Lecture
Activity Lifecycle

DT228/3

Dr. Susan McKeever

Android application

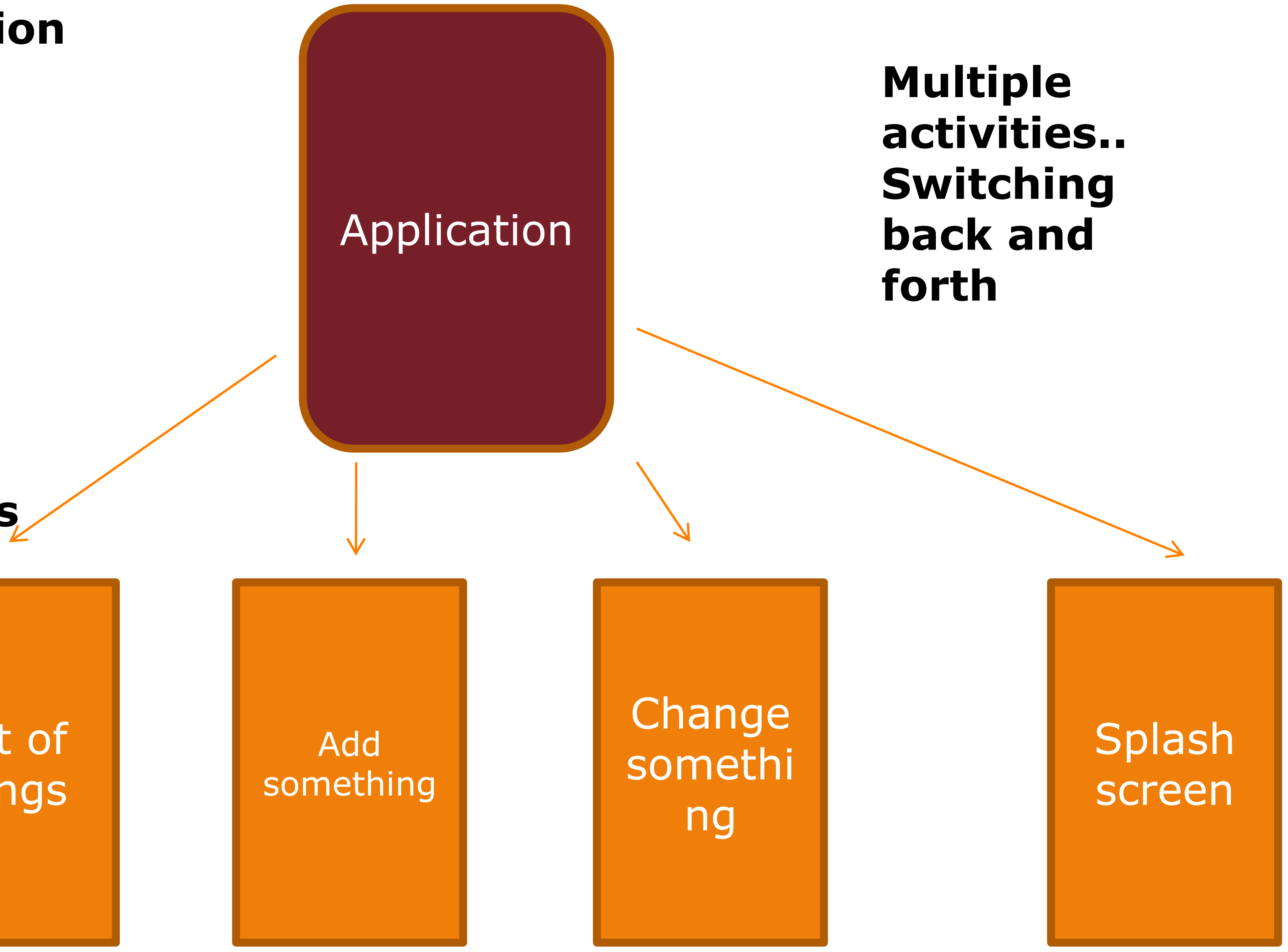
Can be made up of one or more of 4 types of components

Screens that do something. Use *intents* to switch between.	**Activities**
No user interface. Runs in background. E.g. audio player	**Services**
Supplies data to one or more apps. E.g. weather data.	**Content providers**
Listens out and responds to broadcast announcements e.g. "you've got mail"	**Broadcast receivers**

Activities versus applications

Application

Activities



Multiple activities.. Switching back and forth

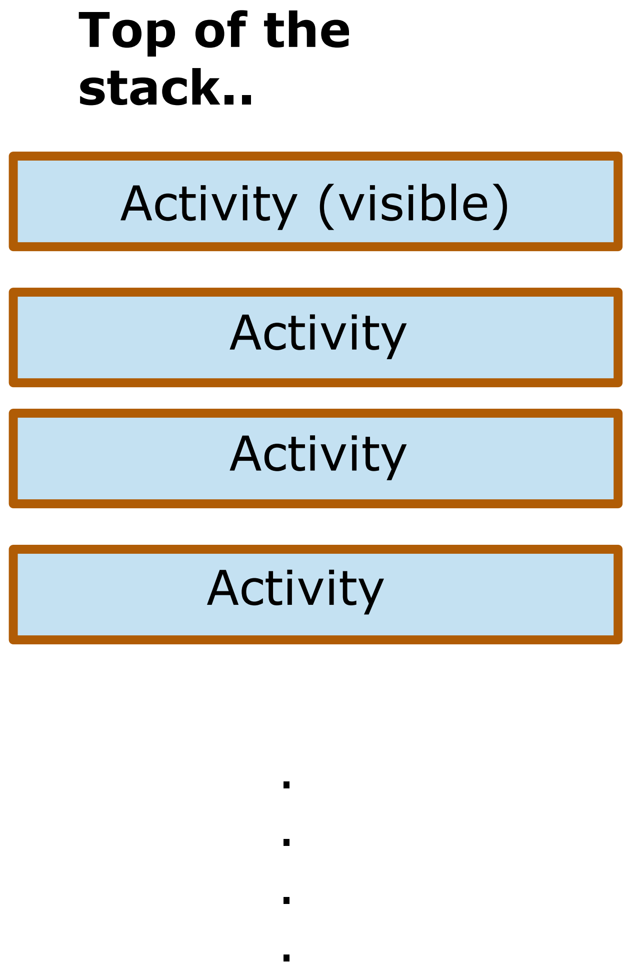
Activities

- Phones have a lot going on.. With prioritisation needed
  - Limited RAM
  - Calls comes in – continue game? Take call?
- Activities get stopped, restarted, killed off all the time
- Design consideration.. What needs to be done if my activity is stopped or killed off? Or if it is restarted?

Scenarios

- ❑ Receives a phone call or switches to another app while using your app.
- ❑ Consuming valuable system resources when the user is not actively using it.
- ❑ Losing the user's progress if they leave your app and return to it at a later time.
- ❑ Crashing or losing the user's progress when the screen rotates between landscape and portrait orientation.

Activity stack



- Start an activity – it goes to the top of the stack
- Your activity starts another activity.. Then the called activity goes to the top
    - Like a card deck..
- Only the activity on top of the stack is visible to the user
- Every time a new activity is started, it is pushed on top of the stack
- When an activity finishes (e.g. by pressing the device's BACK button) it is **removed** from the stack

Activity States

Active

To the foreground, running; Usable

Paused

Lost focus but visible e.g. Call comes in

Stopped

Not visible, Obscured by another active activity

Dead

Dropped from memory by system

Activity lifecycle – moving between the four states

Activities have **lifecycle methods** to support transition

e.g.

- When activity activated, onCreate() method automatically called
- When activity killed off, onDestroy() automatically called

Override the activity lifecycle methods in your activity to make the right things happen at the right time

Activity lifecycle methods

- **OnCreate ()**
  - Called when the activity is first created
  - Put in setup info e.g. layout, listeners
  - *Bundle* supplied in case activity previously frozen..

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
```

- **onRestart()** : Called after your activity has been stopped, before it is started again.

Activity lifecycle methods

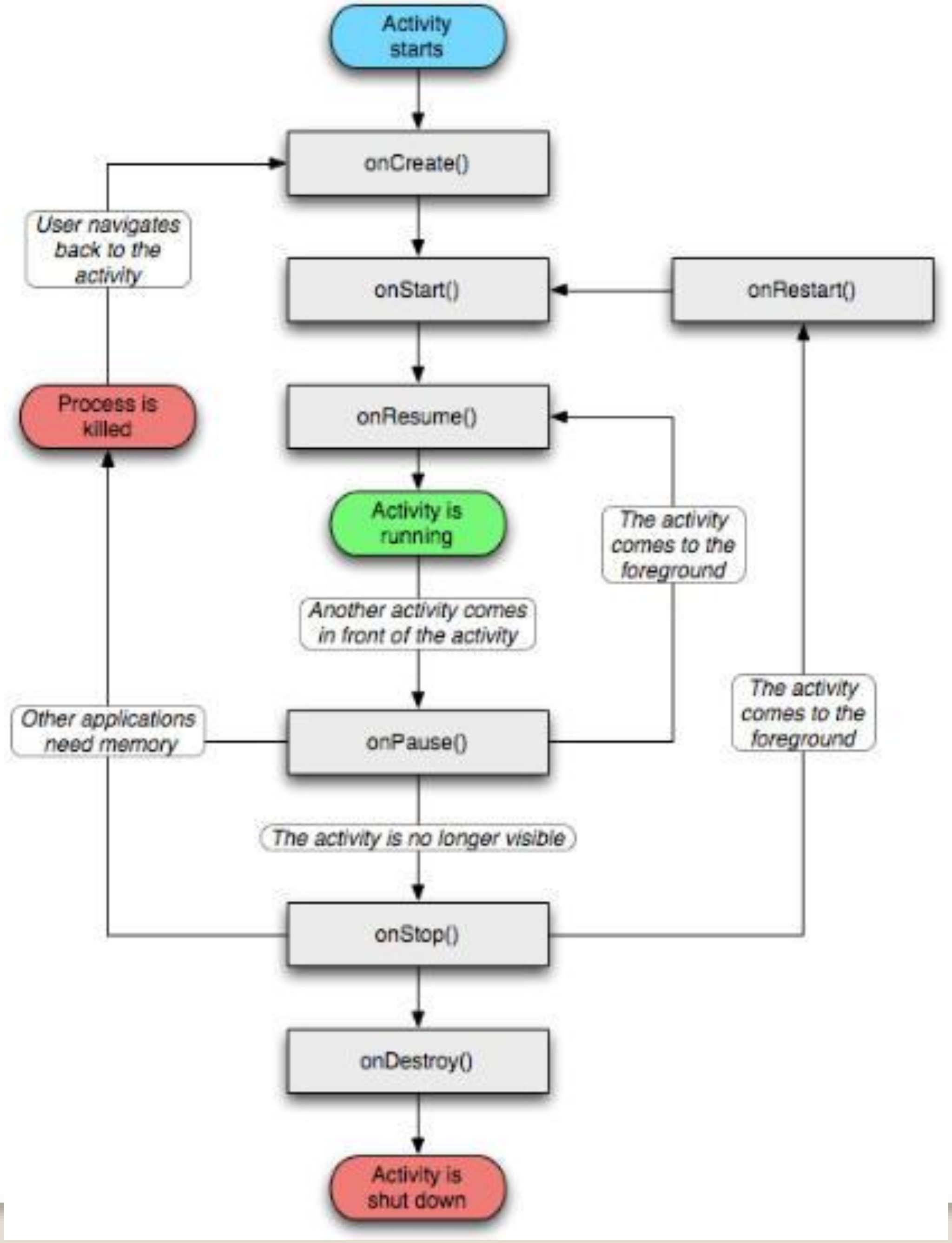
- **OnStart()**
  - Called when the activity is becoming visible to the user.

- **onResume()** : Called when the activity starts interacting with the user.. Activity at the top of the “stack”

- **onPause()** : Another activity is being activated, forcing current activity to be paused

Activity lifecycle methods

- **OnStop()**
  - Activity is about to be stopped
- **onDestroy()** :
  - Activity is being killed off
    - By system (e.g. low memory)
    - By your code (calling finish() method);


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
The activity comes to the foreground
Other applications need memory
onPause()
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

How do I know when to use lifecycle methods?

- onCreate() you use already…
- If you notice that you’re “losing” info when your activity is returned to, you need to use or enhance one or more of the lifecycle methods
  - Becomes obvious when you’re testing..
  - E.g. Rotate your phone.. onCreate() is called again (although you can control this…)
  - E.g. onPause() is a good place to close a database - otherwise, error if attempting to open a dB already open

How do I know when to use lifecycle methods?

- If there are battery hungry or resource hungry processing that needs to be suspended
    - E.g. Location checking

onPause() example

```
Protected void onPause()
{
        super.onPause();
         myDBmanager.close();
}
```

What would this do?

```
Protected void onPause()
{
          super.onPause();
           Log.i.("test", "onPaused called);
}
```

Note: Starting activities..

Various ways. Three common ones:

1. Activity specificied Manifest as launcher activity

2. One activity calls another activity directly.

   - Saw this in flashlight activity: use an Intent (and remember manifest file entry)

   ```
   StartActivity(new Intent(this, NewActivity.class));
   ```

2. One activity calls another activity and waits for a result.
   - E.g. Dialler activity picking a contact from a list:

   ```
   startActivityForResult(Intent, intCode)
   ```

startActivityForResult() method..

One activity starts another activity directly..AND forces the called activity to return to it..
The “onActivityResult()” method is automatically called on return to execute what should happen on return..

```
Intent i = new Intent(this, other.class);
startActivityForResult(i, ACTIVITY_CREATE);

protected void onActivityResult(int requestCode, int
                                resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    //TODO handle here.
}
```

startActivityForResult() Example.

Dialler screen – picking a contact from contacts list –
    launch the activity that has the list of contacts
    user chooses the contact they want from list
    Control returns to dialler with the "result" (contact_)

Dialler activity has the `startActivityForResult()`

Dialler activity has the `onActivityResult() method`

Contacts list activity containts a `setResult(int)method`
    Followed by a `finish() method`