https://github.com/JermNet/MOBI

You are going to sell your own Calculator Android App (or select a different Android project), on the Android "Play Store"; but first, you need to upgrade your calculator code to the "AOSP Java Code Style for Contributors" standard.

- AOSP Java Code Style for Contributors: https://source.android.com/setup/contribute/code-style
- Kotlin Style Guide: https://developer.android.com/kotlin/style-guide

1. Identify and fix your code (fix includes properly done code)
2. [5% per fix, max 40%] Show at least 6 different rules that are properly implemented.

```
public class MusicPlayer {  2 usages

    // RULE 1: Originally I had this in the main activity, not as private, so the first changed rule I have
     implemented is information hiding from "Java Code Review Check List"
    private MediaPlayer mediaPlayer;  3 usages

    public void playSound(View view, int file) {  2 usages
        mediaPlayer = MediaPlayer.create(view.getContext(), file);
        mediaPlayer.start();
    }
```

My first implemented rule is from the Check List, that being information hiding. By making the media player private when it wasn't originally, I've implemented this rule.

```
// RULE 2: I had a redundant cast here, so me removing it was following the "remove unused code" rule since
 it was pointless. Same goes with some other scattered code and casts.
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main_activity_screen_size);
    TextView scrollable = findViewById(R.id.textView);
    scrollable.setMovementMethod(new ScrollingMovementMethod());
}
```

Around my code, there were a few bits of code that weren't doing anything. In this example, I was casting the text view to text view. There wasn't a specific rule for this, but I placed it under the "don't leave in unused code" rule since I felt that made the most sense.

```
// RULE 3: Having this + the music player follows the rule of let things only do what they're expected to do. It was
  described as having no "God classes"
```

Originally I had everything in one class, but I've split it into 3 separate classes. One of the rules described this as "having no God classes."

```
// RULE 4: This follows the rule of not having a bunch of repeated code, in the original these were four
 separate methods, but this works to alleviate that somewhat.

// RULE 5: This throws an exception instead of catching, and then avoiding it, which was explained in the AOSP
 guide.
public void calculate(View view, String operand) throws ArithmeticException {  1 usage
    Log.d( tag: operand + " BUTTON",  msg: "User tapped the " + operand + " button.");
    double1 = Double.parseDouble(textN1.getText().toString());
    double2 = Double.parseDouble(textN2.getText().toString());

    switch (operand) {
        case "+":
            answer = double1 + double2;
            break;
        case ("-"):
            answer = double1 - double2;
            break;
        case "*":
            answer = double1 * double2;
            break;
        case "/":
            answer = double1 / double2;
            break;
        default:
            Log.d( tag: "BUTTON",  msg: "Not a valid operand");
    }
    Log.w( tag: operand + " BUTTON",  msg: "Selected with => " + double1 + operand + double2 + "=" + answer);
    textANS.setText(answer.toString());
}
```

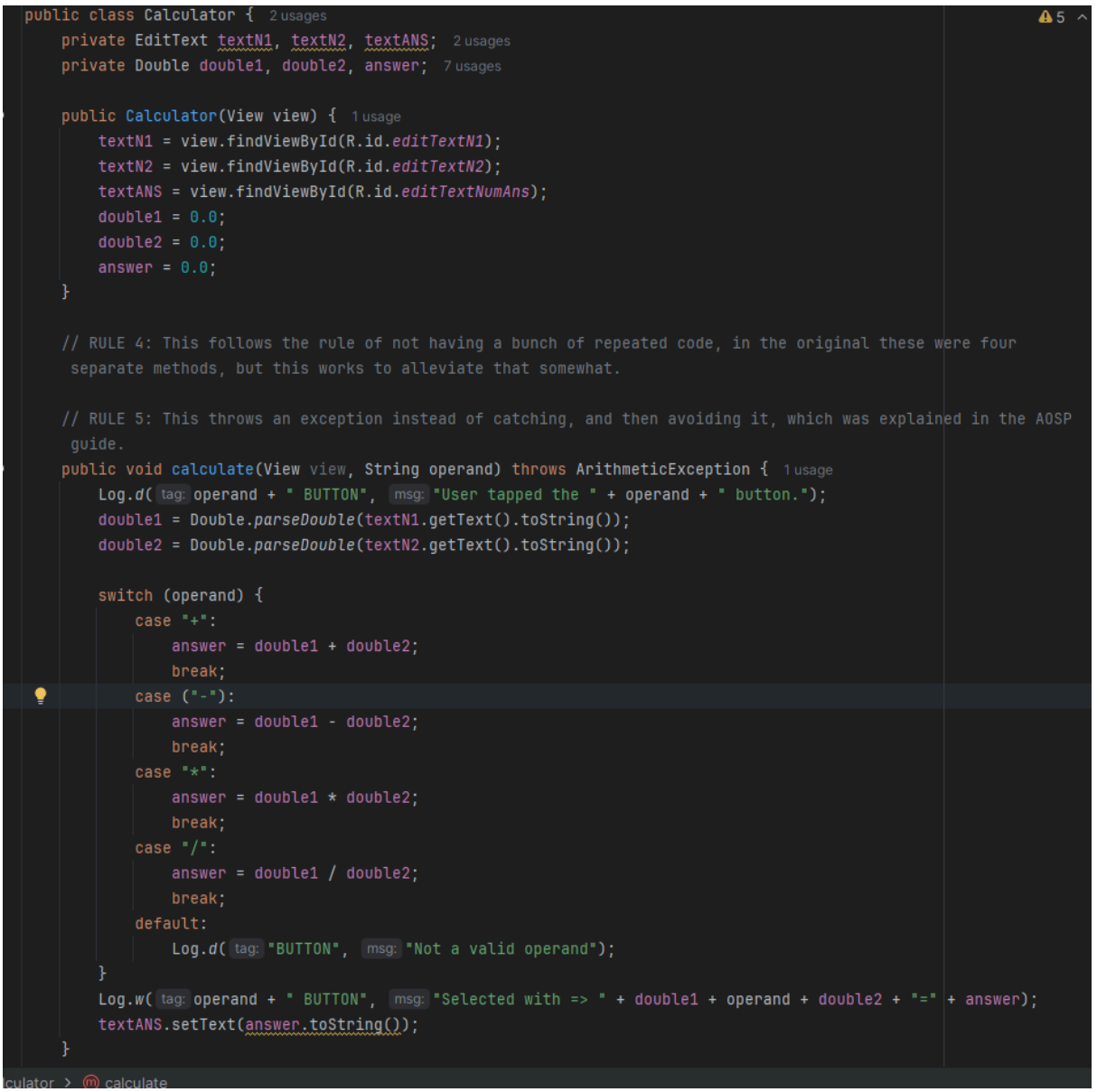
Originally, this was 4 separate methods, but using a switch, I was able to cut this down and not have a bunch of repeated code, one of the important rules in coding in general, but also for this specifically.

```
// RULE 5: This throws an exception instead of catching, and then avoiding it, which was explained in the AOSP
 guide.
public void calculate(View view, String operand) throws ArithmeticException {  1 usage
```

In the AOSP style guide, it mentioned that exceptions should not be ignored. So, I replaced my original try catches with this throws ArithmeticException since the only errors this should produce is dividing by zero or not putting in any numbers in the text fields.

```
// RULE 6: The final rule I've made changes to this code to follow is having the author name at the tops of all
 files (plus a very scary legal warning)!
/**
 * Author: Jeremy Whitenect
 * Year: 2024
 * This code is the intellectual property of Jeremy Whitenect. If this code is stolen, used for monetary gain,
  or reproduced without permission, you will be prosecuted to the fullest extent of the law.
 */
public class MainActivityScreenSize extends AppCompatActivity {
```

Finally, for the rules, I've added this copyright notice, my name, and year the code was made to the Java files to comply with the very first rule in the Check List file.



```
public class Calculator {  2 usages                                                                    ⚠ 5
    private EditText textN1, textN2, textANS;  2 usages
    private Double double1, double2, answer;  7 usages

    public Calculator(View view) {  1 usage
        textN1 = view.findViewById(R.id.editTextN1);
        textN2 = view.findViewById(R.id.editTextN2);
        textANS = view.findViewById(R.id.editTextNumAns);
        double1 = 0.0;
        double2 = 0.0;
        answer = 0.0;
    }

    // RULE 4: This follows the rule of not having a bunch of repeated code, in the original these were four
     separate methods, but this works to alleviate that somewhat.

    // RULE 5: This throws an exception instead of catching, and then avoiding it, which was explained in the AOSP
     guide.
    public void calculate(View view, String operand) throws ArithmeticException {  1 usage
        Log.d( tag: operand + " BUTTON",  msg: "User tapped the " + operand + " button.");
        double1 = Double.parseDouble(textN1.getText().toString());
        double2 = Double.parseDouble(textN2.getText().toString());

        switch (operand) {
            case "+":
                answer = double1 + double2;
                break;
            case ("-"):
                answer = double1 - double2;
                break;
            case "*":
                answer = double1 * double2;
                break;
            case "/":
                answer = double1 / double2;
                break;
            default:
                Log.d( tag: "BUTTON",  msg: "Not a valid operand");
        }
        Log.w( tag: operand + " BUTTON",  msg: "Selected with => " + double1 + operand + double2 + "=" + answer);
        textANS.setText(answer.toString());
    }
lculator  >  calculate
```

Rule 3 was having no God classes, so I'd count that as 1 change for having my music player be separate. And then since I also had to separate the calculator functions, I count this as separate as well for the 2 non-rule related changes (since it says 5% per change and a max of 40% so 8 total).

```
public void calculate(View view, String operand) throws ArithmeticException {  1 usage
    Log.d( tag: operand + " BUTTON",  msg: "User tapped the " + operand + " button.");
    double1 = Double.parseDouble(textN1.getText().toString());
    double2 = Double.parseDouble(textN2.getText().toString());

    switch (operand) {
        case "+":
            answer = double1 + double2;
            break;
        case ("-"):
            answer = double1 - double2;
            break;
        case "*":
            answer = double1 * double2;
            break;
        case "/":
            answer = double1 / double2;
            break;
        default:
            Log.d( tag: "BUTTON",  msg: "Not a valid operand");
    }
    Log.w( tag: operand + " BUTTON",  msg: "Selected with => " + double1 + operand + double2 + "=" + answer);
    textANS.setText(answer.toString());
}
```

Since I wasn't using any nested if statements originally, it makes more sense to use a switch statement here, so that's exactly what I did.

3. [10% per person who reviews YOUR code, max 20%] 2 students review your code. A comment does not require a fix....that's up to you.

   See other included documents

4. [20% per code review YOU do] You review at least 1 person's code, max 3.

   See other included documents

5. [20%]:
   1. Visit and read Release with confidence page: https://play.google.com/console/about/guides/releasewithconfidence/
   2. Visit and read Prepare your app for release page: https://developer.android.com/studio/publish/preparing

3. Comment on your Apps readiness (no need to actually make your app ready, just comment on what you would have to do) on any 4 [5% each] of the Checklist items mentioned.

When it comes to my app's readiness, there are a few things I would need to do to make it be okay, so here are the main four:

1 Remove/Change Copyrighted Material: In my app, I have an image from Seinfeld (I edited it, but still), a sound clip from Seinfeld, an image of a plush from Tsukihime (probably fine since it's just an image of a product, but it's still not my image), and a sound clip from Melty Blood. Obviously that would cause some legal problems for me, and is something that almost all app stores do not allow.

2 Get A Key: For an app to be ready on the PlayStore, it must also be signed with a Cryptographic key. From what I've been able to gleam from reading it, that's something that can be generated when an app is uploaded, and is needed, so an app can actually be installed or updated on the user's end. It's also stored by Google, and is something that needs to be regenerated if lost or compromised. Signing is pretty important itself when it comes to exes on computers, and it's quite the process to get an exe signed, so I thought it would be a good thing to go over here.

3 EULA: I would also have to create an End User Licence agreement. For something like this, there wouldn't be any permissions (like "uses personal information" for example), but still having that legal stuff is very important. Say I don't do my due diligence, and my app crashes and maybe corrupts some data on the user's phone. That's something that should never happen, but in the odd chance it does, that's something I don't want to be blamed for, and the EULA can help with that.

4 More Compatibility: My app is only compatible with 6 different screen sizes, and despite how it felt, that's not a lot all things considered. If I wanted to gross dollar 1, so to speak, doubling that is only 6 devices total (due to landscape and portrait), so that might just be the bare minimum.