

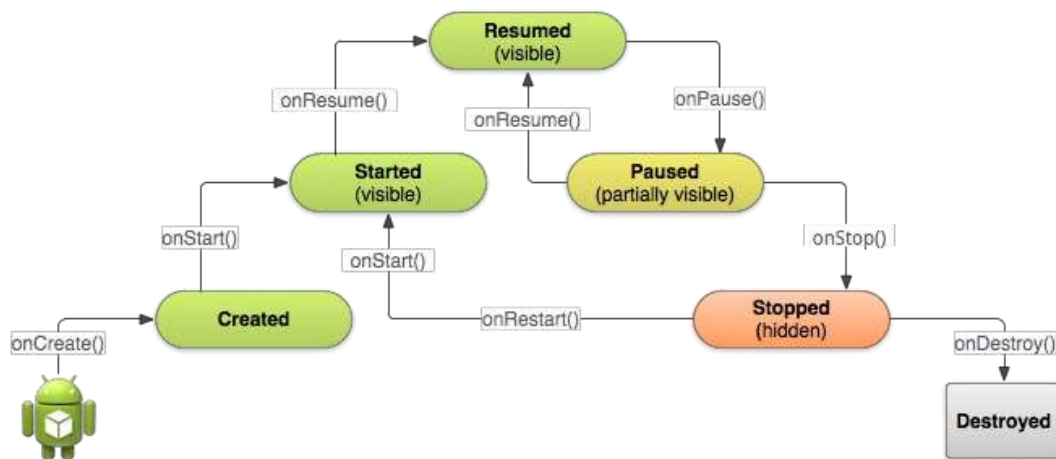
# CSCI3310 Mobile Computing & Application Development

## Lab 3 – Data Persistence: Save My Number

### Introduction

In this lab, you learn more about the *activity lifecycle*.

As a user navigates between activities in your app (as well as into and out of your app), activities transition between different states in their lifecycles.



By overriding any of the lifecycle callback methods in your **Activity** classes, you can change the activity's default behavior in response to user or system actions.

The activity state can also change in response to device-configuration changes, for example when the user rotates the device from portrait to landscape. When these configuration changes happen, the activity is destroyed and recreated in its default state, and the user might lose information that they've entered in the activity. To avoid confusing your users, it's important that you develop your app to prevent unexpected data loss. Later in this lab, you experiment with configuration changes and learn how to preserve an activity's state in response to device configuration changes and other activity lifecycle events.

In the first part of this lab, you add logging statements to the app and observe activity lifecycle changes as you use the app. You then begin working with these changes and exploring how to handle user input under these conditions. In the second part, you save and retrieve data for persistence.

### Objective

- 1) How the **Activity** lifecycle works. When an **Activity**: About the lifecycle callback methods associated with **Activity** starts, pauses, stops, and is destroyed.
- 2) The effect of actions (such as configuration changes) that can result in **Activity** lifecycle events. How to retain **Activity** state across lifecycle events.

- 3) How to create a shared preferences file to save data to them and read those preferences back again.


## Todo

- 1) Implement the various **Activity** lifecycle callbacks to include logging statements and observe the state changes as you interact with each **Activity** in your app.
- 2) Modify your app to retain the instance state of an **Activity** that is unexpectedly recreated in response to user behavior or a configuration change on the device.
- 3) Update an app so it can save, retrieve, and reset shared preferences.

## 1: Open the Magic Number project

For the tasks in this lab, you will reuse the code from the previous lab to fix its abnormal behavior.

### 1.1 Start with the Magic Number project

- 1) Download the **Magic Number Starter** Project file from Blackboard, you may rename the directory back to **Magic Number** if you prefer but it doesn't affect the package name.
- 2) Open the **Magic Number** project.
- 3) Run the app as usual and test the "+" and "-" **Button** elements to check if the number in the middle can be updated accordingly.
- 4) Update the number to "4".
- 5) Click  to rotate the screen and you should obtain the following screen:



Now, the number goes back to "1". This is because the **MainActivity** is re-created due to a change of device configuration. It would be a rather unexpected effect but that's how Android is managing the Activity in device status update and you shall see how to cope with it in the rest of this lab.

## 2. Add lifecycle callbacks

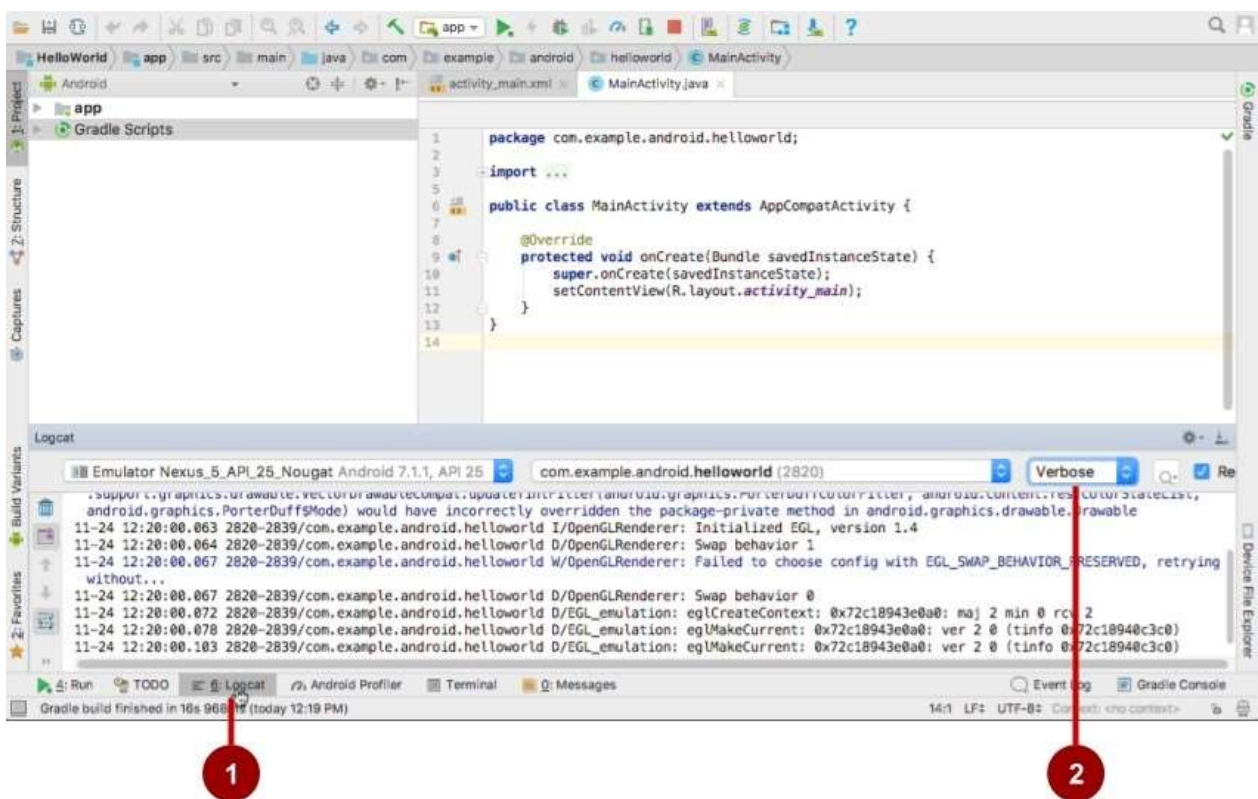
In this task, you will implement all of the **Activity** lifecycle callback methods to print messages to logcat when those methods are invoked.

These log messages will allow you to see when the **Activity** lifecycle changes state, and how those lifecycle state changes affect your app as it runs.

### 2.1 View the Logcat pane

You will add **Log** statements to your app, which display messages in the **Logcat** pane. Log messages are a powerful debugging tool that you can use to check on values, execution paths, and report exceptions.

To see the **Logcat** pane, click the **Logcat** tab at the bottom of the Android Studio window as shown in the figure below.



In the figure above:

- 1) The **Logcat** tab for opening and closing the **Logcat** pane, which displays information about your app as it is running. If you add Log statements to your app, Log messages appear here.
- 2) The Log level menu set to **Verbose** (the default), which shows all Log messages. Other settings include **Debug**, **Error**, **Info**, and **Warn**.

### 2.2 Add log statements to your app

Log statements in your app code display messages in the Logcat pane. For example:

```
Log.d("MainActivity", "Magic Number");
```

The parts of the message are:

- 1) **L**og: The [Log](#) class for sending log messages to the Logcat pane.
- 2) **d**: The **Debug** Log level setting to filter log message display in the Logcat pane. Other log levels are **e** for **Error**,
- 3) **w**: for **Warn**, and
- 4) **i**: for **Info**.

"MainActivity": The first argument is a tag that can be used to filter messages in the Logcat pane. This is commonly the name of the [Activity](#) from which the message originates.

However, you can make this anything that is useful to you for debugging.

By convention, log tags are defined as constants for the [Activity](#):

```
private static final String LOG_TAG = MainActivity.class.getSimpleName();
```

- "Magic Number": The second argument is the actual message.

Follow these steps:

- 5) Open [MainActivity](#).
- 6) In the [onCreate\(\)](#) method of [MainActivity](#), add the following statement:

```
Log.d("MainActivity", "Magic Number");
```

The [onCreate\(\)](#) method should now look like the following code:

```
@Override protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    Log.d("MainActivity", "Magic Number");  
}
```

- 7) If the Logcat pane is not already open, click the **Logcat** tab at the bottom of Android Studio to open it.
- 8) Check that the name of the target and package name of the app are correct.
- 9) Change the Log level in the **Logcat** pane to **Debug** (or leave as **Verbose** since there are so few log messages).
- 10) Run your app.

The following message should appear in the Logcat pane:

```
1-29 15:06:59.001 4696-4696/? D/MainActivity: Magic Number
```

You may apply a filter of "D/MainActivity" to limit the display if too many other Logs are displayed.

## 2.3 Implement callbacks into MainActivity

- 1) Open MainActivity in the Project > Android pane.
- 2) In the [onCreate\(\)](#) method, add the following log statements:

```
Log.d(LOG_TAG, "-----");
Log.d(LOG_TAG, "onCreate");
```

3) Add an override for the `onStart()` callback, with a statement to the log for that event:

```
@Override
public void onStart(){
    super.onStart();
    Log.d(LOG_TAG, "onStart");
}
```

For a shortcut, select **Code > Override Methods** in Android Studio. A dialog appears with all of the possible methods you can override in your class. Choosing one or more callback methods from the list inserts a complete template for those methods, including the required call to the superclass.

4) Use the `onStart()` method as a template to implement the `onPause()`, `onRestart()`, `onResume()`, `onStop()`, and `onDestroy()` lifecycle callbacks.

All the callback methods have the same signatures (except for the name). If you **Copy** and **Paste** `onStart()` to create these other callback methods, don't forget to update the contents to call the right method in the superclass, and to log the correct method.

The following code snippets show the added code in `MainActivity`, but not the entire class.

The `onCreate()` method:

```
@Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // Log the start of the onCreate() method.
    Log.d(LOG_TAG, "-----");
    Log.d(LOG_TAG, "onCreate");
    // Initialize all the view variables if any
    //. . . some views, e.g. textView should be inflated here
}
```

The other lifecycle methods:

```
@Override
protected void onStart() {
    super.onStart();
    Log.d(LOG_TAG, "onStart");
}

@Override protected void onPause() {
    super.onPause();
    Log.d(LOG_TAG, "onPause");
}

@Override
protected void onRestart() {
    super.onRestart();
    Log.d(LOG_TAG, "onRestart");
}

@Override protected void onResume() {
    super.onResume();
    Log.d(LOG_TAG, "onResume");
}
```

```

@Override
protected void onStop() {
    super.onStop();
    Log.d(LOG_TAG, "onStop");
}

@Override
protected void onDestroy() {
    super.onDestroy();
    Log.d(LOG_TAG, "onDestroy");
}

```

- 5) Run your app.
- 6) Click the **Logcat** tab at the bottom of Android Studio to show the **Logcat** pane. You should see three log messages showing the three lifecycle states the Activity has transitioned through as it started:

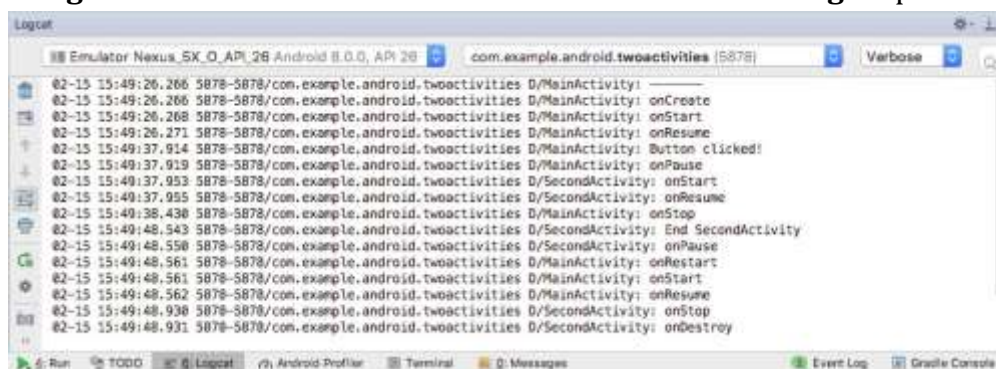
```

D/MainActivity: -----
D/MainActivity: onCreate
D/MainActivity: onStart
D/MainActivity: onResume

```

## 2.4 Observe the log as the app runs

- 1) Run your app.
- 2) Click the **Logcat** tab at the bottom of Android Studio to show the **Logcat** pane.



- 3) Enter **Activity** in the search box. The Android logcat can be very long and cluttered. Because the LOG\_TAG variable in each class contains either the words **MainActivity** or **DetailedViewActivity**, this keyword lets you filter the log for only the things you're interested in.

Experiment using your app and note that the lifecycle events that occur in response to different actions. In particular, try these things:

- 4) Use the app normally (send a message, reply with another message).
- 5) Use the Back button to go back from the second **Activity** to the main **Activity**.
- 6) Use the Up arrow in the app bar to go back from the second **Activity** to the main **Activity**.

- 7) Rotate the device on both the main and second **Activity** at different times in your app and observe what happens in the log and on the screen.
- 8) Press the overview button (the square button to the right of Home) and close the app (tap the **X**). Return to the home screen and restart your app.

**TIP:** If you're running your app in an emulator, you can simulate rotation with Control+F11 or Control+Function+F11.

### 3: Save and restore the Activity instance state

Depending on system resources and user behavior, each **Activity** in your app may be destroyed and reconstructed far more frequently than you might think.

You may have noticed this behavior in the last section when you rotated the device or emulator. Rotating the device is one example of a device *configuration change*. Although rotation is the most common one, all configuration changes result in the current **Activity** being destroyed and recreated as if it were new. If you don't account for this behavior in your code when a configuration change occurs, your **Activity** layout may revert to its default appearance and initial values, and your users may lose their place, their data, or the state of their progress in your app.

The state of each **Activity** is stored as a set of key/value pairs in a **Bundle** object called the *Activity instance state*. The system saves default state information to instance state **Bundle** just before the **Activity** is stopped, and passes that **Bundle** to the new **Activity** instance to restore.

To keep from losing data in an Activity when it is unexpectedly destroyed and recreated, you need to implement the `onSaveInstanceState()` method. The system calls this method on your **Activity** (between `onPause()` and `onStop()`) when there is a possibility the **Activity** may be destroyed and recreated.

The data you save in the instance state is specific to only this instance of this specific **Activity** during the current app session. When you stop and restart a new app session, the **Activity** instance state is lost and the **Activity** reverts to its default appearance. If you need to save user data between app sessions, use shared preferences or a database. You learn about them later.

#### 3.1 Save the Activity instance state with `onSaveInstanceState()`

You may have noticed that rotating the device does not affect the state of the second **Activity** at all. This is because the second **Activity** layout and state are generated from the layout and the Intent that activated it. Even if the **Activity** is recreated, the Intent is still there and the data in that Intent is still used each time the `onCreate()` method in the second **Activity** is called.

In addition, you may notice that in each **Activity**, any text you typed into a message or reply **EditText** elements is retained even when the device is rotated. This is because the state



information of some of the [View](#) elements in your layout is automatically saved across configuration changes, and the current value of an [EditText](#) is one of those cases.

So the only [Activity](#) state you're interested in is the [TextView](#) elements for the reply header and the reply text in the main [Activity](#). Both [TextView](#) elements are invisible by default; they only appear once you send a message back to the main [Activity](#) from the second [Activity](#).

In this task, you add code to preserve the instance state of these two [TextView](#) elements using [onSaveInstanceState\(\)](#).

- 1) Open MainActivity.
- 2) Add this skeleton implementation of [onSaveInstanceState\(\)](#) to the [Activity](#), or use **Code > Override Methods** to insert a skeleton override.

```
@Override
public void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
}
```

- 3) Inside that same check, add the guess number text into the [Bundle](#) before it's being saved.

```
outState.putString("GUESS_NUMBER", mShowNumber.getText().toString());
```

Note that the system will save the state of some [View](#) elements, such as the contents of the [EditText](#) since the content in it is managed by the system.

You only have saved the state of only those [View](#) elements that might change after the [Activity](#) is created. Those [View](#) elements in your app, such as an [EditText](#), or a [Button](#), can be recreated from the default layout at any time.

### 3.2 Restore the Activity instance state in onCreate()

Once you've saved the [Activity](#) instance state, you also need to restore it when the [Activity](#) is recreated. You can do this either in [onCreate\(\)](#), or by implementing the [onRestoreInstanceState\(\)](#) callback, which is called after [onStart\(\)](#) after the [Activity](#) is created.

Most of the time the better place to restore the [Activity](#) state is in [onCreate\(\)](#), to ensure that the UI, including the state, is available as soon as possible. It is sometimes convenient to do it in [onRestoreInstanceState\(\)](#) after all of the initialization has been done, or to allow subclasses to decide whether to use your default implementation.

- 1) In the [onCreate\(\)](#) method, after the [View](#) variables are initialized with [findViewById\(\)](#), add a test to make sure that [savedInstanceState](#) is not [null](#).

```
// Initialize all the view variables.
// ...
// Restore the state.
if (savedInstanceState != null) {
}
```



When your `Activity` is created, the system passes the state `Bundle` to `onCreate()` as its only argument. The first time `onCreate()` is called and your app starts, the `Bundle` is null—there's no existing state the first time your app starts. Subsequent calls to `onCreate()` have a bundle populated with the data you stored in `onSaveInstanceState()`.

- 2) Inside that check, get the current guess number out of the `Bundle` with the key `"GUESS_NUMBER"`.

```
if (savedInstanceState != null) {  
    mShowNumber.setText(savedInstanceState.getString("GUESS_NUMBER"));  
}
```

- 3) Run the app. Try rotating the device or the emulator to ensure that the guess number retains its value on the screen after the `Activity` is recreated.

If the following code snippets show the added code in `MainActivity`, but not the entire class.

The `onSaveInstanceState()` method:

```
@Override  
public void onSaveInstanceState(Bundle outState) {  
    outState.putString("GUESS_NUMBER", mShowNumber.getText().toString());  
    super.onSaveInstanceState(outState);  
}
```

The `onCreate()` method:

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    Log.d(LOG_TAG, "-----");  
    Log.d(LOG_TAG, "onCreate");  
    // Initialize all the view variables.  
    // ...  
    // Restore the saved state.  
    // See onSaveInstanceState() for what gets saved.  
    if (savedInstanceState != null) {  
        mShowNumber.getText(savedInstanceState.getString("GUESS_NUMBER"));  
    }  
}
```

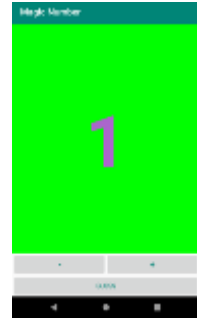
## 4. (Optional) Save and restore data to a shared preferences file

In this task, you save the state of the app to a shared preferences file and read that data back in when the app is restarted.

Now test what happens when you quit and restart the app:

- 1) Force-quit the app using the following method:

- 2) On the device, press the Recents button (the square button in the lower right corner). Swipe the app card to quit the app, or click the X in the right corner of the card. If you quit the app in this manner, wait a few seconds before starting it again so the system can clean up.
- 3) Re-run the app. The app restarts with the default—the Guess Number is 1.



#### 4.1 Initialize the preferences

- 4) Add member variables to the `MainActivity` class to hold the name of the shared preferences file, and a reference to a `SharedPreferences` object as well as the key for storing `Guess Number`.

```
private SharedPreferences mPreferences;  
private String sharedPrefFile = "edu.cuhk.csci3310.magicnumber";  
private final String GUESS_NUMBER_KEY = "guess_number";
```

You can name your shared preferences file anything you want to, but conventionally it has the same name as the package name of your app.

- 5) In the `onCreate()` method, initialize the shared preferences. Insert this code before the `if` statement:

```
mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);
```

The `getSharedPreferences()` method (from the activity `Context`) opens the file at the given filename (`sharedPrefFile`) with the mode `MODE_PRIVATE`.

**Note:** Older versions of Android had other modes that allowed you to create a world-readable or world-writable shared preferences file. These modes were deprecated in API 17, and are now **strongly discouraged** for security reasons. If you need to share data with other apps, you can use content URIs provided by a [FileProvider](#).

Code for `MainActivity` should partially look like:

```
public class MainActivity extends AppCompatActivity {  
    private int mGuessNumber = 1;  
    private TextView mShowNumber;  
    private SharedPreferences mPreferences;  
    private String smSharedPrefFile = "edu.cuhk.csci3310.magicnumber";  
    private final String GUESS_NUMBER_KEY = "guess_number";  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        mShowNumber = (TextView) findViewById(R.id.text_count);  
        mPreferences = getSharedPreferences(  
            sharedPrefFile, MODE_PRIVATE);  
        // ...  
    }  
}
```

## 4.2 Save preferences in onPause()

Saving preferences is a lot like saving the instance state -- both operations set aside the data to a Bundle object as a key/value pair. For shared preferences, however, you save that data in the `onPause()` lifecycle callback, and you need a shared editor object ([SharedPreferences.Editor](#)) to write to the shared preferences object.

- 1) Add the `onPause()` lifecycle method to `MainActivity`.

```
@Override
protected void onPause(){
    super.onPause();
    // ...
}
```

- 2) In `onPause()`, get an editor for the `SharedPreferences` object:

```
SharedPreferences.Editor preferencesEditor = mPreferences.edit();
```

A shared preferences editor is required to write to the shared preferences object. Add this line to `onPause()` after the call to `super.onPause()`.

- 3) Use the `putInt()` method to put the `mGuessNumber` integer into the shared preferences with the appropriate keys:

```
preferencesEditor.putInt(GUESS_NUMBER_KEY, mGuessNumber);
```

The `SharedPreferences.Editor` class includes multiple "put" methods for different data types, including `putInt()` and `putString()`.

- 4) Call `apply()` to save the preferences:

```
preferencesEditor.apply();
```

The `apply()` method saves the preferences asynchronously, off of the UI thread. The shared preferences editor also has a `commit()` method to synchronously save the preferences. The `commit()` method is discouraged as it can block other operations.

- 5) Delete the entire `onSaveInstanceState()` method. Because the activity instance state contains the same data as the shared preferences, you can replace the instance state altogether.

Code for `MainActivity onPause()` method should look like:

```
@Override
protected void onPause(){
    super.onPause();
    SharedPreferences.Editor preferencesEditor = mPreferences.edit();
    preferencesEditor.putInt(GUESS_NUMBER_KEY, mGuessNumber);
    preferencesEditor.apply();
}
```

## 4.3 Restore preferences in onCreate()

As with the instance state, your app reads any saved shared preferences in the `onCreate()` method. In the `onCreate()` method, in the same spot where the instance state code was, get the `Guess Number` from the preferences with the `GUESS_NUMBER_KEY` key and assign it to the `mGuessNumber` variable.

```
mGuessNumber = mPreferences.getInt(GUESS_NUMBER_KEY, 1);
```

When you read data from the preferences you don't need to get a shared preferences editor. Use any of the "get" methods on a shared preferences object (such as [getInt\(\)](#) or [getString\(\)](#) to retrieve preference data.

Note that the [getInt\(\)](#) method takes two arguments: one for the key, and the other for the default value if the key cannot be found. In this case, the default value is 1, which is the same as the initial value of `mGuessNumber`.

1) Update the value of the main `TextView` with the new `Guess Number`.

```
mShowNumber.setText(String.format("%s", mGuessNumber));
```

2) Run the app. Click the "+" or "-" button to update the instance state and the preferences.

3) Rotate the device or emulator to verify that the Guess Number is saved across configuration changes.

4) Force-quit the app:

On the device, press the Recents button (the square button in the lower right corner). Swipe the `Magic Number` app card to quit the app, or click the X in the right corner of the card.

5) Re-run the app. The app restarts and loads the preferences, maintaining the state.

Code for `MainActivity onCreate()` method, should partially look like:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // Initialize views, preferences
    mGuessNumberTextView = (TextView) findViewById(R.id.text_count);
    mPreferences = getSharedPreferences(mSharedPrefFile, MODE_PRIVATE);
    // Restore preferences
    mGuessNumber = mPreferences.getInt(GUESS_NUMBER_KEY, 1);
    mShowNumber.setText(String.format("%s", mGuessNumber));
}
```

## 4.4 Reset preferences in a reset() click handler

Add a reset button in the starter app resets the guess number for the activity to their default values. Because the preferences hold the state of the activity, it's important to also clear the preferences at the same time.

1) Add a `reset() onClick` method, after the guess number is reset, get an editor for the `SharedPreferences` object:

```
SharedPreferences.Editor preferencesEditor = mPreferences.edit();
```

2) Delete all the shared preferences:

```
preferencesEditor.clear();
```

3) Apply the changes:

```
preferencesEditor.apply();
```

## Reference

Android developer documentation:

- 1) [Application Fundamentals](#)
- 2) [Activities](#)
- 3) [Understand the Activity Lifecycle](#)
- 4) [Intents and Intent Filters](#)
- 5) [Handle configuration changes](#)
- 6) [Activity Intent](#)
- 7) [Save key-value data](#)
- 8) [SharedPreferences](#)
- 9) [SharedPreferences.Editor](#)

Stack Overflow:

- 1) [Sending data back to the Main Activity in Android](#)
- 2) [How to use SharedPreferences in Android to store, fetch and edit values onSavedInstanceState vs. SharedPreferences](#)