08309 Lab 4 Threading and Saving State

Goal

In this lab you will learn how to use a background thread to download files from the internet and save them to internal storage. You will also learn how to save small amounts of data to shared preferences. At the same time, you'll finish a push your luck game called Pirate Booty!

In this game you can choose to drink grog, or go on a raid and plunder. Grog is readily available, and the application keeps a total of how much you've drunk. It even displays a ***toast*** to tell you how much you've had. The plunder button doesn't work yet. It should generate a random number, and based on that number you should either execute a successful raid or die trying. Either way, an appropriate picture is downloaded from the internet and display, along with a score if appropriate.

Check out the folder **<your repository>**\Week4\Lab_4_1_PirateBooty to C:\Temp and open this application in Android Studio. When you've finished remember to commit your changes and delete your work from the local drive. Run the application so you can see what is going on. There is just one activity and associated class in the project, so all code happens in the MainActivity.java file.

Setting up the gameplay mechanic

Successful pirates are bold. In order to make themselves bold they drink grog. Drinking a little grog will make them more successful, but if they drink too much they may become reckless. The amount of grog drunk before plundering represents a simple risk/reward mechanic.

When you click plunder, we want to generate a random number between zero and twenty. If the random number is more than the amount of grog you've drank you get a score equal to that same number multiplied by the amount of grog you've drank.

Create a datamember of type Random

```
private Random m_RNG = new Random();
```

In the plunderOnClick method add the following code

```
int randomNumber = m_RNG.nextInt(20);
if(randomNumber > m_NumberOfGrogs) {
    addCurrentScore(randomNumber * m_NumberOfGrogs);
}
else
{
    TextView textView = (TextView)findViewById(R.id.currentScoreLabelTextView);
    textView.setText(R.string.fail_label);
    m_Score = 0;
    m_NumberOfGrogs = 0;
}
```

As described earlier, a random number is generated. If that random number is greater than the number of grogs you have drank then you get treasures equal to the random number multiplied by the number of grogs you have drank. If not you lose, and the score and number of grogs are reset to zero.

Check it works and familiarise yourself with the game. When you've done that check your code in to you repository with an appropriate message like *"Completed lab 4.1 task 1 completed the basic gameplay mechanic"*.

Adding a picture

As it is the game is a little dull. To improve it we're going to add a picture of treasure for when the player is winning, and a picture of a skull for when they are losing. We're going to download these pictures from the internet.

Add the following code to the bottom of the plunderOnClick method.

```
ImageView imageView = (ImageView)findViewById(R.id.imageView);
Bitmap bitmap = null;
try {
    bitmap = BitmapFactory.decodeStream((InputStream) new URL(urlString).getContent());
} catch (IOException e) {
    Log.i("My Exception", e.getMessage());
}
imageView.setImageBitmap(bitmap);
```

You will have to declare a string called urlString to download the content. You should set it to http://www.hull.ac.uk/php/349628/08027/labs/treasure.jpg for treasure or set it to http://www.hull.ac.uk/php/349628/08027/labs/death.jpg for death.

Run the code and you'll find that the program crashes. If you trawl through the host of messages that android dumps into logcat you'll find an exception has been thrown called android.os.NetworkOnMainThreadException.

To fix this issue we're going to have to move the networking part onto another thread using an AsyncTask object. Create a class inside the MainActivity class called downloadImage that extends the AsyncTask class. Give it a doInBackground method, which will be called on the new thread and deals with the networking part of the task. Next give it an onPostExecute method that will run on the UI thread and assigns the bitmap returned from doInBackground to an imageView that already exists in the main activity.

```
private class downloadImage extends AsyncTask<String, String, Bitmap> {

    protected Bitmap doInBackground(String... args) {
        Bitmap bitmap = null;
        try {
            bitmap = BitmapFactory.decodeStream((InputStream)new URL(args[0]).getContent());

        } catch (Exception e) {
            e.printStackTrace();
        }
        return bitmap;
    }

    protected void onPostExecute(Bitmap image) {

        if(image != null){
            ImageView imageView = (ImageView)findViewById(R.id.imageView);
            imageView.setImageBitmap(image);
        }else{
            Toast.makeText(MainActivity.this, "Image Does Not exist or Network Error", Toast.LENGTH_SHORT).show();
        }
    }
}
```

In the plunderOnClick method create a new instance of this class and call execute on it, passing in the urlString.

```
new downloadImage().execute(urlString);
```

Run the code. This time it won't crash, but it will tell us that either the image does not exist or there was a network error. Although it is unusual to ever commit broken code to a repository go ahead and commit this with an appropriate message like *"Completed lab 4.1 task 2 created an asynctask to download an image from the internet – not working yet"*.

To find out what is wrong put a break point in the doInBackground method and step through the code. If you do that it should be obvious what the issue is, and how to fix it. Once you have run the code and as you play the game you should see the pictures appear.

Commit your code with an appropriate message like *"Completed lab 4.1 task 3 found out that the application did not have the correct permissions and added them to the manifest.xml file"*.

So far we've started a new thread by extending asynctask. As this application is a single activity we can get away with some poor thread management, but you should be aware that if the application were to stop we might want to clean up after ourselves by cancelling the thread at the appropriate point in the activity life cycle, and checking to see if the thread is cancelled in doInBackground to finish early if necessary.

Fixing the High Score

Note that at the moment our high score isn't working. To rectify this we need to add an int data member called m_HighScore to our activity. We need to initialise it to zero and then add the following code to the end of the addCurrentScore method.

```
if(m_Score > m_HighScore)
{
    m_HighScore = m_Score;
    setHighScore(m_HighScore);
}
```

Check that this works, then commit your code to your repository using a message like *"Completed lab 4.1 task 4 added high score functionality"*.

Saving State with Shared Preferences

Now we have a high score, but this high score won't persist if the application is stopped. You should also note that if you play the game and amass some treasure, then rotate the emulator all your treasure is lost. To fix this we will store the high score in shared preferences. First, create a private String data member called HIGH_SCORE and set it to be high_score. We will use this string as a key in the key value pairs stored in shared preferences.

Next in the onCreate method get a sharedPreferences object and set m_HighScore to be the whatever value is associated with the HIGH_SCORE key we just defined.

```
SharedPreferences sharedPreferences = getPreferences(MODE_PRIVATE);
m_HighScore = sharedPreferences.getInt(HIGH_SCORE, 0);
```

Finally, whenever we set a new high score we need to update our value in shared preferences. In the setHighScore method create a SharedPreference.Editor object and add our high score value to it.

```
SharedPreferences.Editor editor = getPreferences(MODE_PRIVATE).edit();
editor.putInt(HIGH_SCORE, m_HighScore);
editor.commit();
```

Run the code and check that your high score is persisting by building up a high score and then changing the orientation of the phone. If everything is working commit your changes to subversion with an appropriate message like *"Completed lab 4.1 task 5 saved high score in shared preferences"*.

You should note that whilst your high score is being saved the treasure you have amassed is not. Repeat the process to save the amount of treasure, and the amount of grog you have drank. You should do this in the activities onPause method, and rebuild these in the onStart method.

Once you've done that commit your changes to subversion with an appropriate message like *"Completed lab 4.1 task 6 saved and reloaded amount of treasure and grog using shared preferences"*.

Saving to Internal Storage

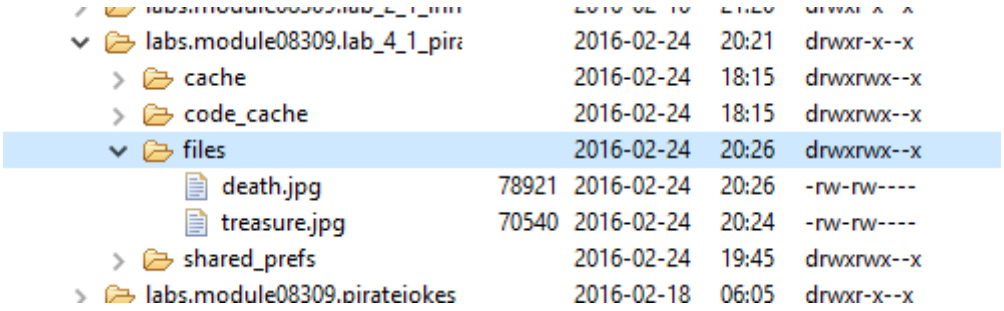
You've probably realised by now that downloading an image file from the internet every time the plunder button is clicked is a little inefficient. To avoid this before downloading the file we're going to check to see if it exists on the devices internal storage. If it does we will load it from there, and if not we can download it and save it.

The first step is to adapt the current solution so that only the filename is passed to the doInBackground method. The rest of the url is then constructed inside that method if necessary.

Then, inside doInBackground we want to try to load the requested file from internal storage. If the file doesn't exist an exception will be thrown. We can catch that exception and then download the requested file and then save that to internal storage for next time.

```
protected Bitmap doInBackground(String... args) {
    Bitmap bitmap = null;
    try {
        // Try to read from internal storage
        FileInputStream reader = getApplicationContext().openFileInput(args[0]);
        bitmap = BitmapFactory.decodeStream(reader);
    }
    catch (FileNotFoundException fileNotFound) {
        try {
            // if file was not found try to download it and store it for next time
            String url = "http://www.hull.ac.uk/php/349628/08027/labs/" + args[0];
            bitmap = BitmapFactory.decodeStream((InputStream) new URL(url).getContent());
            FileOutputStream writer = null;
            try {
                writer = getApplicationContext().openFileOutput(args[0], Context.MODE_PRIVATE);
                bitmap.compress(Bitmap.CompressFormat.JPEG, 100, writer);
            } catch (Exception e) {
                Log.i("My Error", e.getMessage());
            } finally {
                writer.close();
            }
        } catch (Exception e) {
            Log.i("My Error", e.getMessage());
        }
    }
    return bitmap;
}
```

Once you've done that you can check to see if it works. You can view the file system in internal storage through the DDMS tool. Click on file explorer and find the appropriate directory in data/data/your.app.name/



You should be able to see when the new files are created.

Once you've done that commit your changes to subversion with an appropriate message like *"Completed lab 4.1 task 7 saved and reloaded files using internal storage"*.

It's probably worth noting that loading images from internal storage for every button press also is not very efficient. Think about what you might do to avoid that.

Summary

In this lab you've completed a highly entertaining push your luck game. You've created an asynctask object to download pictures from the internet on a background thread. You've saved application state using shared preferences, and saved and reloaded files to internal storage.