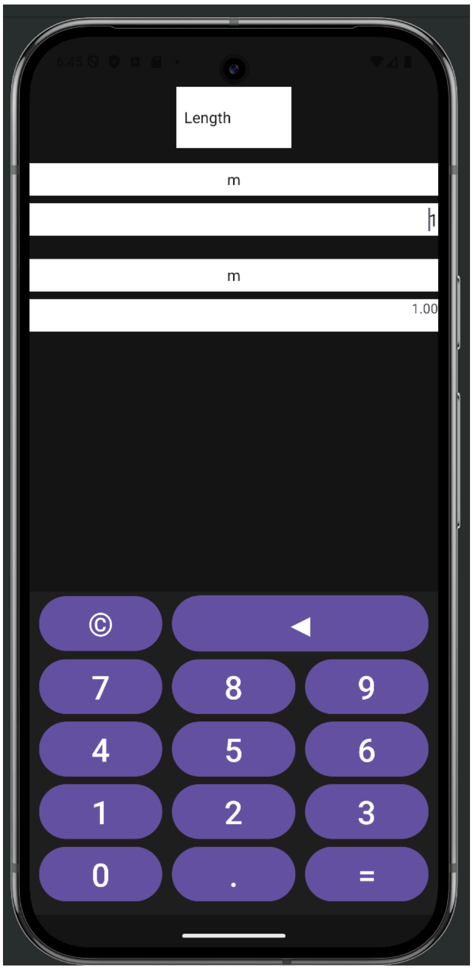
Documentation

Link to Github: https://github.com/Drogshell/SIT305.git
Link to YouTube Vid: https://youtu.be/S38DmG3yCoU

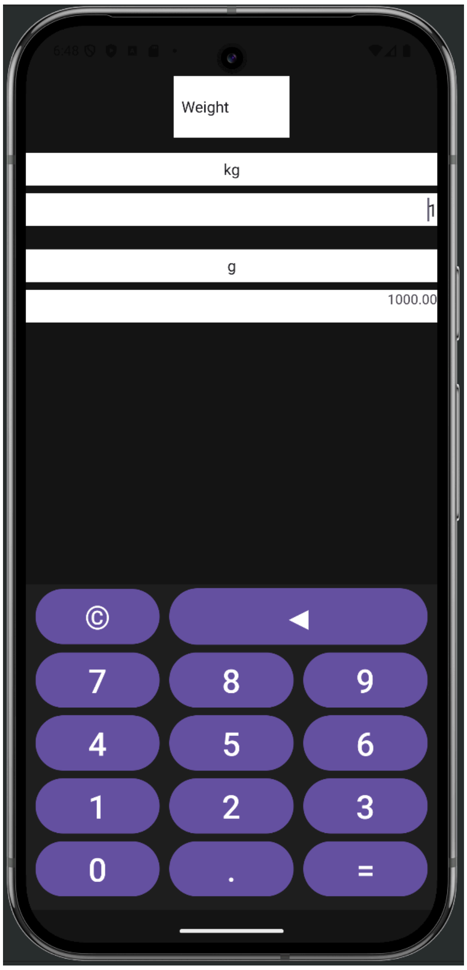


Length

m

m

1.00

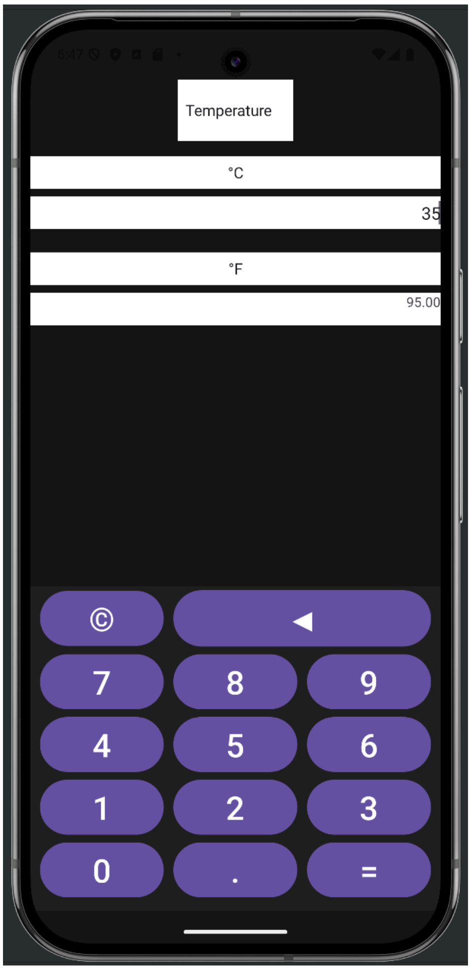


Weight

kg

g

1000.00



Temperature

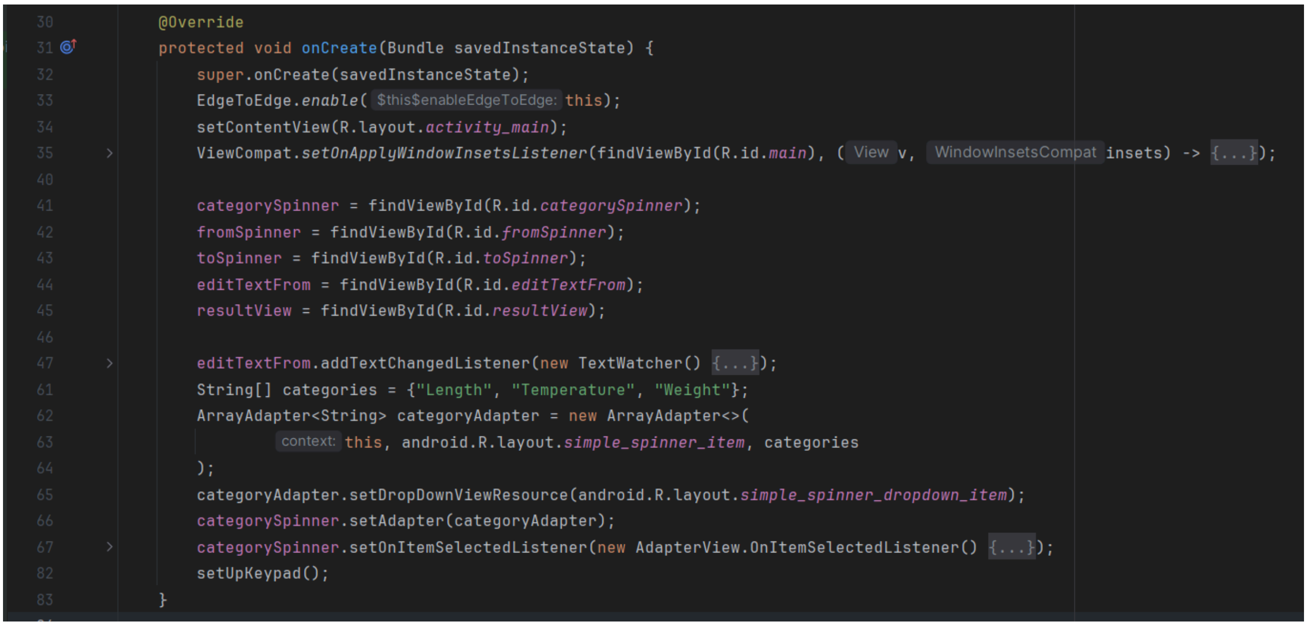
°C

35

°F

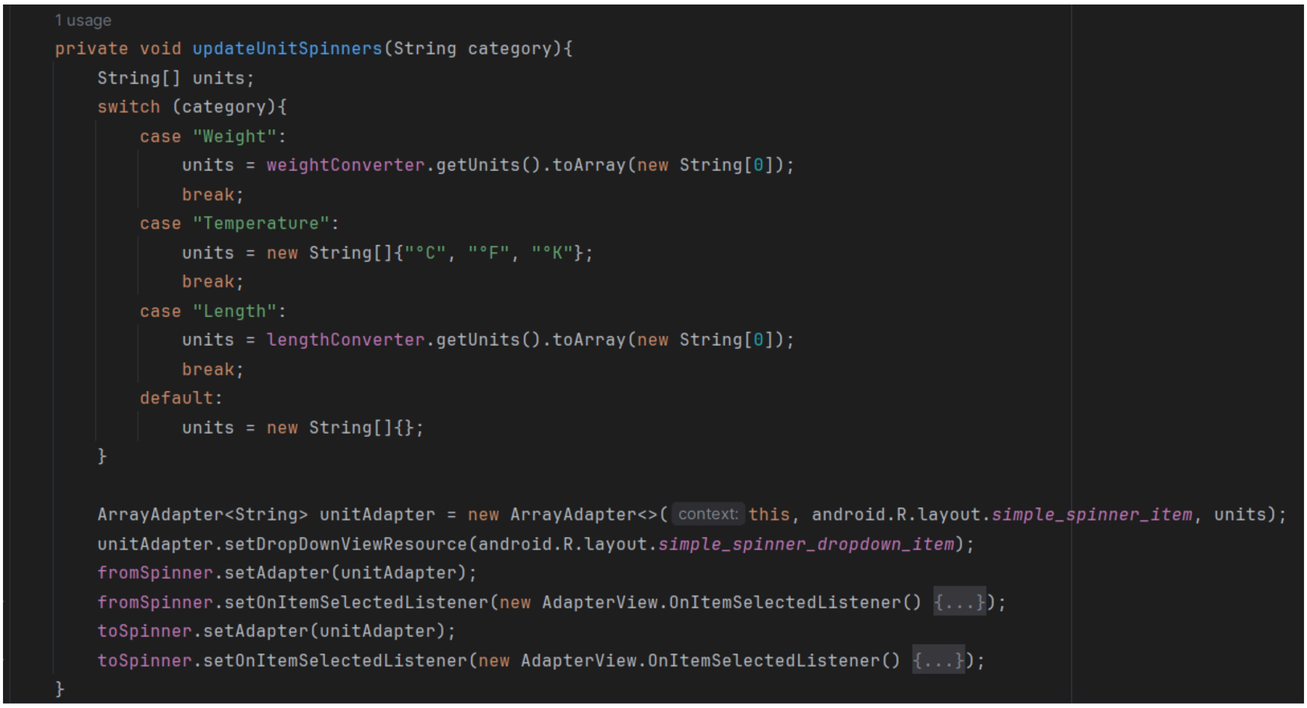
95.00

Here is a walk-through of how the code works:



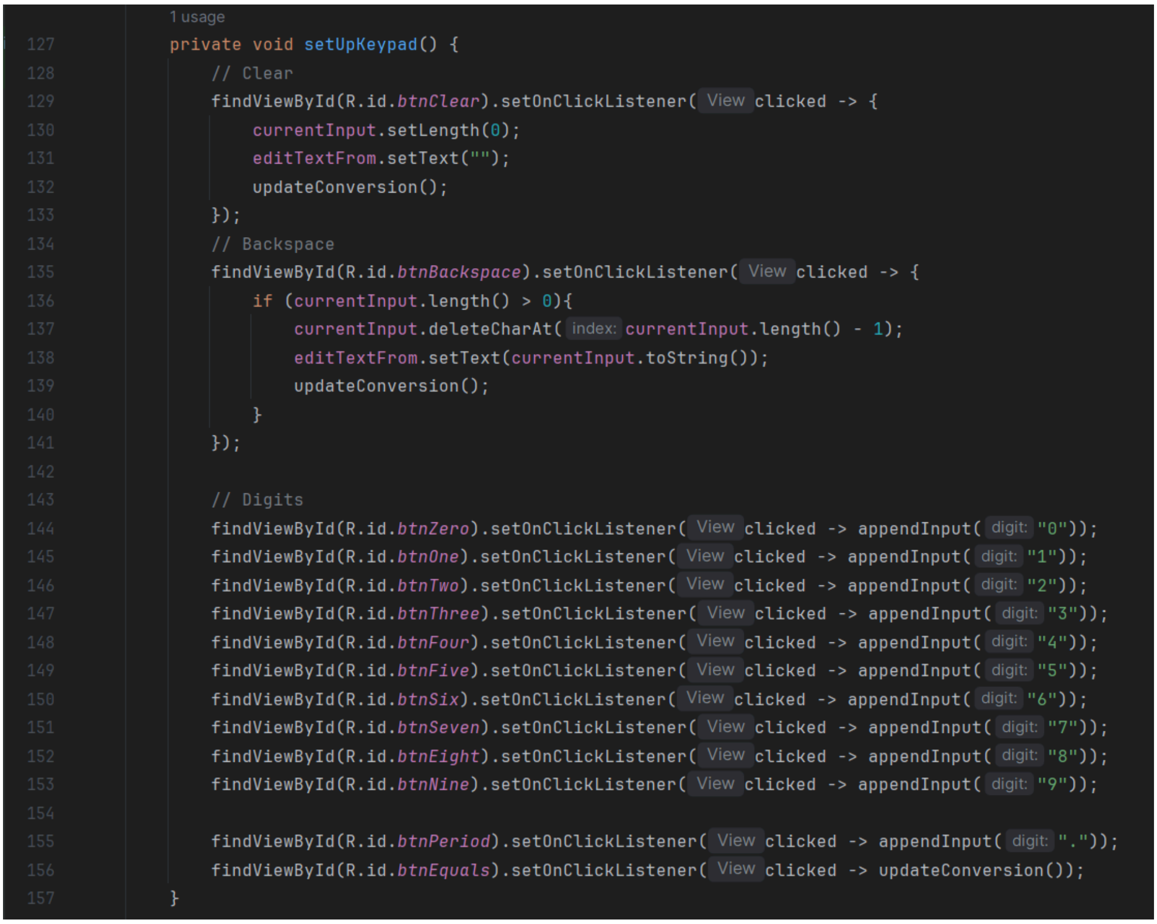
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    EdgeToEdge.enable( $this$enableEdgeToEdge: this);
    setContentView(R.layout.activity_main);
    ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), ( View v,  WindowInsetsCompat insets) -> {...});

    categorySpinner = findViewById(R.id.categorySpinner);
    fromSpinner = findViewById(R.id.fromSpinner);
    toSpinner = findViewById(R.id.toSpinner);
    editTextFrom = findViewById(R.id.editTextFrom);
    resultView = findViewById(R.id.resultView);

    editTextFrom.addTextChangedListener(new TextWatcher() {...});
    String[] categories = {"Length", "Temperature", "Weight"};
    ArrayAdapter<String> categoryAdapter = new ArrayAdapter<>(
             context: this, android.R.layout.simple_spinner_item, categories
    );
    categoryAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
    categorySpinner.setAdapter(categoryAdapter);
    categorySpinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {...});
    setUpKeypad();
}
```

OnCreate first instantiates all the required variables I'll be using across the class. Adds a text changed listener to the editTextFrom variable so that the text updates as the user adds or subtracts numbers. Then, sets up the keypad with the numbers.



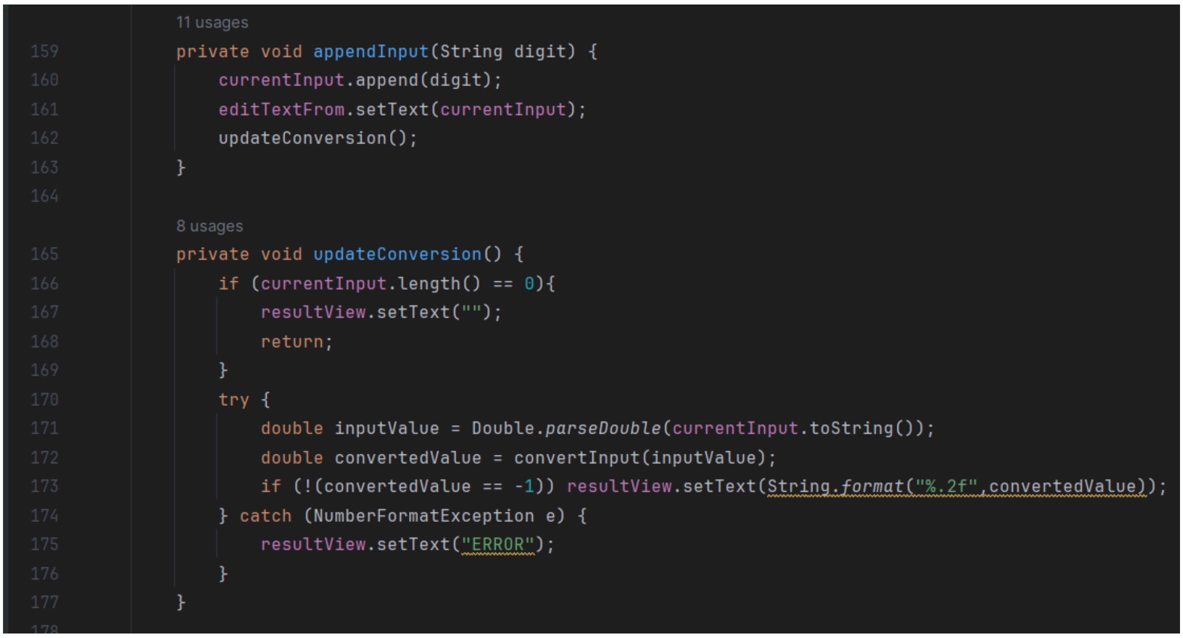
```
1 usage
private void updateUnitSpinners(String category){
    String[] units;
    switch (category){
        case "Weight":
            units = weightConverter.getUnits().toArray(new String[0]);
            break;
        case "Temperature":
            units = new String[]{"°C", "°F", "°K"};
            break;
        case "Length":
            units = lengthConverter.getUnits().toArray(new String[0]);
            break;
        default:
            units = new String[]{};
    }

    ArrayAdapter<String> unitAdapter = new ArrayAdapter<>( context: this, android.R.layout.simple_spinner_item, units);
    unitAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
    fromSpinner.setAdapter(unitAdapter);
    fromSpinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {...});
    toSpinner.setAdapter(unitAdapter);
    toSpinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {...});
}
```

The unit spinners are the dropdowns that the user can click on to select what units they want to convert from and to. They are populated based on the category that is selected. The strings that are displayed are based on the conversion factor classes. More on that later.



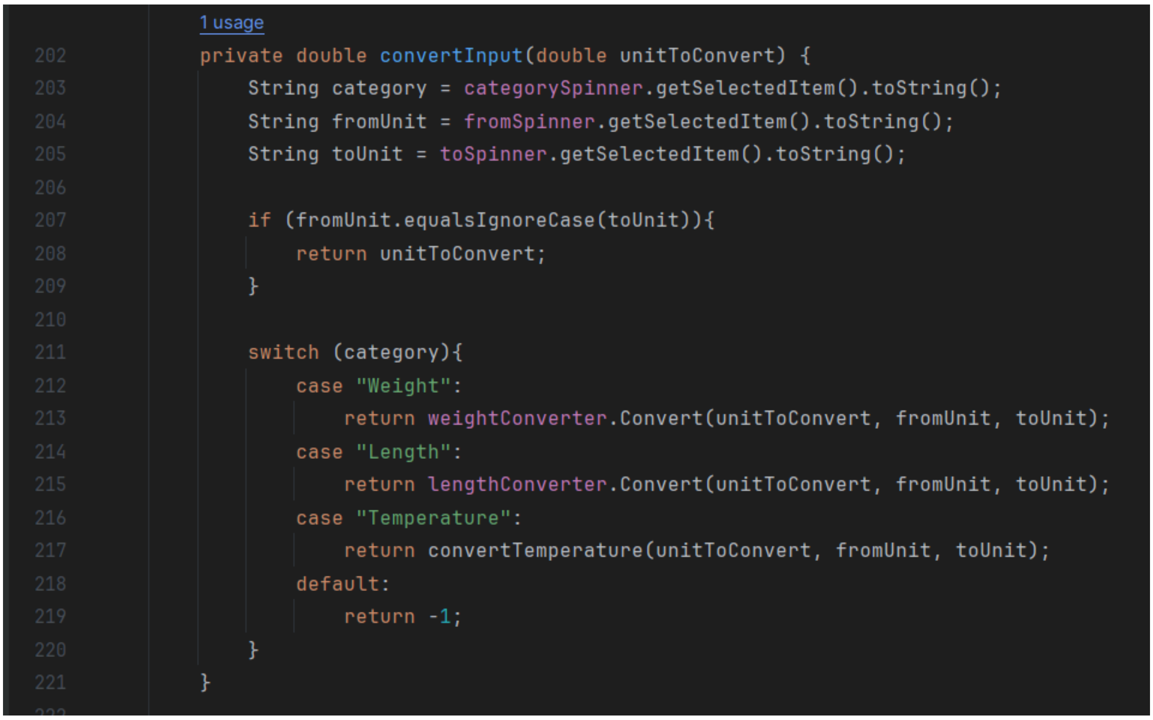
```
1 usage
private void setUpKeypad() {
    // Clear
    findViewById(R.id.btnClear).setOnClickListener( View clicked -> {
        currentInput.setLength(0);
        editTextFrom.setText("");
        updateConversion();
    });
    // Backspace
    findViewById(R.id.btnBackspace).setOnClickListener( View clicked -> {
        if (currentInput.length() > 0){
            currentInput.deleteCharAt( index: currentInput.length() - 1);
            editTextFrom.setText(currentInput.toString());
            updateConversion();
        }
    });

    // Digits
    findViewById(R.id.btnZero).setOnClickListener( View clicked -> appendInput( digit: "0"));
    findViewById(R.id.btnOne).setOnClickListener( View clicked -> appendInput( digit: "1"));
    findViewById(R.id.btnTwo).setOnClickListener( View clicked -> appendInput( digit: "2"));
    findViewById(R.id.btnThree).setOnClickListener( View clicked -> appendInput( digit: "3"));
    findViewById(R.id.btnFour).setOnClickListener( View clicked -> appendInput( digit: "4"));
    findViewById(R.id.btnFive).setOnClickListener( View clicked -> appendInput( digit: "5"));
    findViewById(R.id.btnSix).setOnClickListener( View clicked -> appendInput( digit: "6"));
    findViewById(R.id.btnSeven).setOnClickListener( View clicked -> appendInput( digit: "7"));
    findViewById(R.id.btnEight).setOnClickListener( View clicked -> appendInput( digit: "8"));
    findViewById(R.id.btnNine).setOnClickListener( View clicked -> appendInput( digit: "9"));

    findViewById(R.id.btnPeriod).setOnClickListener( View clicked -> appendInput( digit: "."));
    findViewById(R.id.btnEquals).setOnClickListener( View clicked -> updateConversion());
}
```

setUpKeyPad actually hooks the UI to the backend code and determines what happens when a user taps on a number.



```
11 usages
private void appendInput(String digit) {
    currentInput.append(digit);
    editTextFrom.setText(currentInput);
    updateConversion();
}

8 usages
private void updateConversion() {
    if (currentInput.length() == 0){
        resultView.setText("");
        return;
    }
    try {
        double inputValue = Double.parseDouble(currentInput.toString());
        double convertedValue = convertInput(inputValue);
        if (!(convertedValue == -1)) resultView.setText(String.format("%.2f",convertedValue));
    } catch (NumberFormatException e) {
        resultView.setText("ERROR");
    }
}
```

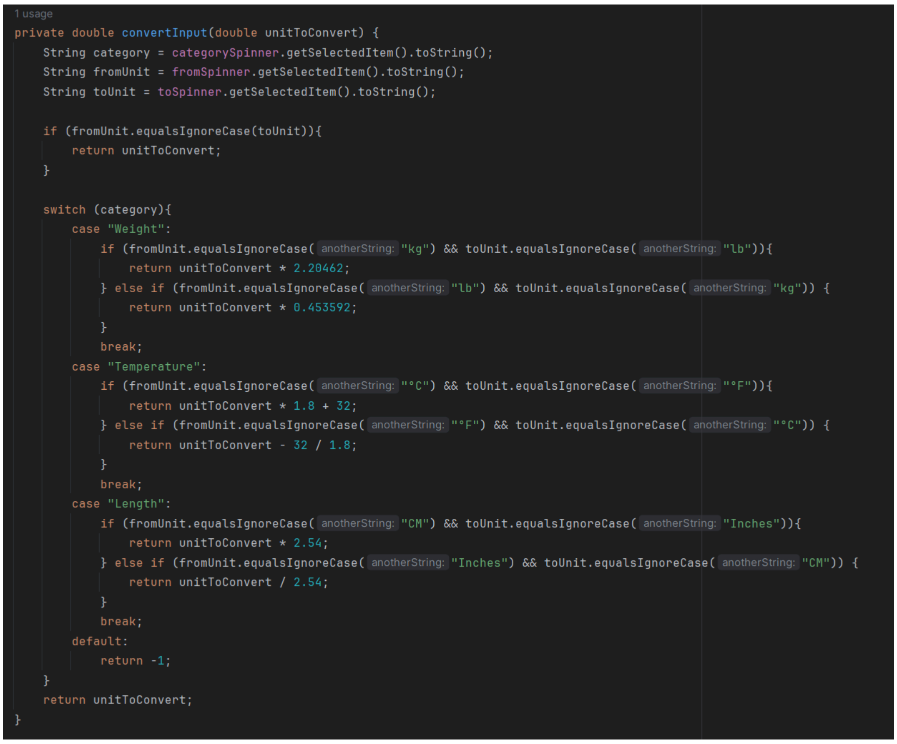
Append Input just adds the string representation of the numbers in the keypad and then immediately calls updateConversion so that users can see it in real time.



```
1 usage
private double convertInput(double unitToConvert) {
    String category = categorySpinner.getSelectedItem().toString();
    String fromUnit = fromSpinner.getSelectedItem().toString();
    String toUnit = toSpinner.getSelectedItem().toString();

    if (fromUnit.equalsIgnoreCase(toUnit)){
        return unitToConvert;
    }

    switch (category){
        case "Weight":
            return weightConverter.Convert(unitToConvert, fromUnit, toUnit);
        case "Length":
            return lengthConverter.Convert(unitToConvert, fromUnit, toUnit);
        case "Temperature":
            return convertTemperature(unitToConvert, fromUnit, toUnit);
        default:
            return -1;
    }
}
```

convertInput calls the respective classes that then handle the conversion. With the exception of temperature, since temperatures have very specific formulas for conversion.

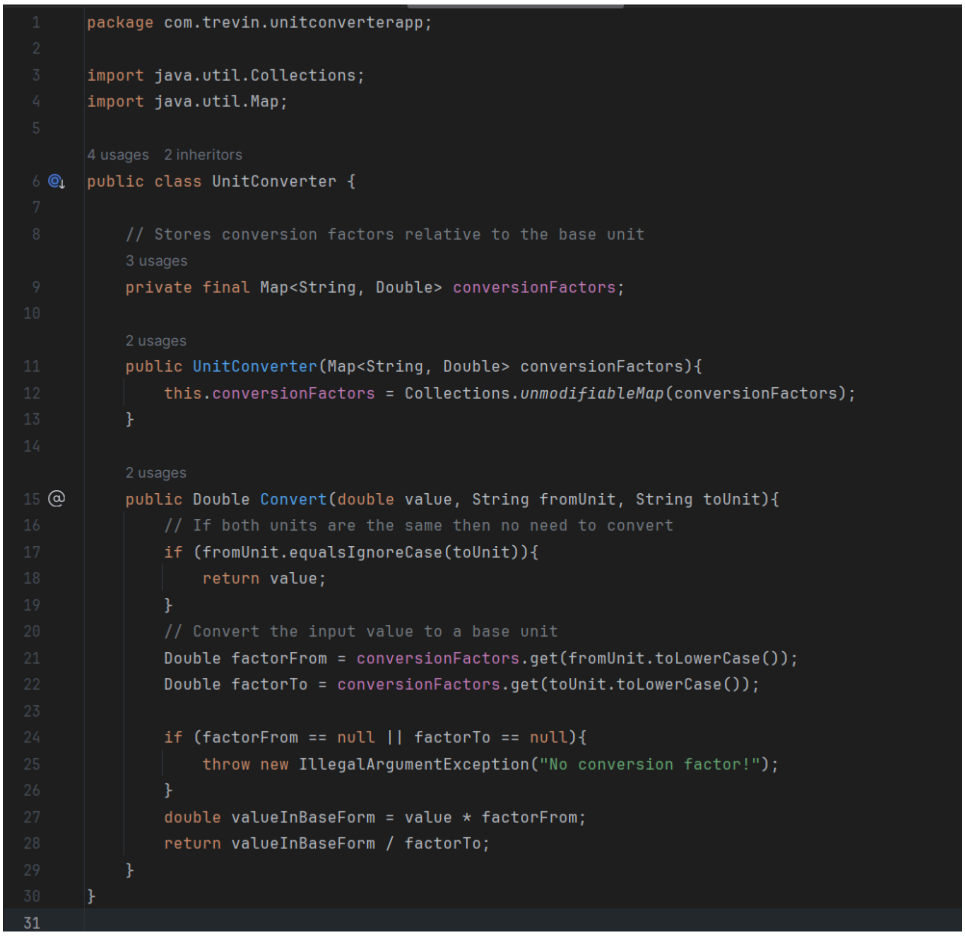
When writing the conversion code, the original method I used, as shown below, was to use switch case statements.



```
1 usage
private double convertInput(double unitToConvert) {
    String category = categorySpinner.getSelectedItem().toString();
    String fromUnit = fromSpinner.getSelectedItem().toString();
    String toUnit = toSpinner.getSelectedItem().toString();

    if (fromUnit.equalsIgnoreCase(toUnit)){
        return unitToConvert;
    }

    switch (category){
        case "Weight":
            if (fromUnit.equalsIgnoreCase( anotherString: "kg") && toUnit.equalsIgnoreCase( anotherString: "lb")){
                return unitToConvert * 2.20462;
            } else if (fromUnit.equalsIgnoreCase( anotherString: "lb") && toUnit.equalsIgnoreCase( anotherString: "kg")) {
                return unitToConvert * 0.453592;
            }
            break;
        case "Temperature":
            if (fromUnit.equalsIgnoreCase( anotherString: "°C") && toUnit.equalsIgnoreCase( anotherString: "°F")){
                return unitToConvert * 1.8 + 32;
            } else if (fromUnit.equalsIgnoreCase( anotherString: "°F") && toUnit.equalsIgnoreCase( anotherString: "°C")) {
                return unitToConvert - 32 / 1.8;
            }
            break;
        case "Length":
            if (fromUnit.equalsIgnoreCase( anotherString: "CM") && toUnit.equalsIgnoreCase( anotherString: "Inches")){
                return unitToConvert * 2.54;
            } else if (fromUnit.equalsIgnoreCase( anotherString: "Inches") && toUnit.equalsIgnoreCase( anotherString: "CM")) {
                return unitToConvert / 2.54;
            }
            break;
        default:
            return -1;
    }
    return unitToConvert;
}
```

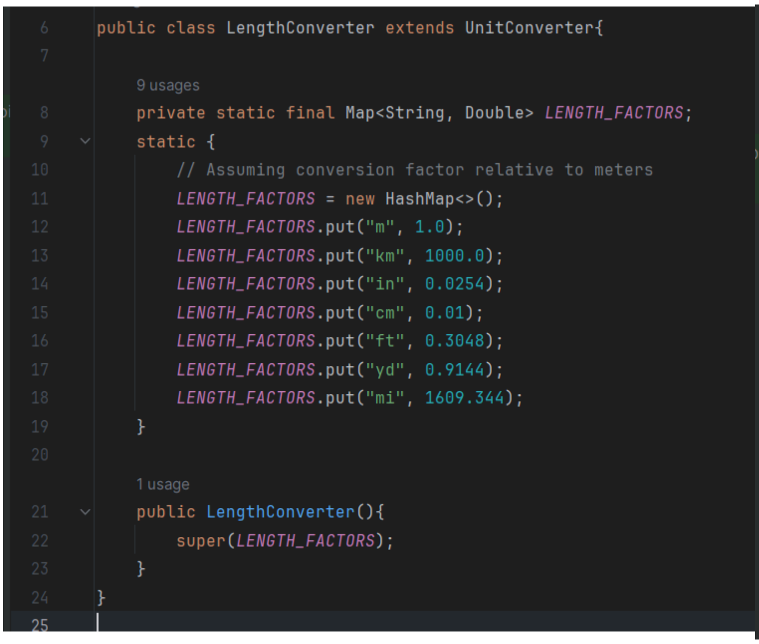
This worked, but it was absolutely horrible to write and is not easily extensible. I would need dozens of if else statements. If I ever needed to change or add more units to convert, I would have to make changes in multiple places.

When thinking about using SOLID architecture to write clean code, I decided to apply the open/close principle to encapsulate the conversion.
Converting weight and length can be done by using a relative factor.
So, I created a UnitConverter class:

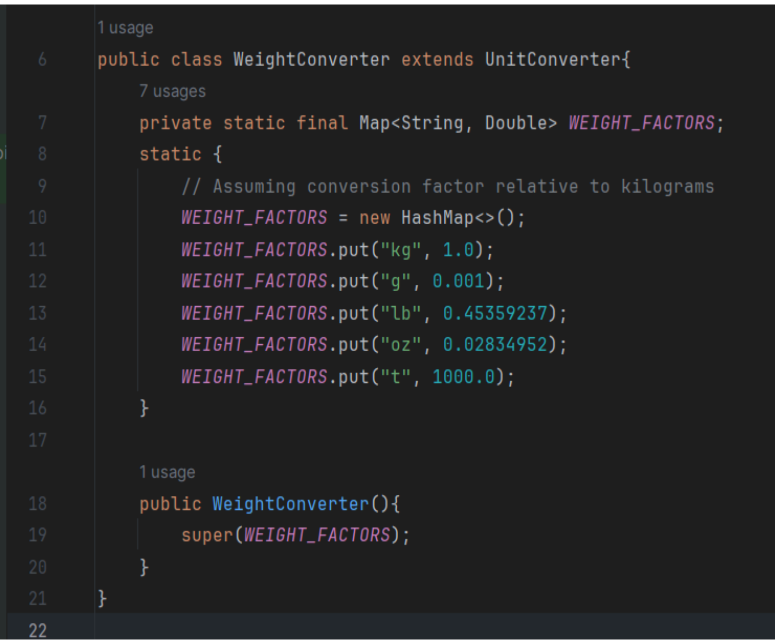


```java
package com.trevin.unitconverterapp;

import java.util.Collections;
import java.util.Map;

public class UnitConverter {

    // Stores conversion factors relative to the base unit
    private final Map<String, Double> conversionFactors;

    public UnitConverter(Map<String, Double> conversionFactors){
        this.conversionFactors = Collections.unmodifiableMap(conversionFactors);
    }

    public Double Convert(double value, String fromUnit, String toUnit){
        // If both units are the same then no need to convert
        if (fromUnit.equalsIgnoreCase(toUnit)){
            return value;
        }
        // Convert the input value to a base unit
        Double factorFrom = conversionFactors.get(fromUnit.toLowerCase());
        Double factorTo = conversionFactors.get(toUnit.toLowerCase());

        if (factorFrom == null || factorTo == null){
            throw new IllegalArgumentException("No conversion factor!");
        }
        double valueInBaseForm = value * factorFrom;
        return valueInBaseForm / factorTo;
    }
}
```

This class handles the conversion by using a Map where each Key is mapped to a conversion factor. Then, all I have to do is create classes that inherit from the unitConverter class. These individual classes handle all the data related to their conversion.



```java
public class LengthConverter extends UnitConverter{

    private static final Map<String, Double> LENGTH_FACTORS;
    static {
        // Assuming conversion factor relative to meters
        LENGTH_FACTORS = new HashMap<>();
        LENGTH_FACTORS.put("m", 1.0);
        LENGTH_FACTORS.put("km", 1000.0);
        LENGTH_FACTORS.put("in", 0.0254);
        LENGTH_FACTORS.put("cm", 0.01);
        LENGTH_FACTORS.put("ft", 0.3048);
        LENGTH_FACTORS.put("yd", 0.9144);
        LENGTH_FACTORS.put("mi", 1609.344);
    }

    public LengthConverter(){
        super(LENGTH_FACTORS);
    }
}
```



```java
public class WeightConverter extends UnitConverter{
    private static final Map<String, Double> WEIGHT_FACTORS;
    static {
        // Assuming conversion factor relative to kilograms
        WEIGHT_FACTORS = new HashMap<>();
        WEIGHT_FACTORS.put("kg", 1.0);
        WEIGHT_FACTORS.put("g", 0.001);
        WEIGHT_FACTORS.put("lb", 0.45359237);
        WEIGHT_FACTORS.put("oz", 0.02834952);
        WEIGHT_FACTORS.put("t", 1000.0);
    }

    public WeightConverter(){
        super(WEIGHT_FACTORS);
    }
}
```

This not only means that the data is far more encapsulated than before but adding more conversions is also much easier now, requiring changes in only one place: the class where the map is located.