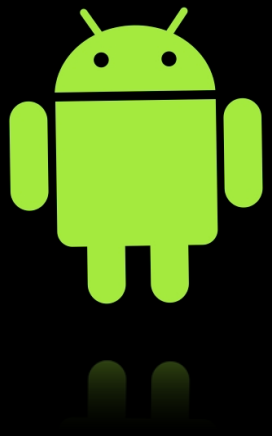
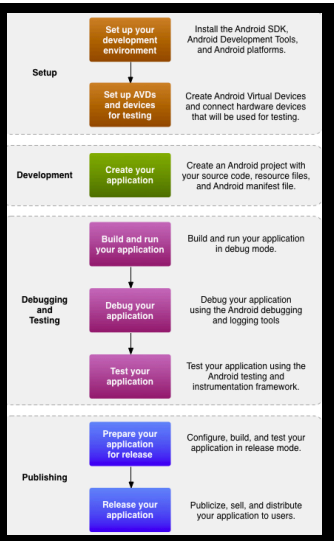


What you need to know for Lab 1



code to publish workflow

application components

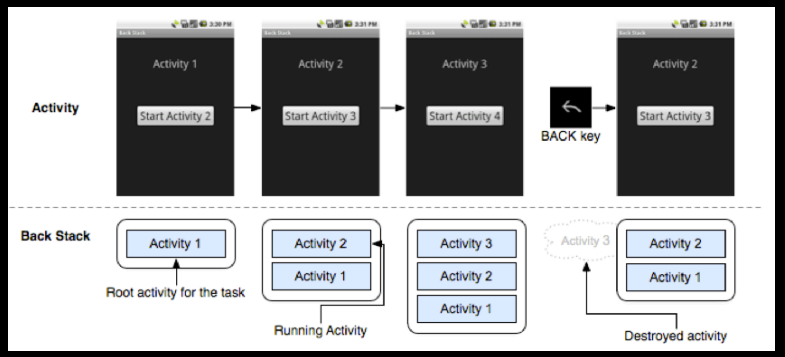
activities
An activity is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map. Each activity is given a window in which to draw its user interface. The window typically fills the screen, but may be smaller than the screen and float on top of other windows

An application usually consists of multiple activities that are loosely bound to each other. Typically, one activity in an application is specified as the "main" activity, which is presented to the user when launching the application for the first time. Each activity can then start another activity in order to perform different actions

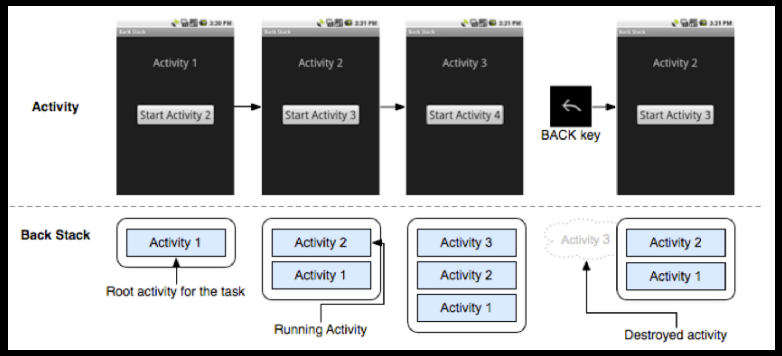
services
A *service* is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. For example, a service might play music in the background

content providers
A *content provider* manages a shared set of application data. You can store the data in the file system, an SQLite database, on the web, or any other persistent storage location your application can access. Through the content provider, other applications can query or even modify the data (if the content provider allows it).

broadcast receivers
A *broadcast receiver* is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Applications can also initiate broadcasts—for example, to let other applications know that some data has been downloaded to the device and is available for them to use

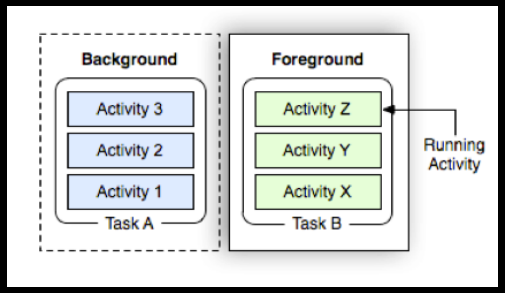


stack (last in, first out): when the current activity starts another, the new activity is pushed on the top of the “stack” and “takes focus”.



When the user presses the BACK key, the current activity is popped from the top of the stack (the activity is destroyed) and the previous activity resumes (the previous state of its UI is restored).

background and foreground tasks



At this point, the user can also switch back to Task B by going Home and selecting the application icon that started that task (or by touching and holding the HOME key to reveal recent tasks and selecting one). This is an example of multitasking on Android.

Create

The system calls this when creating your activity. Within your implementation, you should initialize the essential components of your activity. Most importantly, this is where you must call setContentView() to define the layout for the activity's user interface

Resumed

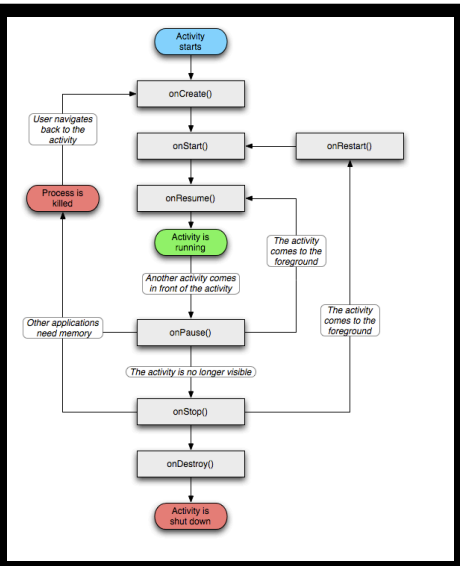
The activity is in the foreground of the screen and has user focus. (This state is also sometimes referred to as "running".)

Paused

Another activity is in the foreground and has focus, but this one is still visible. That is, another activity is visible on top of this one and that activity is partially transparent or doesn't cover the entire screen. A paused activity is completely alive (the activity object is retained in memory, it maintains all state and member information, and remains attached to the window manager), but can be killed by the system in extremely low memory situations.

Stopped

The activity is completely obscured by another activity (the activity is now in the "background"). A stopped activity is also still alive (the activity object is retained in memory, it maintains all state and member information, but is *not* attached to the window manager). However, it is no longer visible to the user and it can be killed by the system when memory is needed elsewhere.



activity life-cycle

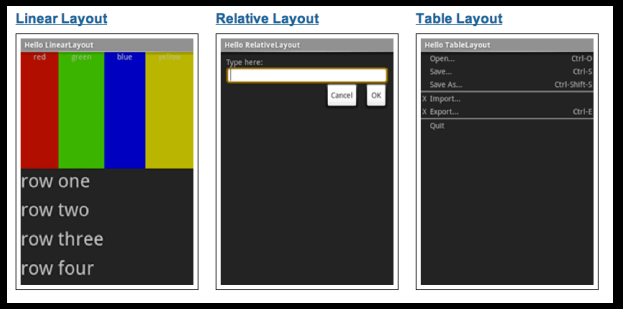
```java
public class ExampleActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // The activity is being created.
    }
    @Override
    protected void onStart() {
        super.onStart();
        // The activity is about to become visible.
    }
    @Override
    protected void onResume() {
        super.onResume();
        // The activity has become visible (it is now "resumed").
    }
    @Override
    protected void onPause() {
        super.onPause();
        // Another activity is taking focus (this activity is about to be "paused").
    }
    @Override
    protected void onStop() {
        super.onStop();
        // The activity is no longer visible (it is now "stopped")
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        // The activity is about to be destroyed.
    }
}
```

life-cycle callbacks

tutorials

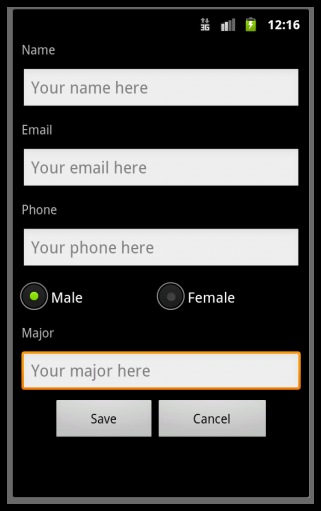
Hello world

Hello views

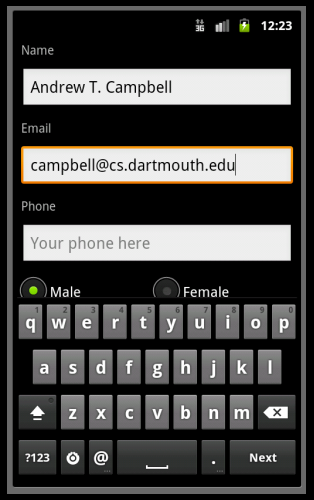


Lab 1 MyRuns

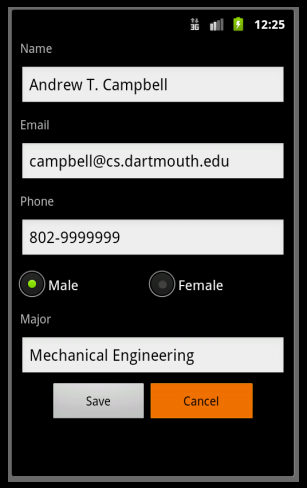
designing and implementing a simple UI



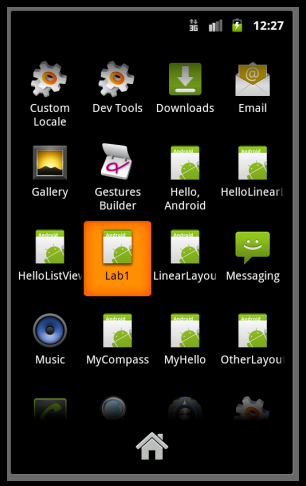
layout (UI design)

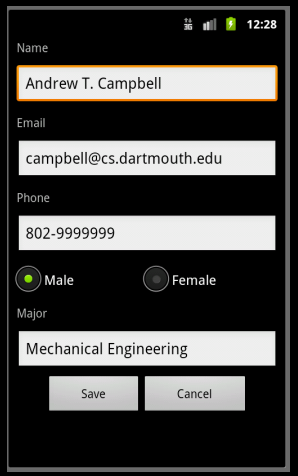


user input (event handling)



save (storing data)





exit app; reopen; data must persists

setting up the UI

activities and views
layout (views)
event handling (click on)
shared preferences (storage of data)
run code
logging - Log.d(TAG,“”);
emulator

lets look at the code