IT 4785. Phát triển ứng dụng cho thiết bị di động

Chapter 3. Life Cycle of Android Application

TS. Nguyễn Hồng Quang
Viện Công nghệ thông tin và Truyền thông
Trường Đại học Bách Khoa Hà Nội

Tài liệu tham khảo

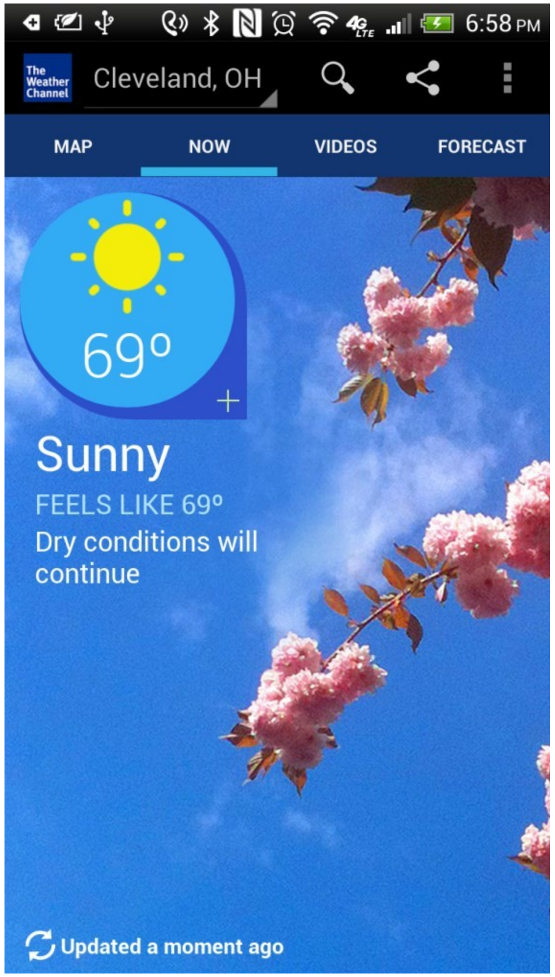
Mobile Application Development – Android OS, Victor Matos, Cleveland State University

Activity Class

- An Activity object is similar to a WindowsForm. It usually presents a single graphical visual interface (GUI) which in addition to the displaying/collecting of data, provides some kind of ‘code-behind‘ functionality.
- A typical Android application contains one or more Activity objects.
- Applications must designate one activity as their **main** task or entry point. That activity is the first to be executed when the app is launched.
- An activity may transfer control and data to another activity through an interprocess communication protocol called **intents**.
- For example, a login activity may show a screen to enter user name and password. After clicking a button some authentication process is applied on the data, and before the login activity ends some other activity is called.

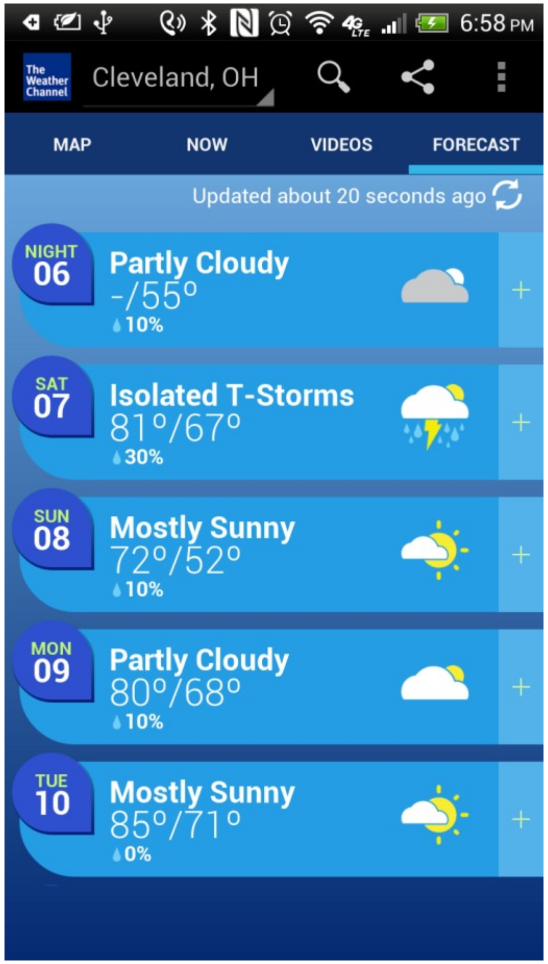
Ví dụ một ứng dụng bao gồm nhiều Activity

Weather Channel app
GUI-1- Activity 1



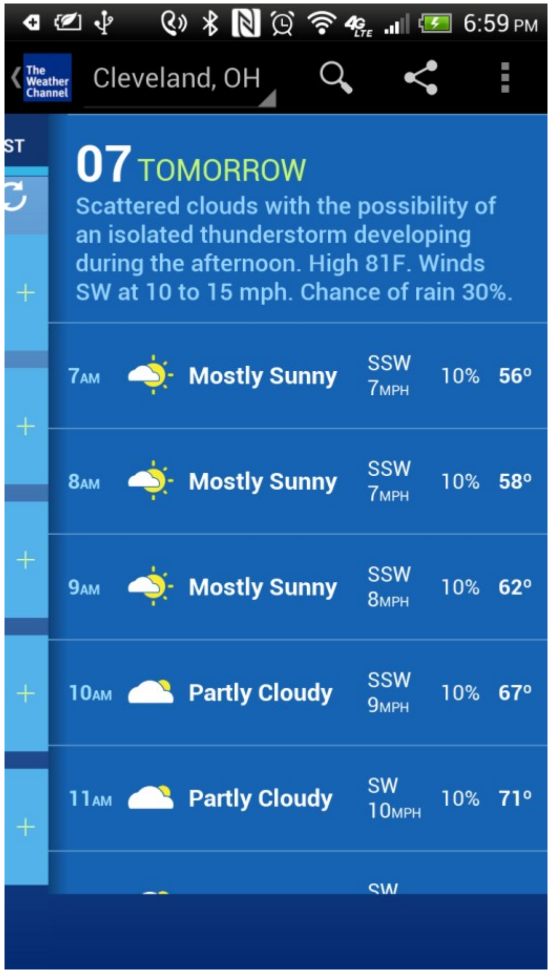
Sunny
FEELS LIKE 69°
Dry conditions will continue

Weather Channel app
GUI-2- Activity 2



NIGHT 06	Partly Cloudy	-/55°	10%
SAT 07	Isolated T-Storms	81°/67°	30%
SUN 08	Mostly Sunny	72°/52°	10%
MON 09	Partly Cloudy	80°/68°	10%
TUE 10	Mostly Sunny	85°/71°	0%

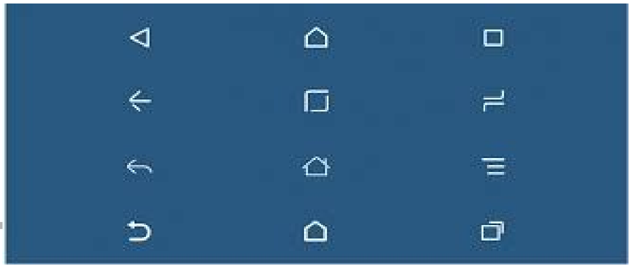
Weather Channel app
GUI-3- Activity 3



07 TOMORROW
Scattered clouds with the possibility of an isolated thunderstorm developing during the afternoon. High 81F. Winds SW at 10 to 15 mph. Chance of rain 30%.

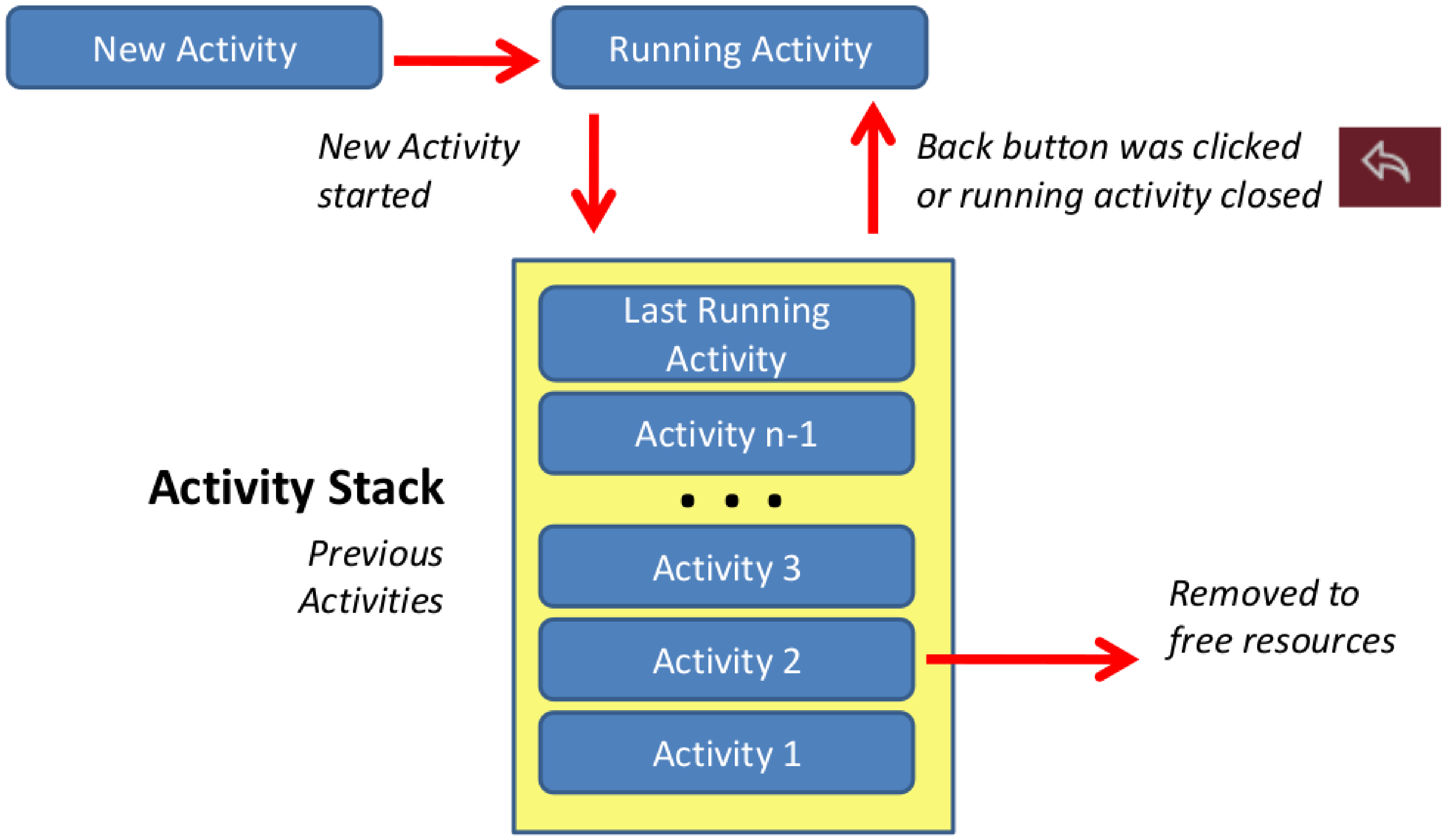
7AM	Mostly Sunny	SSW 7MPH	10%	56°
8AM	Mostly Sunny	SSW 7MPH	10%	58°
9AM	Mostly Sunny	SSW 8MPH	10%	62°
10AM	Partly Cloudy	SSW 9MPH	10%	67°
11AM	Partly Cloudy	SW 10MPH	10%	71°

The Activity Stack



- Activities in the system are scheduled using an activity stack.
- When a new activity is ***started***, it is placed on top of the stack to become the running activity
- The previous activity is pushed-down one level in the stack, and may come back to the foreground once the new activity finishes.
- If the user presses the **Back Button** the current activity is terminated and the previous activity on the stack moves up to become active.
- Android 4.0 introduced the **‘Recent app’** button to arbitrarily pick as ‘next’ any entry currently in the stack

The Activity Stack



Sự tồn tại của một tiến trình trong ứng dụng Android

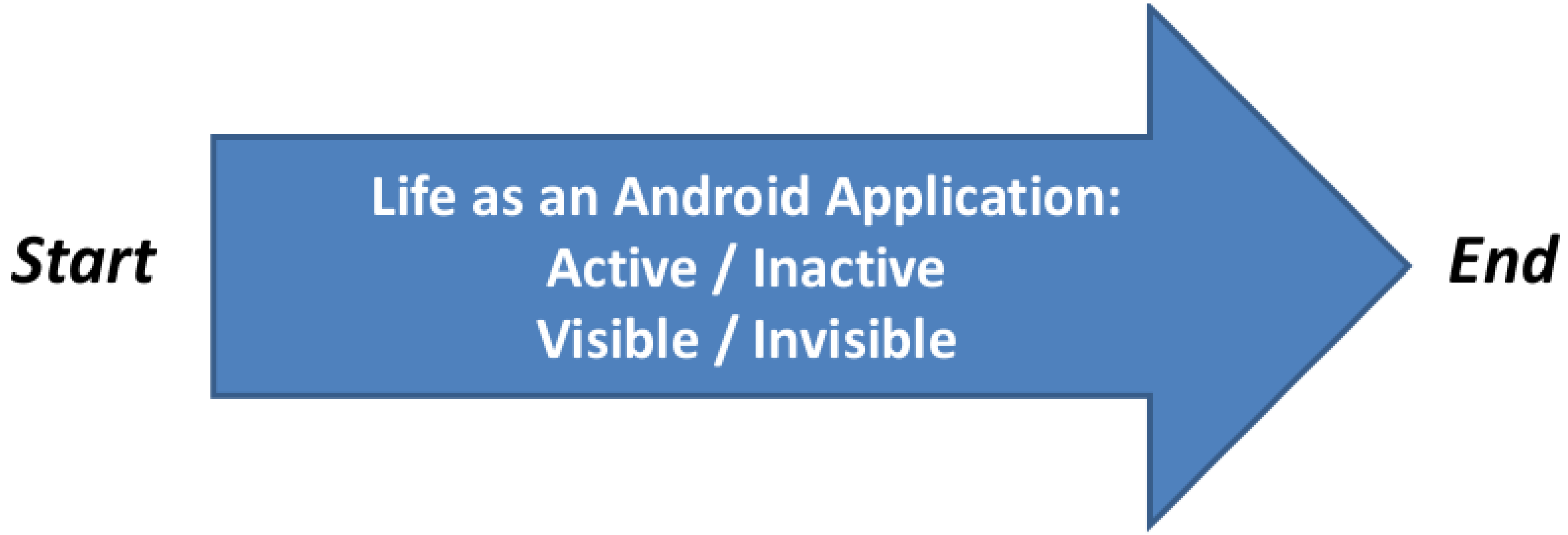
Occasionally hardware resources may become critically low and the OS could order early termination of any process.

The decision considers factors such as:

1. Number and age of the application's components currently running,
2. Relative importance of those components to the user, and
3. How much free memory is available in the system.

Android Activity’s Life Cycle

1. A **beginning** - responding to a request to instantiate them
2. An **end** - when the instances are destroyed.
3. A sequence of **in-between** states – components sometimes are active or inactive, or in the case of activities - visible or invisible.



Life Cycle Callbacks

When progressing from one state to the other, the OS notifies the application of the changes by issuing calls to the following protected transition methods:

void onCreate( )
void onStart( )
void onRestart( )
void onResume( )

void onPause( )
void onStop( )
void onDestroy( )

```java
public class ExampleActivity extends Activity {
  @Override
  public void onCreate (Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // The activity is being created.
  }

  @Override
  protected void onPause() {
    super.onPause();
    // Another activity is taking focus
    // (this activity is about to be "paused").
  }
```

Most of your code goes here

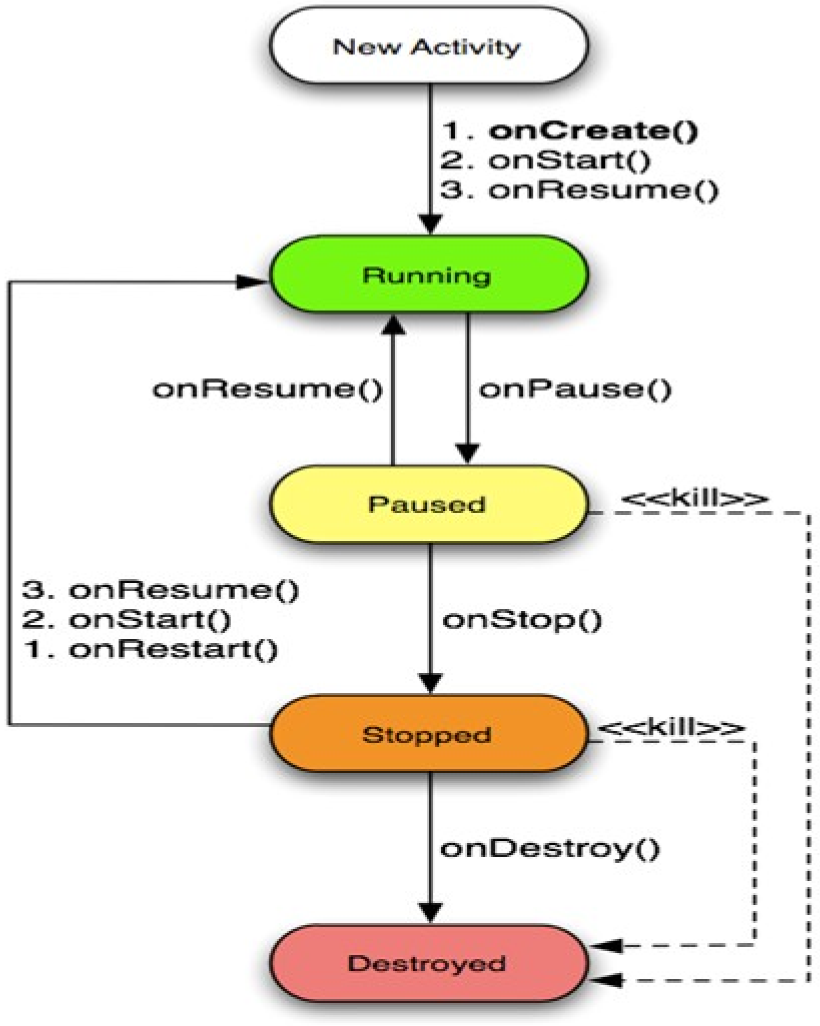
Save your important data here

Life Cycle:
Activity States and Callback Methods

An activity has essentially three phases:

1. It is ***active*** or ***running***
2. It is **paused** or
3. It is ***stopped***

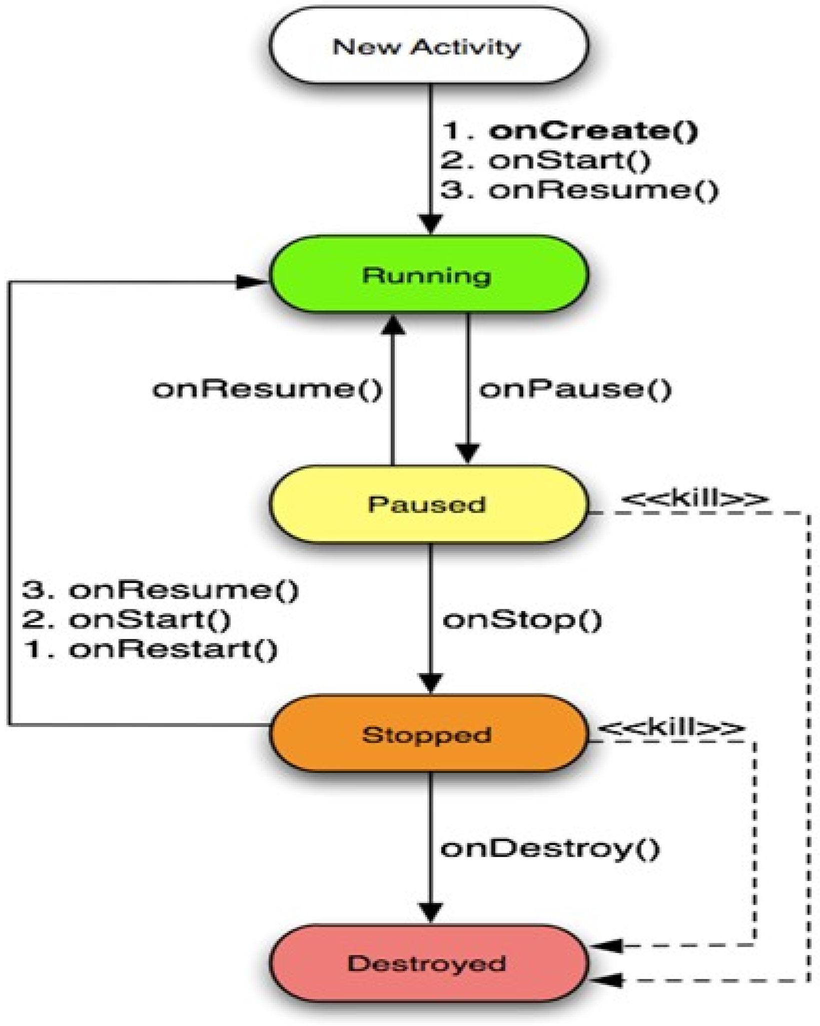
Moving from one state to the other is accomplished by means of the callback methods listed on the edges of the diagram.



Activity State: RUNNING

1. Your activity is active or running when it is in the foreground of the screen (seating on top of the activity stack).

This is the activity that has “focus” and its graphical interface is responsive to the user’s interactions.



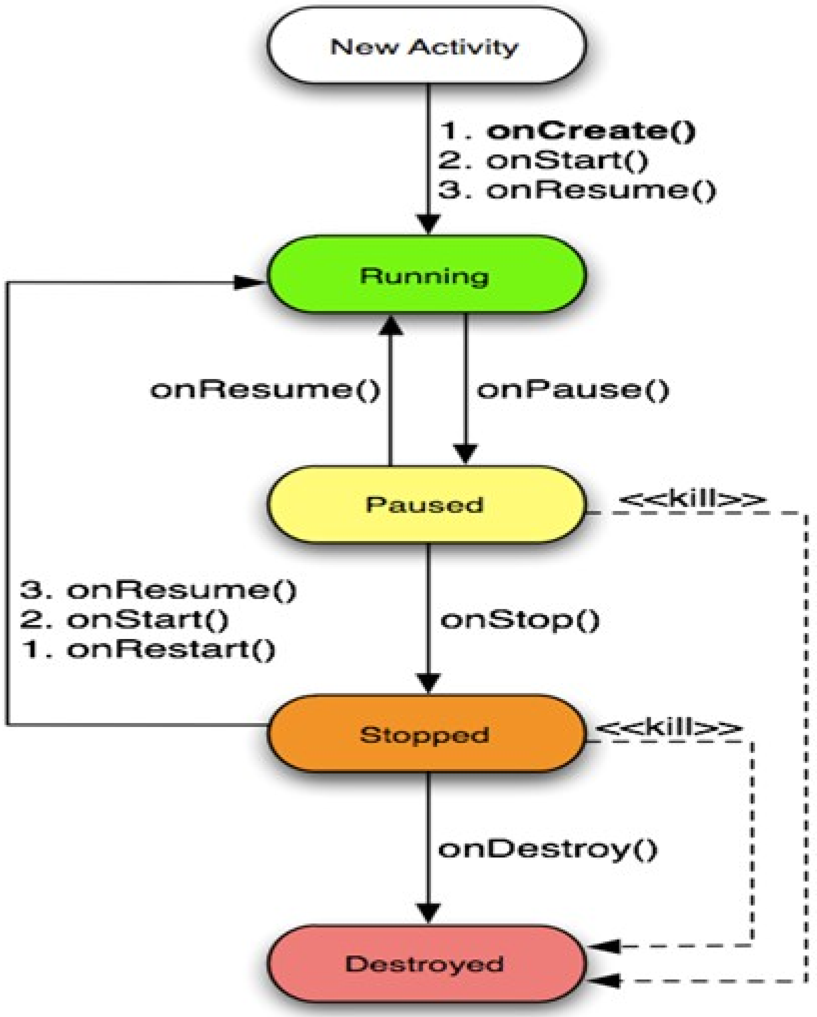
Activity State: PAUSED

2. Your Activity is paused if it has lost focus but is still visible to the user.

That is, another activity seats on top of it and that new activity either is transparent or doesn't cover the full screen.

A paused activity is alive (maintaining its state information and attachment to the window manager).

Paused activities can be killed by the system when available memory becomes extremely low.

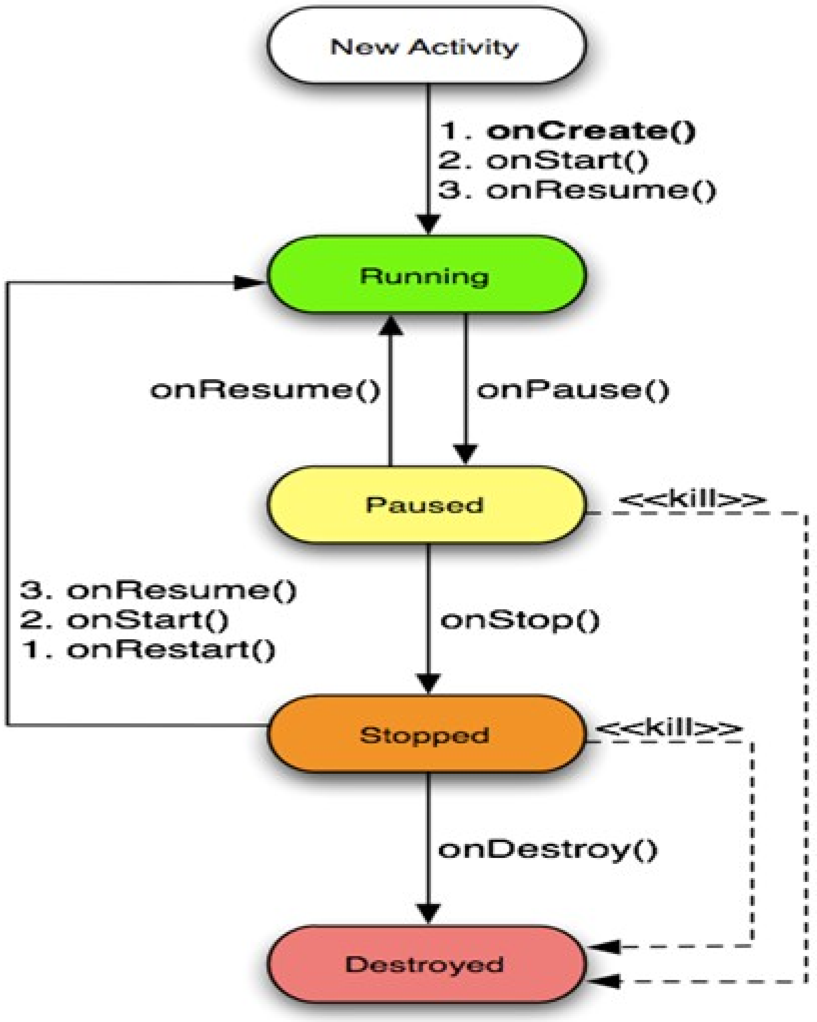


Activity State: STOPPED

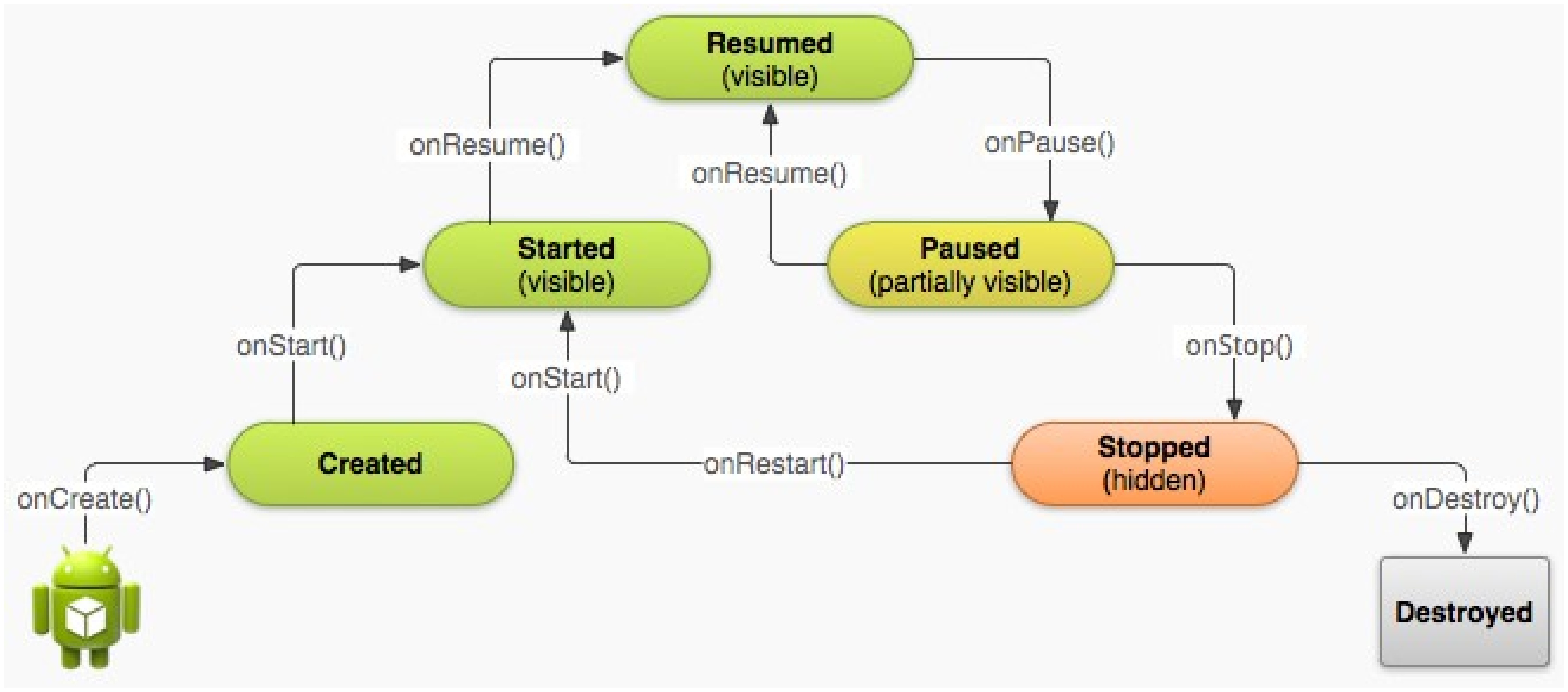
3. Your Activity is stopped if it is completely obscured by another activity.

Although stopped, it continues to retain all its state information.

It is no longer visible to the user (its window is hidden and its life cycle could be terminated at any point by the system if the resources that it holds are needed elsewhere).

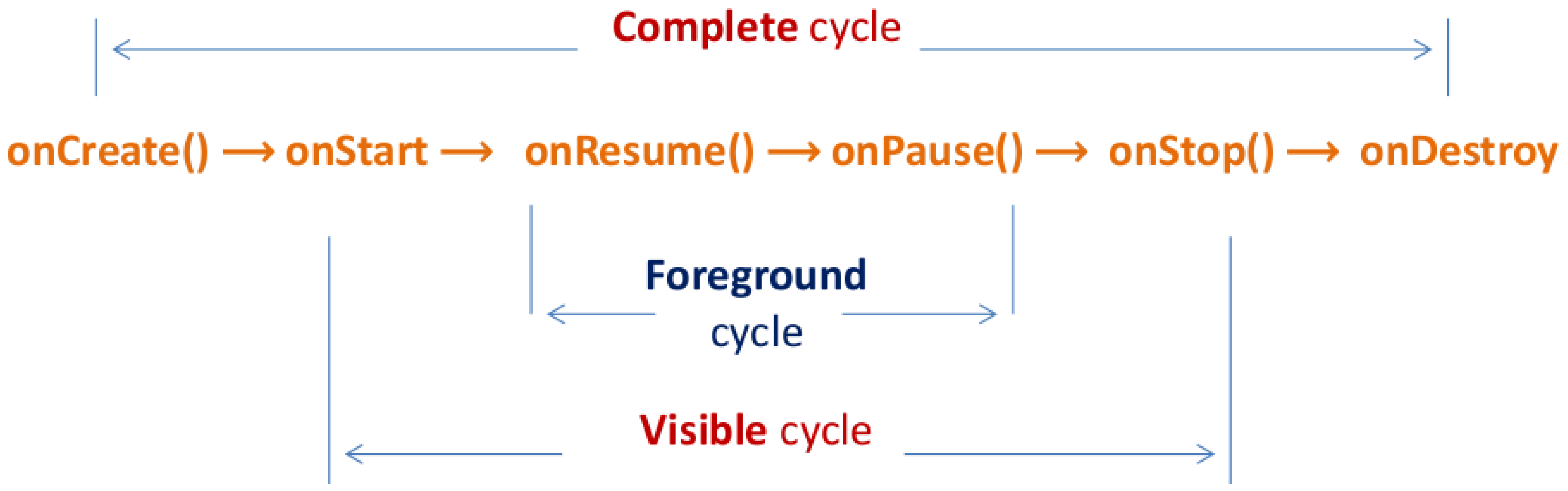


Activity Life Cycle



Foreground Lifetime

- An activity begins its lifecycle when it enters the onCreate() state.
- If it is not interrupted or dismissed, the activity performs its job and finally terminates and releases resources when reaching the onDestroy() event.


Complete cycle
onCreate() ⟶ onStart ⟶ onResume() ⟶ onPause() ⟶ onStop() ⟶ onDestroy
Foreground cycle
Visible cycle

Associating Lifecycle Events with Application’s Code

Applications do not need to implement each of the transition methods, however there are mandatory and recommended states to consider

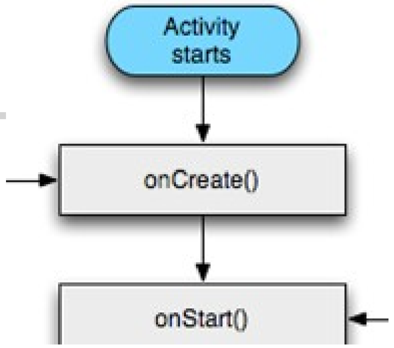
(Mandatory)

onCreate() must be implemented by each activity to do its initial setup. The method is executed only once on the activity’s lifetime.

(Highly Recommended)

onPause() should be implemented whenever the application has some important data to be committed so it could be reused.

Associating Lifecycle Events with Application’s Code

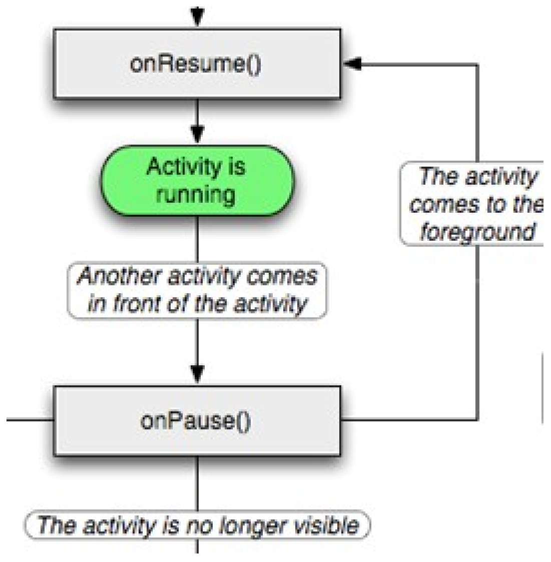


onCreate()

- This is the first callback method to be executed when an activity is created.
- Most of your application’s code is written here.
- Typically used to initialize the application’s data structures, wire-up UI view elements (buttons, text boxes, lists) with local Java controls, define listeners’ behavior, etc.
- It may receive a data Bundle object containing the activity's previous state (if any).
- Followed by onStart() ⟶ onResume() ....

onPause()

1. Called when the system is about to transfer control to another activity. It should be used to safely write uncommitted data and stop any work in progress.
2. The next activity waits until completion of this state.
3. Followed either by onResume() if the activity returns back to the foreground, or by onStop() if it becomes invisible to the user.
4. A paused activity could be killed by the system.



onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
onPause()
The activity is no longer visible

Killable States

- Android OS may terminate a killable app whenever the resources needed to run other operation of higher importance are critically low.
- When an activity reaches the methods: ***onPause()***, ***onStop()***, and ***onDestroy()***, it becomes killable.
- ***onPause()*** is the only state that is guaranteed to be given a chance to complete before the process is terminated.
- You should use ***onPause()*** to write any pending persistent data.

Data Persistence using Android SharedPreferences Class

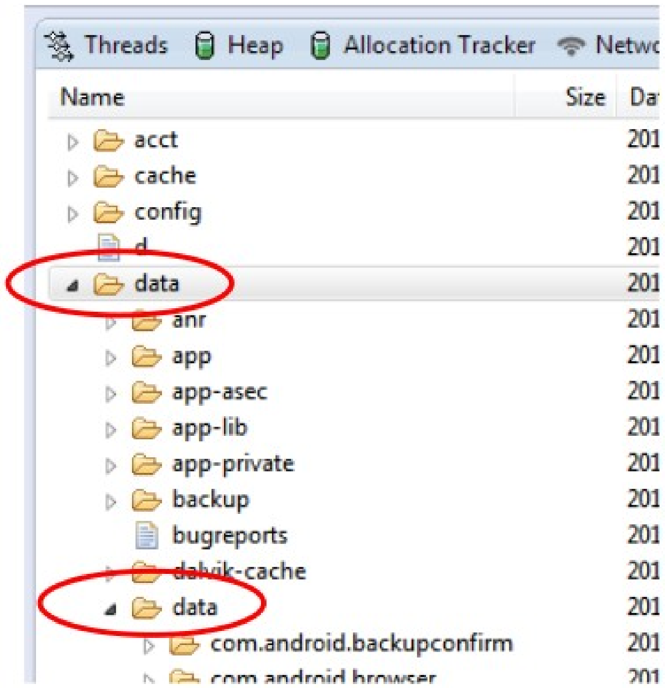
- SharedPreferences is a simple Android persistence mechanism used to store and retrieve <key,value> pairs, where key is a string and value is a primitive data type (int, float, string...).
- This container class reproduces the structure and behavior of a Java HashMap, however; unlike HashMaps it is persistent.
- Appropriate for storing small amounts of state data across sessions.

```
 SharedPreferences myPrefSettings =
    getSharedPreferences(MyPreferrenceFile, actMode);
```

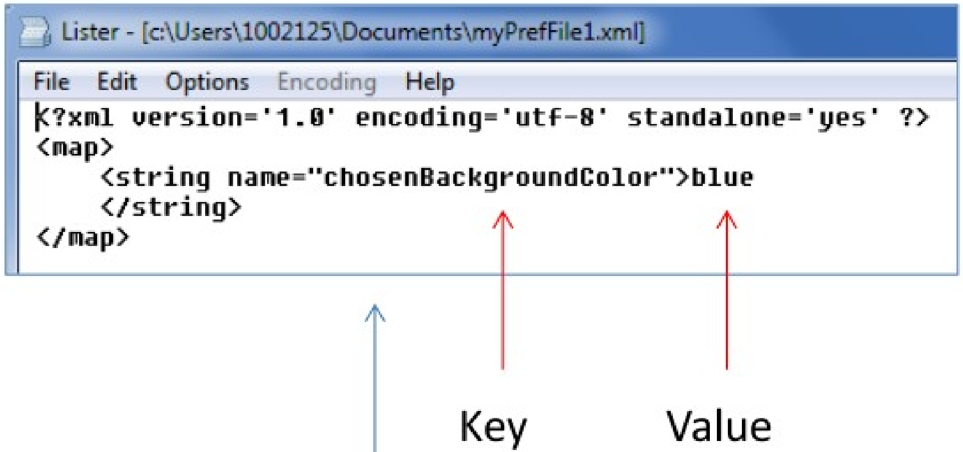
Persistence is an important concept in Android, and it is discussed in more detail latter.

Data Persistence using Android SharedPreferences Class

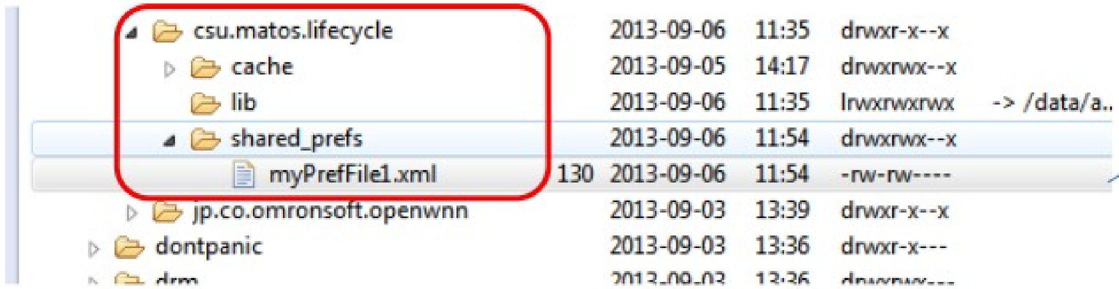
SharedPreference files are permanently stored in the application's process space.
Use DDMS file explorer to locate the entry:
data/data/your-package-name/shared-prefs



```
Name                          Size  Da
▷ acct                              201
▷ cache                             201
▷ config                            201
  d                                 201
◢ data                              201
    ▷ anr                           201
    ▷ app                           201
    ▷ app-asec                      201
    ▷ app-lib                       201
    ▷ app-private                   201
    ▷ backup                        201
      bugreports                    201
    ▷ dalvik-cache                  201
    ◢ data                          201
        ▷ com.android.backupconfirm 201
        ▷ com.android.browser       201
```



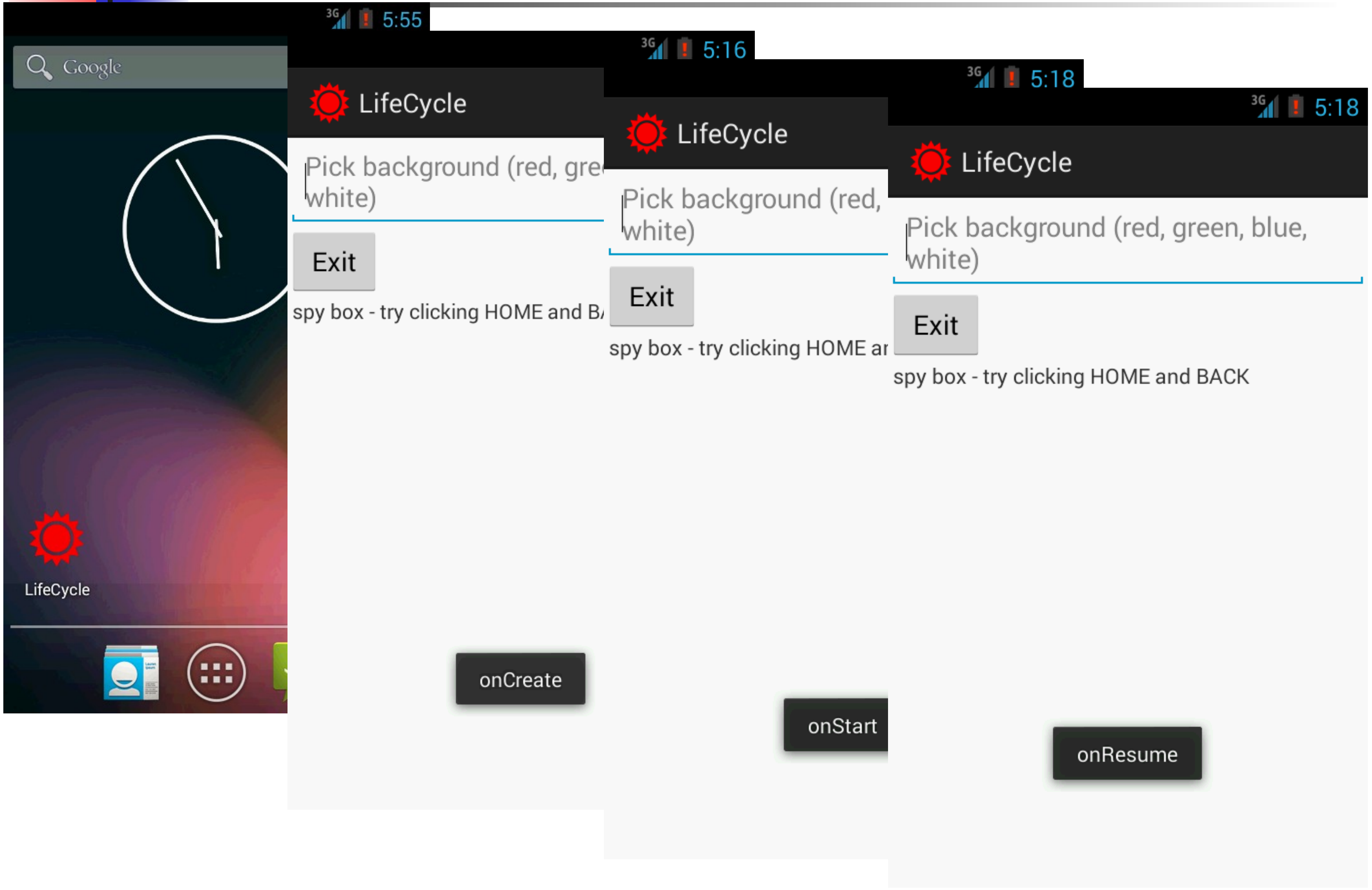
```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="chosenBackgroundColor">blue
    </string>
</map>
```



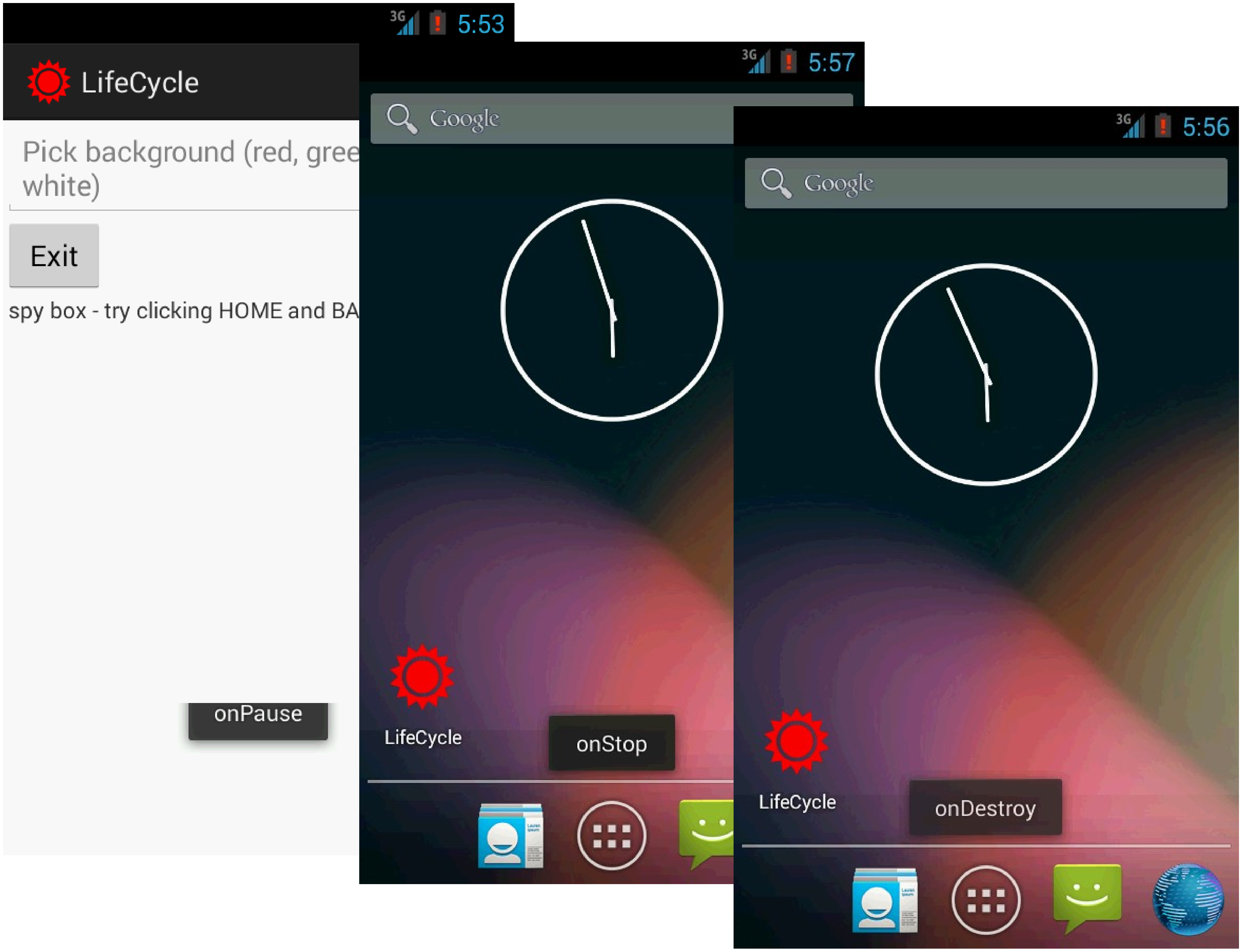
```
◢ csu.matos.lifecycle          2013-09-06  11:35  drwxr-x--x
    ▷ cache                    2013-09-05  14:17  drwxrwx--x
      lib                      2013-09-06  11:35  lrwxrwxrwx   -> /data/a..
    ◢ shared_prefs             2013-09-06  11:54  drwxrwx--x
          myPrefFile1.xml  130 2013-09-06  11:54  -rw-rw----
▷ jp.co.omronsoft.openwnn      2013-09-03  13:39  drwxr-x--x
▷ dontpanic                    2013-09-03  13:36  drwxr-x---
▷ drm                          2013-09-03  13:36  drwxrwx---
```

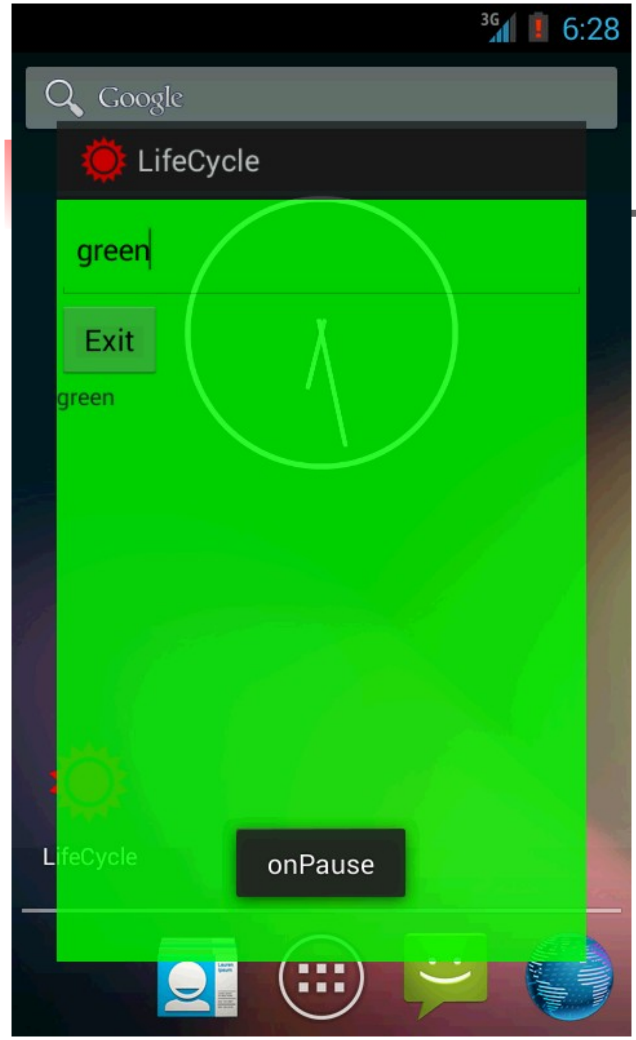
3 - 33

A complete Example: The LifeCycle App

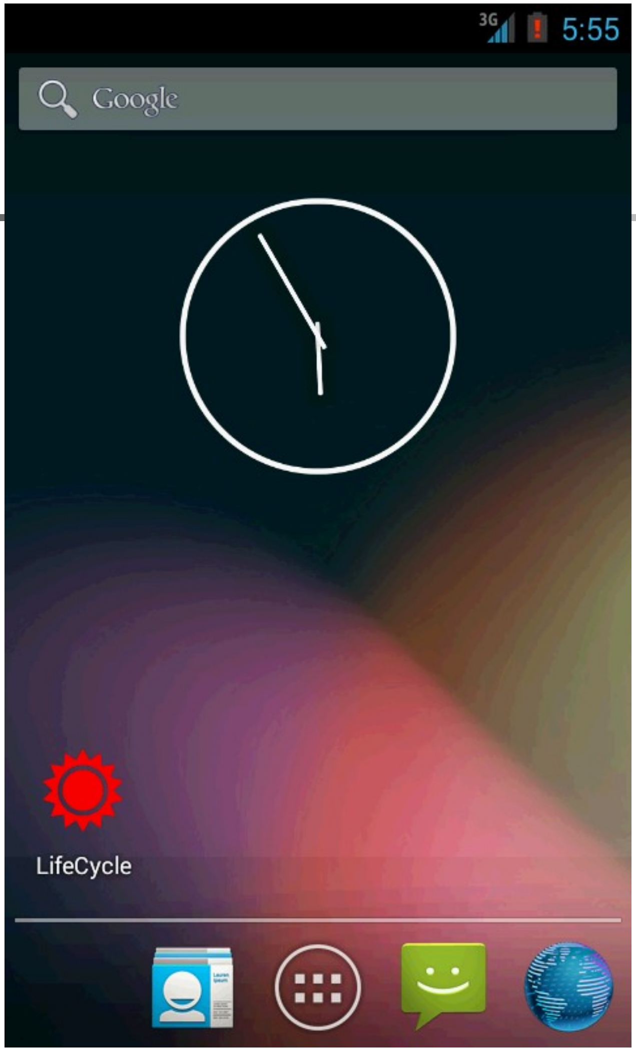


Google
LifeCycle
5:55
LifeCycle
Pick background (red, gre
white)
Exit
spy box - try clicking HOME and B
onCreate
5:16
LifeCycle
Pick background (red,
white)
Exit
spy box - try clicking HOME ar
onStart
5:18
5:18
LifeCycle
Pick background (red, green, blue,
white)
Exit
spy box - try clicking HOME and BACK
onResume

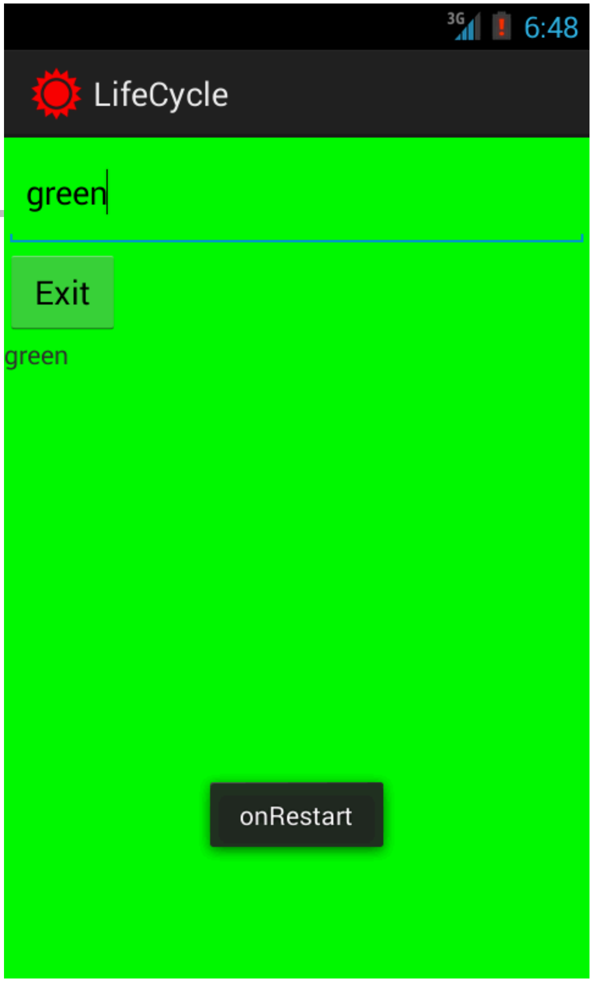

5:53
LifeCycle
Pick background (red, gree
white)
Exit
spy box - try clicking HOME and BA
onPause
5:57
Google
LifeCycle
onStop
5:56
Google
LifeCycle
onDestroy



User selects a green background and clicks the HOME key. When the app is paused, the user's selection is saved, the app is still active but it is not visible.



The app is re-executed



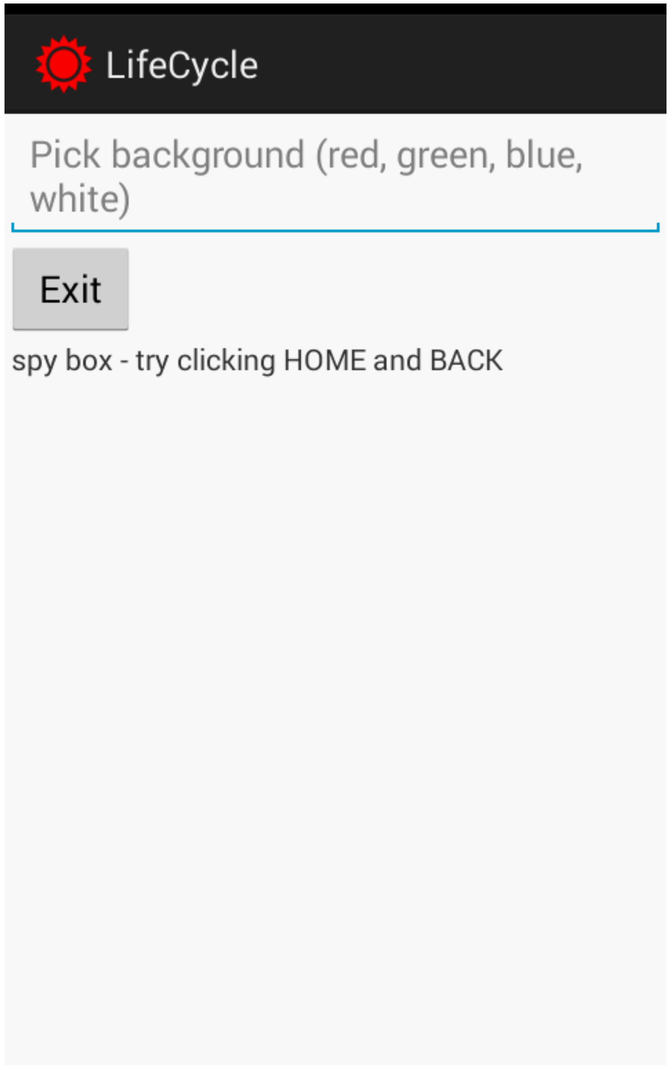
The app is re-started and becomes visible again, showing all the state values previously set by the user (see the text boxes)

A complete Example: The LifeCycle App

The following application demonstrates the transitioning of a simple activity through the Android's sequence of Life-Cycle states.

1. A Toast-msg will be displayed showing the current event's name.
2. An EditText box is provided for the user to indicate a background color.
3. When the activity is paused the selected background color value is saved to a SharedPreferences container.
4. When the application is re-executed, the last choice of background color should be applied.
5. An EXIT button should be provide to terminate the app.
6. You are asked to observe the sequence of messages displayed when the application:
    1. Loads for the first time
    2. Is paused after clicking HOME button
    3. Is re-executed from launch-pad
    4. Is terminated by pressing BACK and its own EXIT button
    5. Re-executed after a background color is set

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.com/tools"
 android:id="@+id/myScreen1"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 android:orientation="vertical"
 tools:context=".MainActivity" >
 <EditText
   android:id="@+id/editText1"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:hint="Pick background (red, green, blue, white)"
   android:ems="10" >
   <requestFocus />
 </EditText>
 <Button
   android:id="@+id/button1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Exit" />
 <TextView
   android:id="@+id/textView1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text=" spy box - try clicking HOME and BACK" />
</LinearLayout>
```


LifeCycle
Pick background (red, green, blue, white)
Exit
spy box - try clicking HOME and BACK

MainActivity.java

```
package csu.matos.lifecycle;
import java.util.Locale;
. . . // other libraries omitted for brevity
public class MainActivity extends Activity {
  // class variables
  private Context context;
  private int duration = Toast.LENGTH_SHORT;
  // PLUMBING: Pairing GUI controls with Java objects
  private Button btnExit;
  private EditText txtColorSelected;
  private TextView txtSpyBox;
  private LinearLayout myScreen;
  private String PREFNAME = "myPrefFile1";
```

MainActivity.java

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  // display the main screen
  setContentView(R.layout.activity_main);
  // wiring GUI controls and matching Java objects
  txtColorSelected = (EditText) findViewById(R.id.editText1);
  btnExit = (Button) findViewById(R.id.button1);
  txtSpyBox = (TextView)findViewById(R.id.textView1);
  myScreen = (LinearLayout)findViewById(R.id.myScreen1);
  // set GUI listeners, watchers,...
  btnExit.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
      finish();
    }
  });
```

```java
//observe (text) changes made to EditText box (color selection)
txtColorSelected.addTextChangedListener(new TextWatcher() {
  @Override
  public void onTextChanged(CharSequence s, int start, int before, int count) {
    // nothing TODO, needed by interface
  }
  @Override
  public void beforeTextChanged(CharSequence s, int start, int count,
      int after) {
    // nothing TODO, needed by interface
  }
  @Override
  public void afterTextChanged(Editable s) {
    // set background to selected color
    String chosenColor = s.toString().toLowerCase(Locale.US);
    txtSpyBox.setText(chosenColor);
    setBackgroundColor(chosenColor, myScreen);
  }
}); // end of TextWatcher
```

onCreate()

onCreate()

```
  //show the current state's name
  context = getApplicationContext();
  Toast.makeText(context, "onCreate", duration).show();
} //onCreate
@Override
protected void onDestroy() {
  super.onDestroy();
  Toast.makeText(context, "onDestroy", duration).show();
}
@Override
protected void onPause() {
  super.onPause();
  // save state data (background color) for future use
  String chosenColor = txtSpyBox.getText().toString();
  saveStateData(chosenColor);
  Toast.makeText(context, "onPause", duration).show();
}
@Override
protected void onRestart() {
  super.onRestart();
  Toast.makeText(context, "onRestart", duration).show();
}
```

```
@Override
protected void onResume() {
  super.onResume();
  Toast.makeText(context, "onResume", duration).show();
}
@Override
protected void onStart() {
  super.onStart();
  // if appropriate, change background color to chosen value
  // updateMeUsingSavedStateData();
  Toast.makeText(context, "onStart", duration).show();
}
@Override
protected void onStop() {
  super.onStop();
  Toast.makeText(context, "onStop", duration).show();
}
```

MainActivity.java

```
private void setBackgroundColor(String chosenColor, LinearLayout
   myScreen) {
  // hex color codes: 0xAARRGGBB AA:transp, RR red,
  // GG green, BB blue
  if (chosenColor.contains("red"))
    myScreen.setBackgroundColor(0xffff0000);
  if (chosenColor.contains("green"))
    myScreen.setBackgroundColor(0xff00ff00);
  if (chosenColor.contains("blue"))
    myScreen.setBackgroundColor(0xff0000ff);
  if (chosenColor.contains("white"))
    myScreen.setBackgroundColor(0xffffffff);
} // end of method setBackgroundColor
```

MainActivity.java

```
private void saveStateData(String chosenColor) {
  // this is a little <key,value> table permanently kept in memory
  SharedPreferences myPrefContainer =
     getSharedPreferences(PREFNAME, Activity.MODE_PRIVATE);
  // pair <key,value> to be stored represents our 'important' data
  SharedPreferences.Editor myPrefEditor =
     myPrefContainer.edit();
  String key = "chosenBackgroundColor";
  String value = txtSpyBox.getText().toString();
  myPrefEditor.putString(key, value);
  myPrefEditor.commit();
} // end of saveStateData
```

MainActivity.java

```
private void updateMeUsingSavedStateData() {
  // (in case it exists) use saved data telling backg color
  SharedPreferences myPrefContainer =
    getSharedPreferences(PREFNAME, Activity.MODE_PRIVATE);
  String key = "chosenBackgroundColor";
  String defaultValue = "white";
  if (( myPrefContainer != null ) &&
    myPrefContainer.contains(key)){
    String color = myPrefContainer.getString(key, defaultValue);
    setBackgroundColor(color, myScreen);
  }
} // updateMeUsingSavedStateData
} // end of MainActivity class
```

Appendix A: Using Bundles to Save/Restore State Values

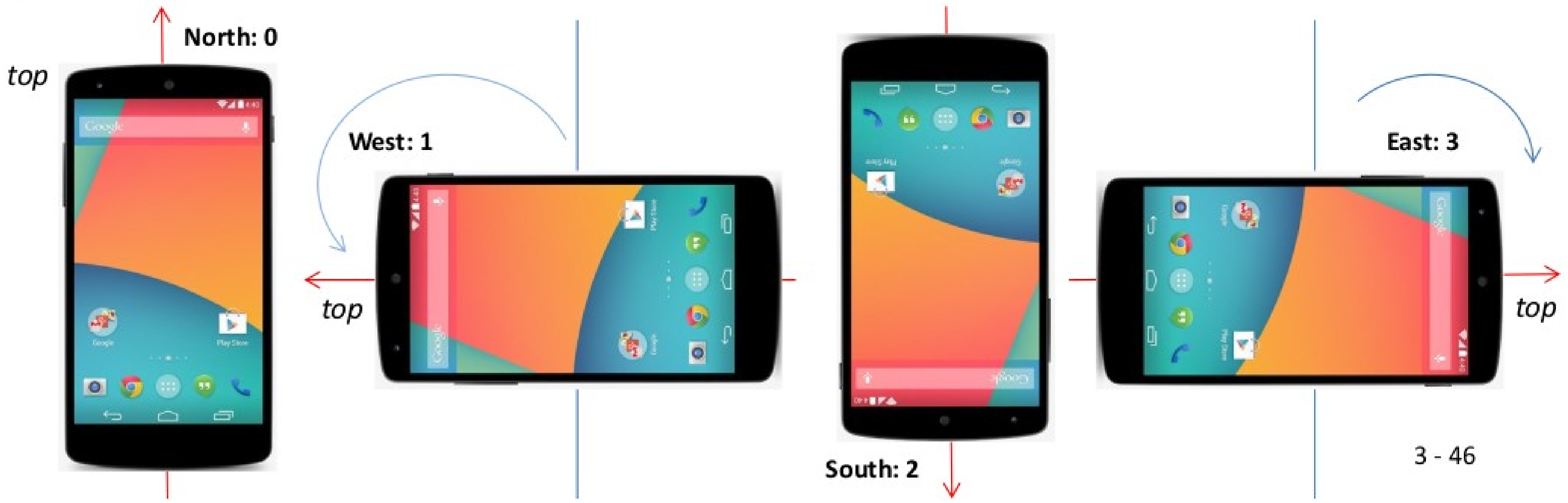
```
@Override
public void onCreate(Bundle savedInstanceState) {
  ...
  if ( savedInstanceState != null )
    String someStrValue = savedInstanceState.getString("STR_KEY",
                                                        "Default");

  ...
}
@Override
public void onSaveInstanceState(Bundle outState) {
  ...
  myBundle.putString("STR_KEY", "blah blah blah");
  onSaveInstanceState( myBundle );
  ...
}
```

Appendix A: Using Bundles to Save/Restore State Values

Note: This approach works well when Android kills the app (like in a device-rotation event), however; it will not create the state bundle when the user kills the app (eg. pressing BackButton).

Hint: It is a better practice to save state using SharedPreferences in the onPause( ) method.

Appendix B: Detecting Device Rotation



Appendix B: Detecting Device Rotation

The function below allows you to obtain the current ORIENTATION of the device as NORTH(0), WEST(1), SOUTH(2) and EAST(3).

```
private int getOrientation() {
  // the TOP of the device points to
  // [0:North, 1:West, 2:South, 3:East]
  Display display = ((WindowManager) getApplication()
    .getSystemService(Context.WINDOW_SERVICE))
    .getDefaultDisplay();
  display.getRotation();
  return display.getRotation();
}
```

Method

Use the onCreate method to initialize a control variable with the original device’s orientation.

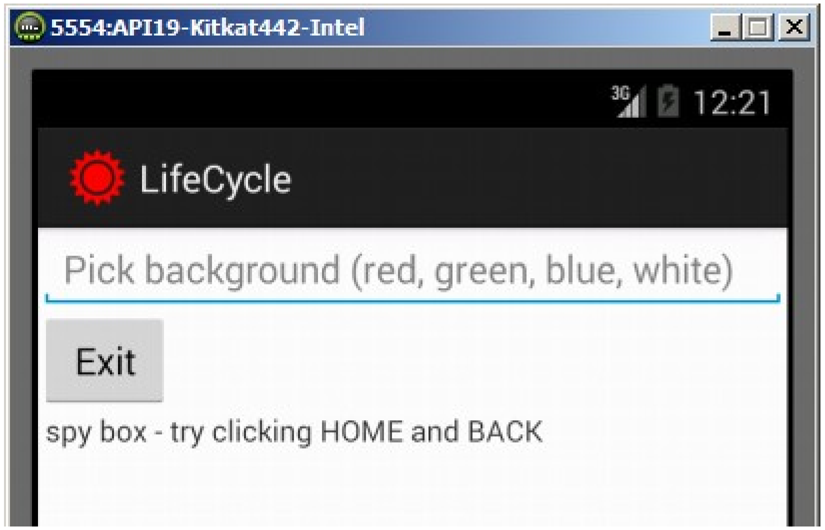
During onPause compare the current orientation with its original value; if they are not the same then the device was rotated.

```
int originalOrientation;
// used to detect orientation change
@Override
protected void onCreate(Bundle savedInstanceState) {
  ...
  setContentView(R.layout.activity_main);
  originalOrientation = getOrientation();
  ...
}
@Override
protected void onPause() {
  super.onPause();
  If ( getOrientation() != originalOrientation ){
    // Orientation changed - phone was rotated
    // put a flag in outBundle, call onSaveInstanceState(...)
  } else {
    // no orientation change detected in the session
  }
}
```

Bài thực hành 1

Transitioning: One State at the Time

1. Create an Android app (LifeCycle) to show the different states traversed by an application.
2. The activity_main.xml layout should include an EditText box (txtMsg), a button (btnExit), and a TextView (txtSpy). Add to the EditText box the hint depicted in the figure on the right.


5554:API19-Kitkat442-Intel
12:21
LifeCycle
Pick background (red, green, blue, white)
Exit

spy box - try clicking HOME and BACK

Bài thực hành 1 (tiếp)

3. Use the onCreate method to connect the button and textbox to the program. Add the following line of code:

```
Toast.makeText(this, "onCreate",
  Toast.LENGTH_SHORT ).show();
```

4. The onClick method has only one command: finish(); called to terminate the application.

5. Add a Toast-command (as the one above) to each of the remaining six main events. Top menu: Source > Override/Implement Methods... (look for callback methods). On the Option-Window check mark each of the following events: onStart, onResume, onPause, onStop, onDestry, onRestart (notice how many onEvent... methods are there!!!) .

6. Save your code.

Bài thực hành 1 (tiếp)

7. Compile and execute the application.
8. Write down the sequence of messages displayed using the Toast-commands.
9. Press the EXIT button. Observe the sequence of states displayed.
10. Re-execute the application
11. Press emulator’s HOME button. What happens?
12. Click on launch pad, look for the app’s icon and return to the app. What sequence of messages is displayed?
13. Click on the emulator’s CALL (Green phone). Is the app paused or stopped?
14. Click on the BACK button to return to the application.
15. Long-tap on the emulator’s HANG-UP button. What happens?

Lab Experience 2

Calling & Texting Emulator-to-Emulator

7. Run a second emulator.
   a. Make a voice-call to the first emulator that is still showing our app. What happens on this case? (real-time synchronous request)
   b. Send a text-message to first emulator (asynchronous attention request)
8. Write a phrase in the EditText box: “these are the best moments of my life....”.
9. Re-execute the app. What happened to the text?

Lab Experience 3

Provide data persistence .

16. Use the **onPause** method to add the following fragment:

```
SharedPreferences myFile1 = getSharedPreferences("myFile1",
  Activity.MODE_PRIVATE);
SharedPreferences.Editor myEditor = myFile1.edit();
String temp = txtMsg.getText().toString();
myEditor.putString("mydata", temp);
myEditor.commit();
```

17. Use the **onResume** method to add the following fragment:

```
SharedPreferences myFile = getSharedPreferences("myFile1",
  Activity.MODE_PRIVATE);
if ( (myFile != null) && (myFile.contains("mydata")) ) {
  String temp = myFile.getString("mydata", "***");
  txtMsg.setText(temp);
}
```

18. What happens now with the data previously entered in the text box?

Next: Android-Lesson04-User-Interfaces