?

We need help from the broader community to improve these guides, add new topics and keep the topics up-to-date. See our contribution guidelines here and our topic issues list for great ways to help out.

Check these same guides through our standalone viewer for a better browsing experience and an improved search. Follow us on twitter @codepath for access to more useful Android development resources.

Interested in ramping up on Android quickly?

(US Only) If you are an existing engineer with 2+ years of professional experience in software development and are serious about ramping up on Android quickly, be sure to apply for our free evening 8-week Android bootcamp.

We've trained over a thousand engineers from top companies including Apple, Twitter, Airbnb, Uber, and many others leveraging this program. The course is taught by Android experts from the industry and is specifically designed for existing engineers.

Not in the United States? Please fill out our application of interest form and we'll notify you as classes become available in your area powered by local organizers.

Clone this wiki locally

`https://github.com/codepa`

```
        }

        // Called by the system when the device configuration changes while your com
        // Overriding this method is totally optional!
        @Override
        public void onConfigurationChanged(Configuration newConfig) {
            super.onConfigurationChanged(newConfig);
        }

        // This is called when the overall system is running low on memory,
        // and would like actively running processes to tighten their belts.
        // Overriding this method is totally optional!
        @Override
        public void onLowMemory() {
            super.onLowMemory();
        }
}
```

And specify the `android:name` property in the the `<application>` node in `AndroidManifest.xml` :

```
<application
   android:name=".MyCustomApplication"
   android:icon="@drawable/icon"
   android:label="@string/app_name"
   ...>
```

That's all you should need to get started with your custom application.

Limitations and Warnings

There is always data and information that is needed in many places within your app. This might be a session token, the result of an expensive computation, etc. It might be tempting to use the application instance in order to avoid the overhead of passing objects between activities or keeping those in persistent storage.

However, you should **never store mutable instance data** inside the `Application` object because if you assume that your data will stay there, your application will inevitably crash at some point with a `NullPointerException` . The application object is **not guaranteed to stay in memory forever, it will get killed**. Contrary to popular belief, the app won't be restarted from scratch. Android will create a new `Application` object and start the activity where the user was before to give the illusion that the application was never killed in the first place.

So how should we store shared application data? We should store shared data in one of the following ways:

- Explicitly pass the data to the Activity through the intent.
- Use one of the many ways to persist the data to disk.

Bottom Line: Storing data in the `Application` object is error-prone and can crash your app. Prefer storing your global data on disk if it is really needed later or explicitly pass to your activity in the intent's extras.

Resources

- https://developer.android.com/reference/android/app/Application.html
- http://stackoverflow.com/questions/18002227/why-extend-an-application-class
- https://tausiq.wordpress.com/2013/01/27/android-make-use-of-android-application-class-as-singleton-object/

- https://androidresearch.wordpress.com/2012/03/22/defining-global-variables-in-android/
- http://www.helloandroid.com/tutorials/maintaining-global-application-state
- https://www.mobomo.com/2011/5/how-to-use-application-object-of-android/
- http://www.developerphil.com/dont-store-data-in-the-application-object/
- http://www.vogella.com/tutorials/AndroidLifeCycle/article.html#managing-the-application-life-cycle