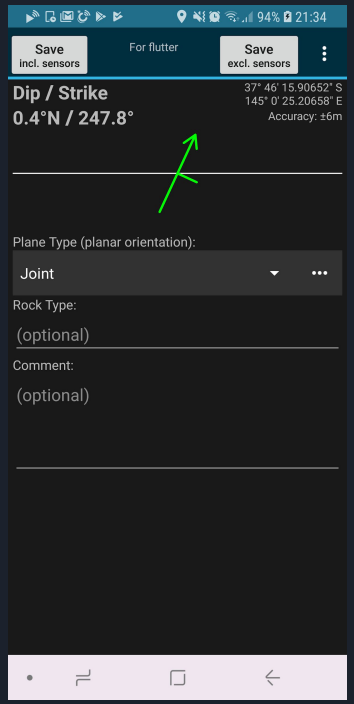
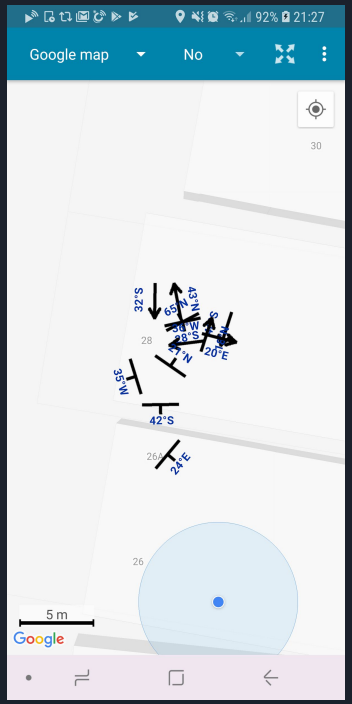
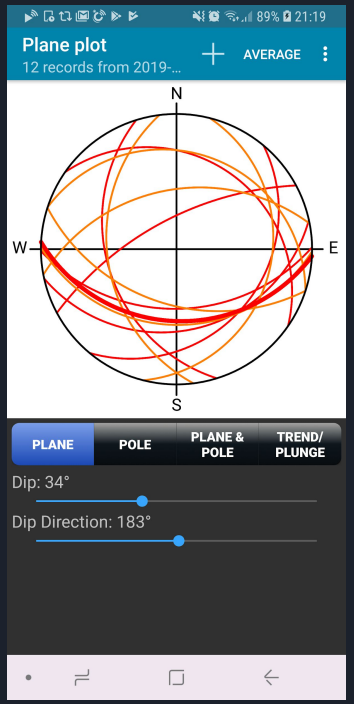
Porting an Android app to Flutter

Bramley Turner-Jones
rockgecko.com

Rocklogger Android







Aim

- Learn Flutter
- Publish Rocklogger for iOS
- 1:1 copy of existing app
- Rearchitecting only where it makes sense - primarily UI into stateless widgets
- Navigation the same
- Core data classes & business logic basically exactly the same - for easier maintenance of both codebases

Start with R.string

```
<string name="l_save_dip_dir_toast">"Saved %1$.1f° / %2$s (dip / d
<string name="l_save_dip_strike_toast">"Saved %1$s / %2$s (dip / s
<string name="l_save_magnetic_toast">"Saved %1$.1fuT, with %2$s"</
<string name="l_save_no_dip_toast">"Saved text information only, w
<string name="l_save_photo_toast">"Photo %1$s saved, with %2$s"</s
<string name="l_angle">"Dip angle: "</string>
<string name="l_direction">"Dip direction: "</string>
<string name="l_strike">"Strike: "</string>
```

```
class _Strings {
  final String l_save_dip_dir_toast =
      "Saved %.1f° / %s (dip / direction) with %s";
  final String l_save_dip_strike_toast = "Saved %s / %s (dip / 
  final String l_save_magnetic_toast = "Saved %.1fuT, with %s";
  final String l_save_no_dip_toast = "Saved text information on
  final String l_save_photo_toast = "Photo %s saved, with %s";
  final String l_angle = "Dip angle: ";
  final String l_direction = "Dip direction: ";
  final String l_strike = "Strike: ";
```

```
class R {
  static _Strings string = _Strings();
  static _Arrays array = _Arrays();
  static _Colors color = _Colors();
  static void setLocale(Locale l) {
    switch (l.languageCode) {
      case "FR":
        string = _Strings_FR();
        break;
      default:
        string = _Strings();
        break;
    }
```

Using R.string and getString()

- `sprintf` package, which formats `%.1f` strings
- Create a simple `getString` function with optional positional parameters, put them into a list and pass to `sprintf`

```dart
import 'package:sprintf/sprintf.dart';

//Note: outside of a class body. When this file is imported
//getString becomes available with no qualification
String getString(String str, [Object p1, Object p2, Object p3, Object p4]) {
  List<Object> params = [p1, p2, p3, p4].takeWhile((p) => p != null).toList();
  return sprintf(str, params);
}
```

Which one’s which?

```
String getToastTextByMode(DipMode loggingMode) {
  if (loggingMode == DipMode.DIP_DIRECTION &&
      (this.mode == DipMode.DIP_BOTH ||
          this.mode == DipMode.DIP_DIRECTION)) {
    //"Saving %1$.1f° dip, direction: %2$s with %3$s"
    return getString(R.string.l_save_dip_dir_toast, dipAngle,
        getDipDirectionStringHR(), super.getLocationToastText());
  }
  else if (loggingMode == DipMode.DIP_STRIKE &&
       (this.mode == DipMode.DIP_BOTH ||
           this.mode == DipMode.DIP_STRIKE)) {
      //"Saving %1$s, striking %2$s with %3$s"
      return getString(R.string.l_save_dip_strike_toast, dipAngleStrike,
          getStrikeStringHR(), super.getLocationToastText());
  }
  else {
      throw "DipDirectionStrikeRecord: getToastTextByMode( ) "
      + "not defined for type " + this.mode.toString();
  }
}
```

```
String getToastTextByMode(DipMode loggingMode, Context c){
      if (loggingMode==DipMode.DIP_DIRECTION &&
              (this.mode== DipMode.DIP_BOTH
               || this.mode == DipMode.DIP_DIRECTION)){
          //"Saving %1$.1f° dip, direction: %2$s with %3$s"
          return c.getString(R.string.l_save_dip_dir_toast, dipAngle,
              getDipDirectionStringHR(), super.getLocationToastText(c));
      }
      else if (loggingMode==DipMode.DIP_STRIKE &&
              (this.mode== DipMode.DIP_BOTH ||
                  this.mode == DipMode.DIP_STRIKE)){
          //"Saving %1$s, striking %2$s with %3$s"
          return c.getString(R.string.l_save_dip_strike_toast, dipAngleStrike,
              getStrikeStringHR(), super.getLocationToastText(c));
      }
      else{
          throw new UnsupportedOperationException(
            "DipDirectionStrikeRecord: getToastTextByMode( ) "
              +"not defined for type " + this.mode.toString());
      }
}
```

Similar for R.color

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="marker_green">#ff23CF34</color>
    <color name="marker_orange">#ff7f00</color>


    <color name="body_text_1">#ff000000</color>
    <color name="body_text_2">#555555</color>
    <color name="body_text_3">#A2A2A2</color>
```

```dart
class _Colors {
  final Color marker_green = Color(0xff23CF34);
  final Color marker_orange = Color(0xffff7f00);

  final Color body_text_1 = Color(0xff000000);
  final Color body_text_2 = Color(0xff555555);
  final Color body_text_3 = Color(0xffA2A2A2);
```

Convert core logic from Java to Dart

1. Convert all `float` to `double`, `boolean` to `bool`, arrays to fixed-size `List`
2. Remove all access modifiers. Dart only has public and private. After converting the class, use refactor->rename to add an underscore for private functions.
3. To be more idomatic, remove getters and setters

Convert core logic from Java to Dart

```
//Explicit arrays:
//java
float[] result = new float[] {dipAngle, dipDirection};

//dart
//(can also use var)
List<double> result = [dipAngle, dipDirection];

//Empty arrays:
//java
float[] empty = new float[2];

//dart
//a fixed-size list contains only nulls by default,
//need to use the .filled named constructor
var empty = List<double>.filled(2, 0);
```

Wins

- Main logging screen code 2.5x smaller, not even counting Android's separate layout.xml file. Other screens code over 4x smaller
- `setState` and stateless widgets make ui state management so much simpler to reason about, compared to trying to update different parts of the UI in different places at different times trying to be more efficient
- ⚡
- Launching the camera and getting the photo back as a file could not be easier:

```
File image = await ImagePicker.pickImage(source: ImageSource.camera);
```

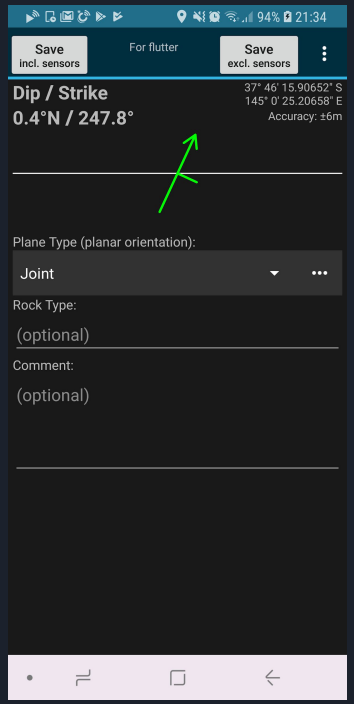
- `await` is also fantastic anywhere you'd use `startActivityForResult`, or for the selected button from an alert dialog

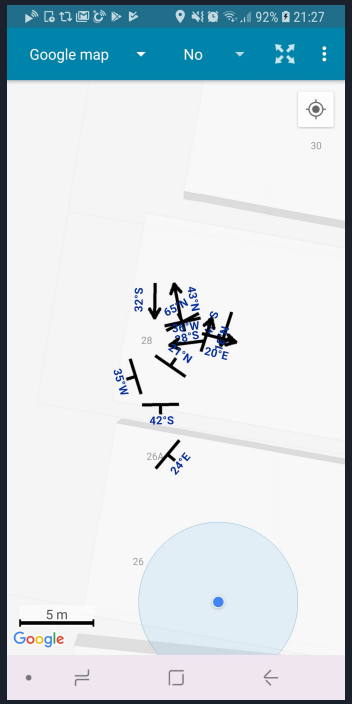
Problems with compiler type checking

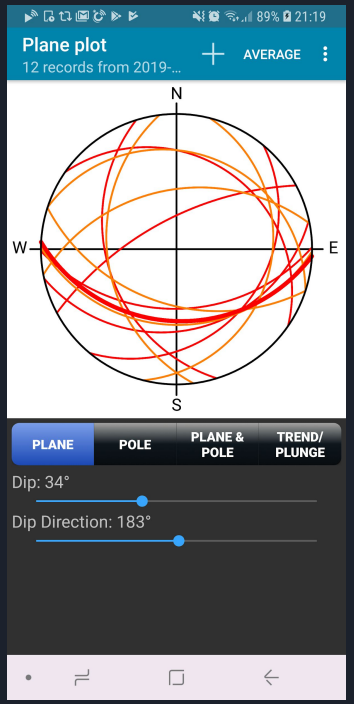
```
bool includeCats = doYouWantCats();
List<Animal> animals = [
  new Mouse(),
  new Alligator(),
  includeCats ? new Cat() : null
];
animals = animals.where((a) => a != null); //Runtime CRASH:
//type 'WhereIterable<Animal>' is not a subtype of type 'List<Animal>'
//Solution: add .toList()

List<Mouse> mice = animals.whereType<Mouse>().toList();
List<Animal> miceUpcast = mice; //compiles. Handy!
List<Mouse> animalsDowncast = animals; //Runtime CRASH:
//type 'List<Animal>' is not a subtype of type 'List<Mouse>'
```

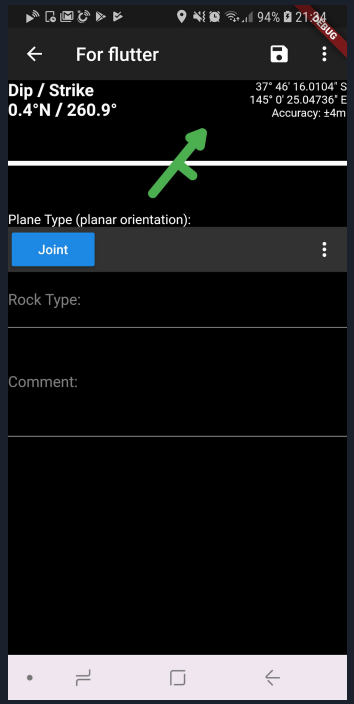
Rocklogger Android

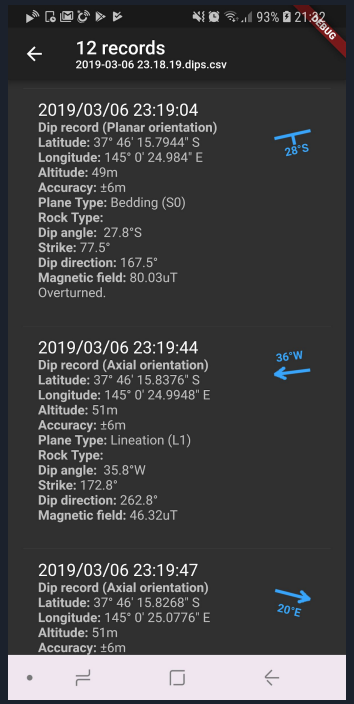


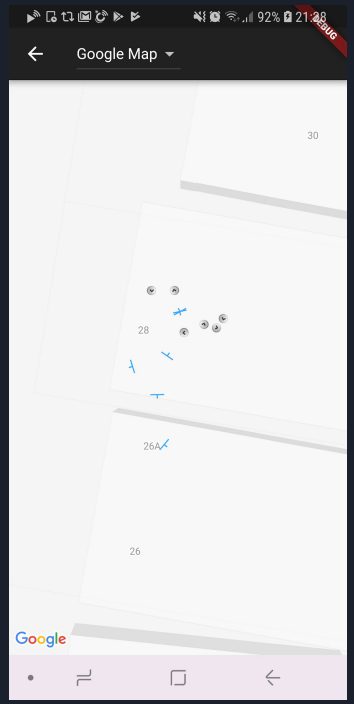


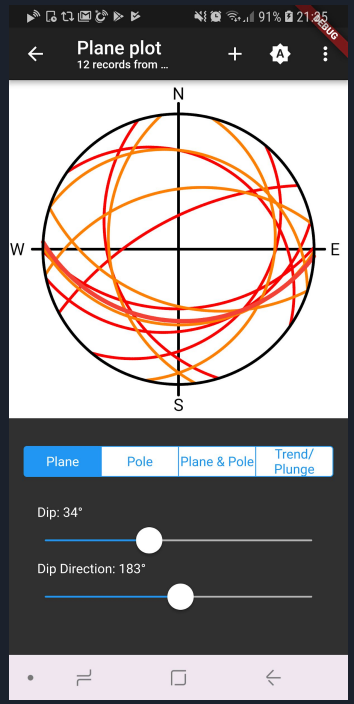


Rocklogger Flutter









Not Ported

- Android wear app (can’t communicate with iOS)
- Alternate basemaps (waiting for improved Google Maps plugin)
- Ability to screenshot & print stereonet plot