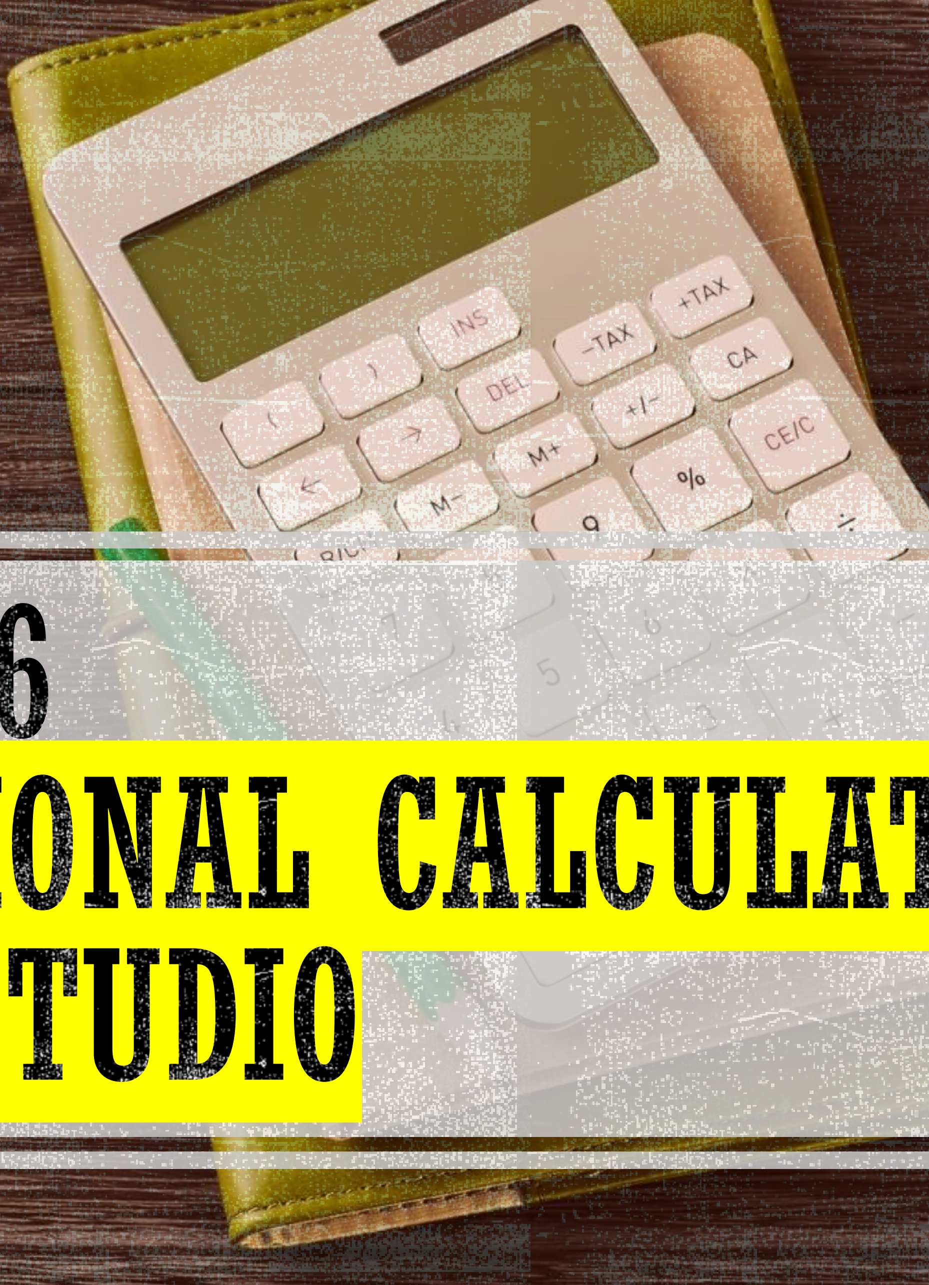


LECTURE # 06
FULLY FUNCTIONAL CALCULATOR IN ANDROID STUDIO

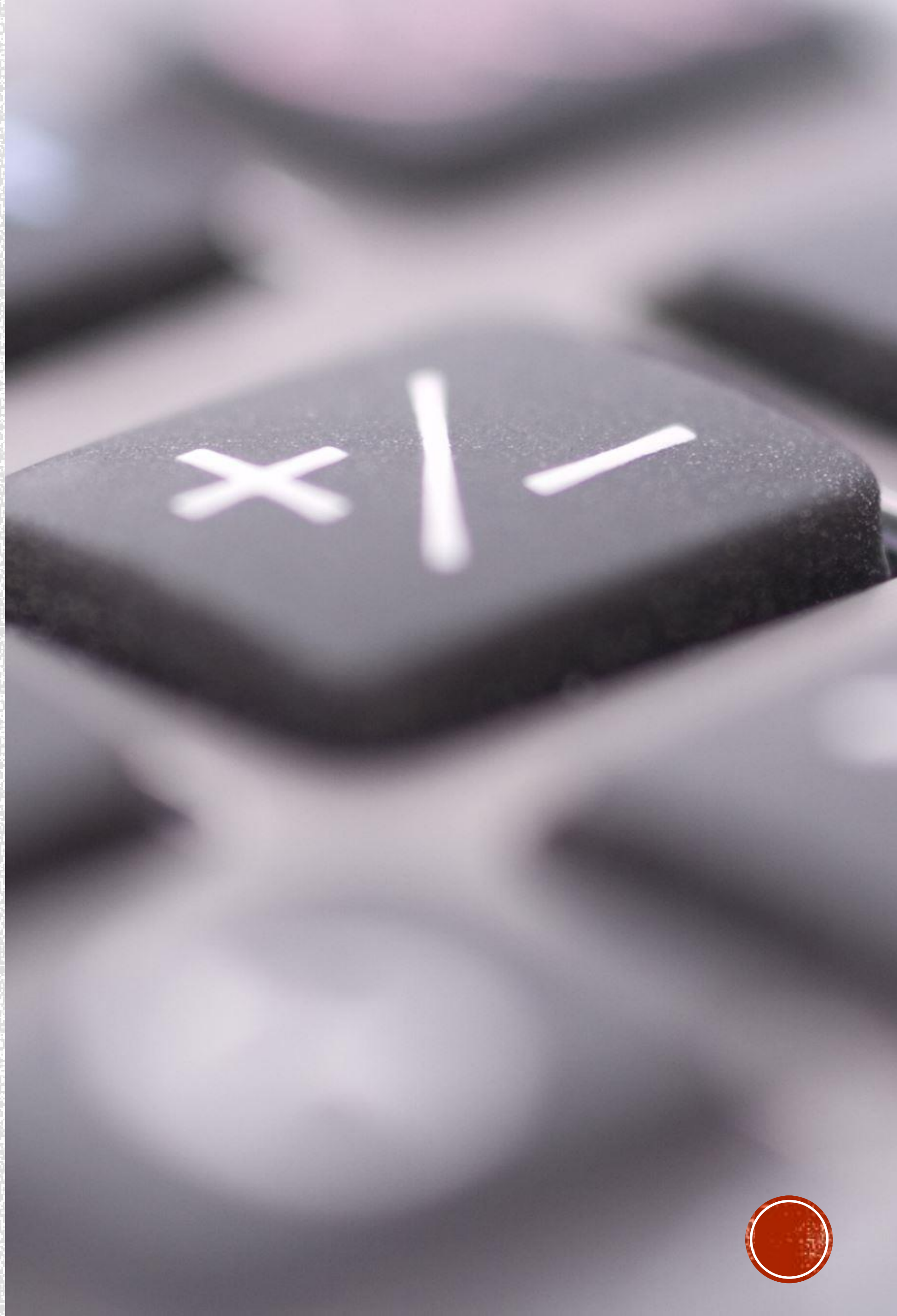
OUTCOMES:

- After completing this lab student will be able to understand:

1. **Overview of the Calculator Application**
2. **Exploring the Code Structure**
   a. Package and Imports
   b. Activity Class and Layout
3. **User Interface Components**
   a. TextView and Buttons
   b. Initializing UI Components
4. **Button Click Listeners**
   a. Numeric Buttons
   b. Operator Buttons
   c. Special Function Buttons
5. **Mathematical Operations**
   a. Evaluating Expressions
   b. Handling Parentheses
   c. Unary Operations and Functions
6. **Additional Features**
   a. Clearing and Deleting Input
   b. Factorial and Square Functions

INTRODUCTION

- A calculator is a fundamental tool that allows users to perform mathematical operations conveniently. In this presentation, we will discuss how to build a fully functional calculator application for Android. We will explore the code provided and explain its structure, functionality, and usage.



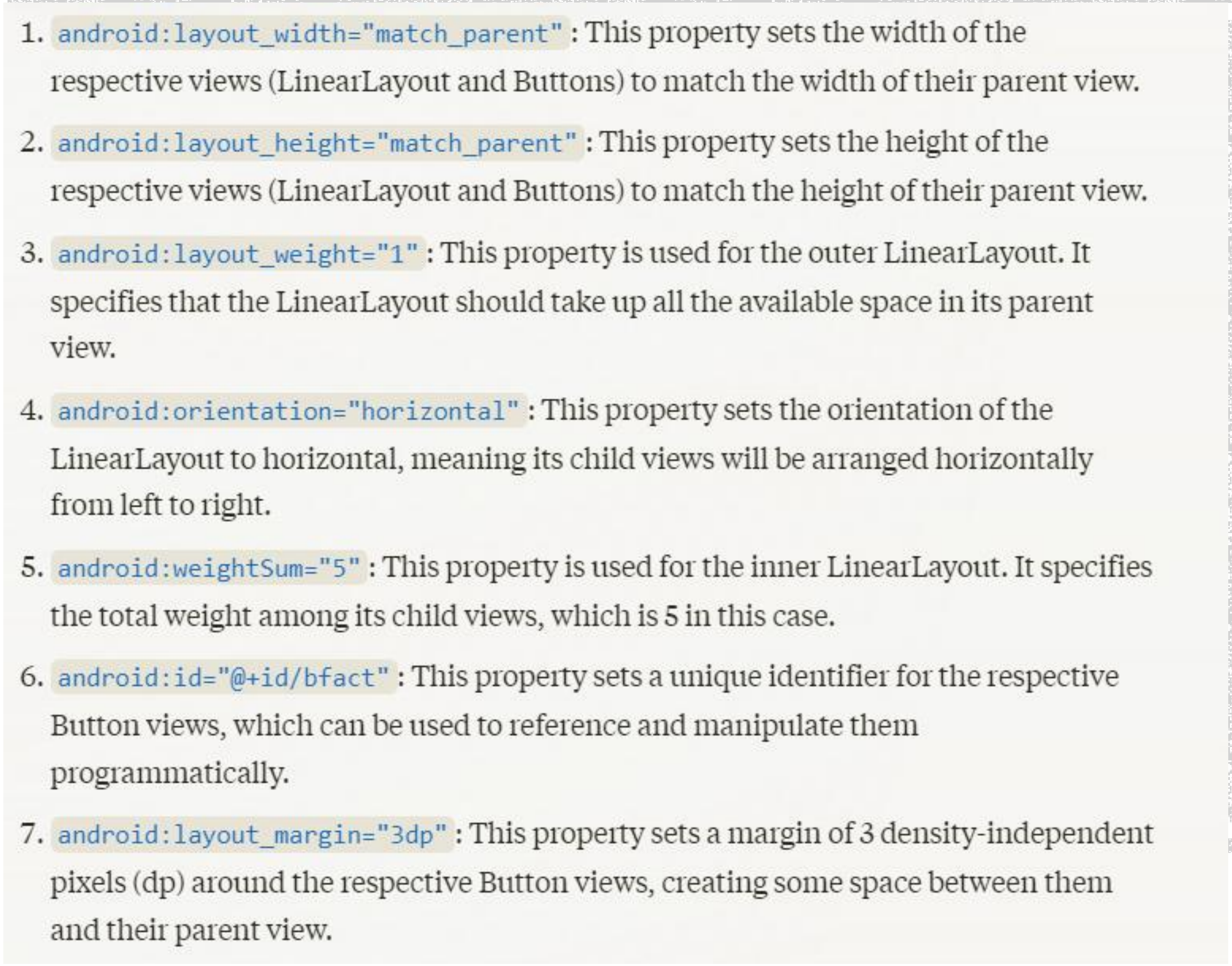
ADVANTAGES:

1. **Convenience:** A calculator app provides users with a convenient tool for performing mathematical calculations directly on their mobile devices. It eliminates the need to carry a physical calculator and allows users to access it anytime, anywhere.
2. **Accessibility:** Calculators are essential in various fields, including education, finance, engineering, and science. By building a calculator app, you make these tools easily accessible to a wide range of users. Students, professionals, and individuals in different industries can benefit from having a calculator readily available on their smartphones.
3. **Enhanced Functionality:** Unlike traditional calculators, a calculator app can offer advanced features and functions. Developers have the flexibility to integrate additional mathematical operations, scientific functions, unit conversions, and other useful tools. This increases the app's utility and makes it a powerful tool for complex calculations.
4. **Customization:** Calculator apps can be customized to meet specific user requirements. They can be tailored to accommodate different themes, color schemes, button layouts, and user preferences. This level of customization enhances the user experience and allows individuals to personalize the app according to their preferences.

REAL LIFE APPLICATION:

1. **Education:** Calculator apps are widely used by students at various academic levels. They assist in solving mathematical equations, performing scientific calculations, and simplifying complex problems. Additionally, specialized calculator apps can be designed for specific subjects such as physics, engineering, or finance, providing students with subject-specific tools.
2. **Finance and Accounting:** Calculator apps are valuable for financial professionals, accountants, and individuals handling financial matters. They enable calculations related to budgeting, loan calculations, interest rates, investment returns, and currency conversions. These apps can also include functions for tax calculations, mortgage calculations, and other financial planning tasks.
3. **Engineering and Science:** Scientists, engineers, and researchers often require calculators for complex calculations and data analysis. Calculator apps can be equipped with advanced mathematical functions, statistical operations, unit conversions, and constants specific to these fields. This allows professionals to perform calculations efficiently and accurately.
4. **Business and Retail:** In business and retail settings, calculator apps can be used for inventory management, profit calculations, pricing calculations, and sales analysis. These apps simplify financial calculations and provide quick and accurate results, enhancing productivity and decision-making processes.

LAYOUT PROPERTIES:

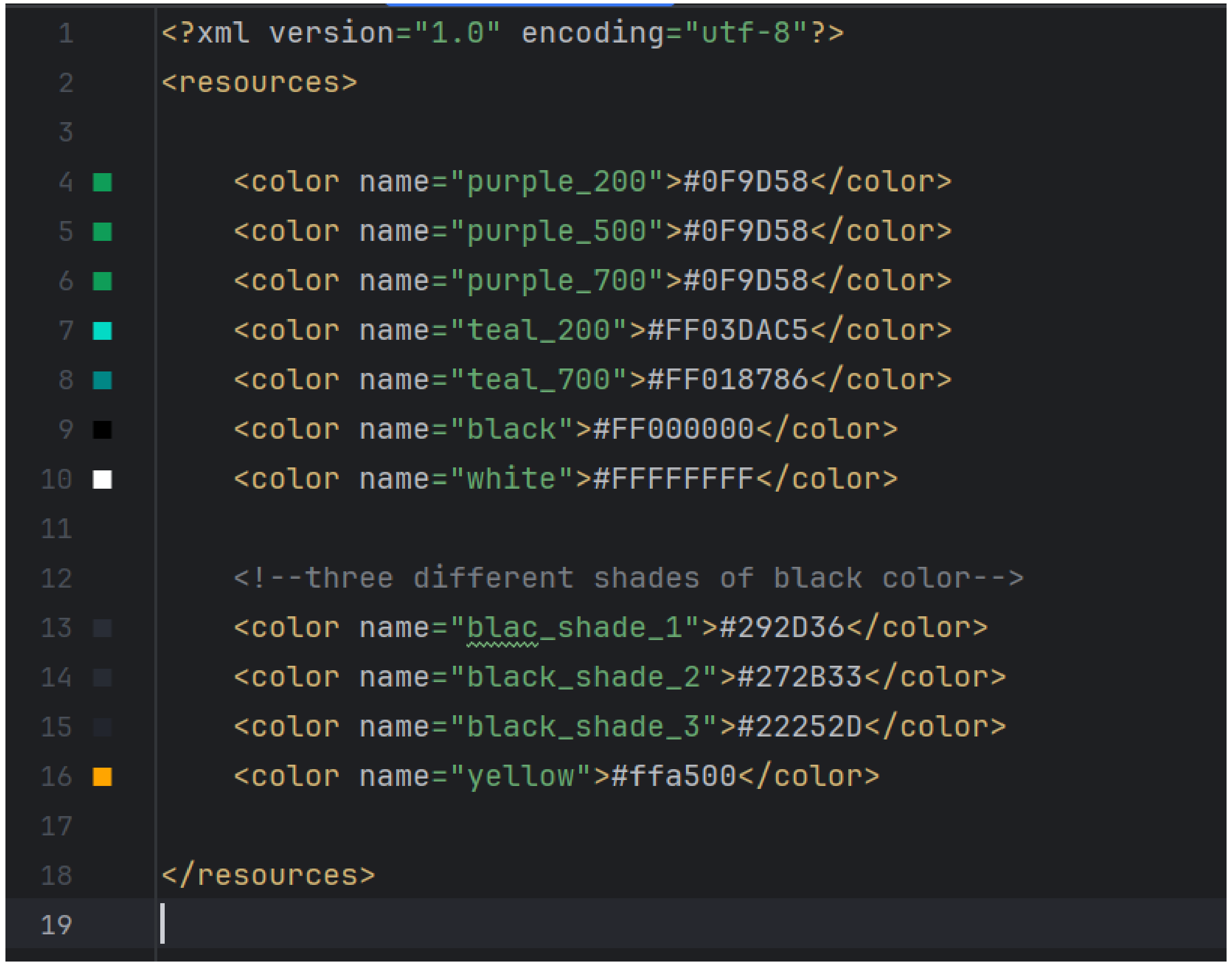


1. `android:layout_width="match_parent"` : This property sets the width of the respective views (LinearLayout and Buttons) to match the width of their parent view.
2. `android:layout_height="match_parent"` : This property sets the height of the respective views (LinearLayout and Buttons) to match the height of their parent view.
3. `android:layout_weight="1"` : This property is used for the outer LinearLayout. It specifies that the LinearLayout should take up all the available space in its parent view.
4. `android:orientation="horizontal"` : This property sets the orientation of the LinearLayout to horizontal, meaning its child views will be arranged horizontally from left to right.
5. `android:weightSum="5"` : This property is used for the inner LinearLayout. It specifies the total weight among its child views, which is 5 in this case.
6. `android:id="@+id/bfact"` : This property sets a unique identifier for the respective Button views, which can be used to reference and manipulate them programmatically.
7. `android:layout_margin="3dp"` : This property sets a margin of 3 density-independent pixels (dp) around the respective Button views, creating some space between them and their parent view.

8. `android:layout_weight="1"` : This property is used for the Button views. It specifies that each Button should take up an equal portion (1/5) of the available space in its parent LinearLayout (since the total weightSum is 5).
9. `android:backgroundTint="@color/black_shade_2"` : This property sets the background tint color of the respective Button views to a color resource named "black_shade_2".
10. `android:padding="6dp"` : This property sets a padding of 6 density-independent pixels (dp) around the text within the respective Button views.
11. `android:text="x!"` : This property sets the text displayed on the respective Button view.
12. `android:textColor="#ffa500"` : This property sets the text color of the respective Button views to an orange shade (#ffa500).
13. `android:textSize="15sp"` : This property sets the text size of the respective Button views to 15 scaled pixels (sp), which scales based on the user's font size preference.
14. `android:textAllCaps="false"` : This property is used for the "1/x" Button view. It specifies that the text should not be displayed in all capital letters.

CONTINUE:

LAYOUT/RES/COLORS FILE IN ANDROID STUDIO:



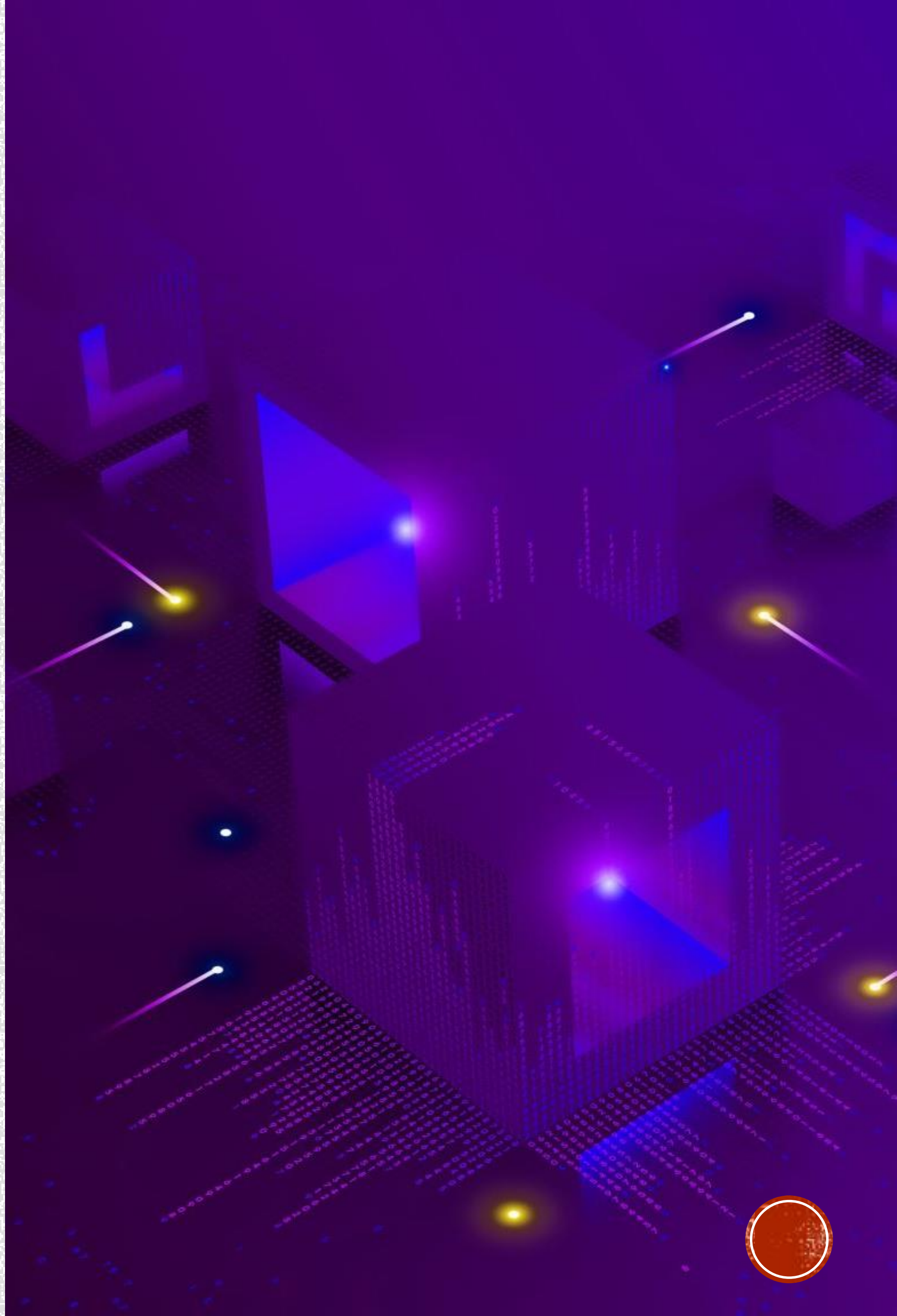
```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <color name="purple_200">#0F9D58</color>
    <color name="purple_500">#0F9D58</color>
    <color name="purple_700">#0F9D58</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>

    <!--three different shades of black color-->
    <color name="blac_shade_1">#292D36</color>
    <color name="black_shade_2">#272B33</color>
    <color name="black_shade_3">#22252D</color>
    <color name="yellow">#ffa500</color>

</resources>
```

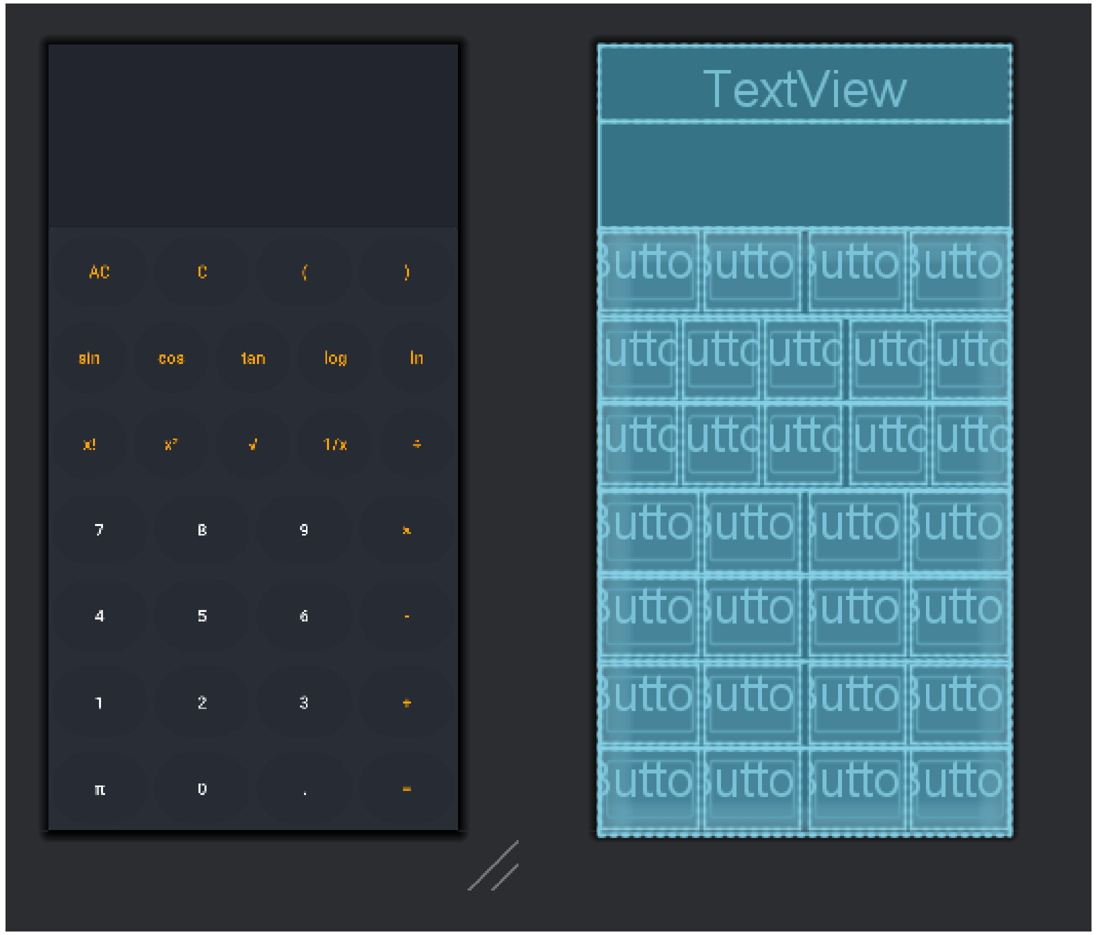
CARD VIEW IN ANDROID STUDIO:

- In Android Studio, the Card View widget is a UI component used to display content in a card-like format. It provides a way to present information in a visually appealing manner with rounded corners and elevation (shadow) effects, similar to physical cards.

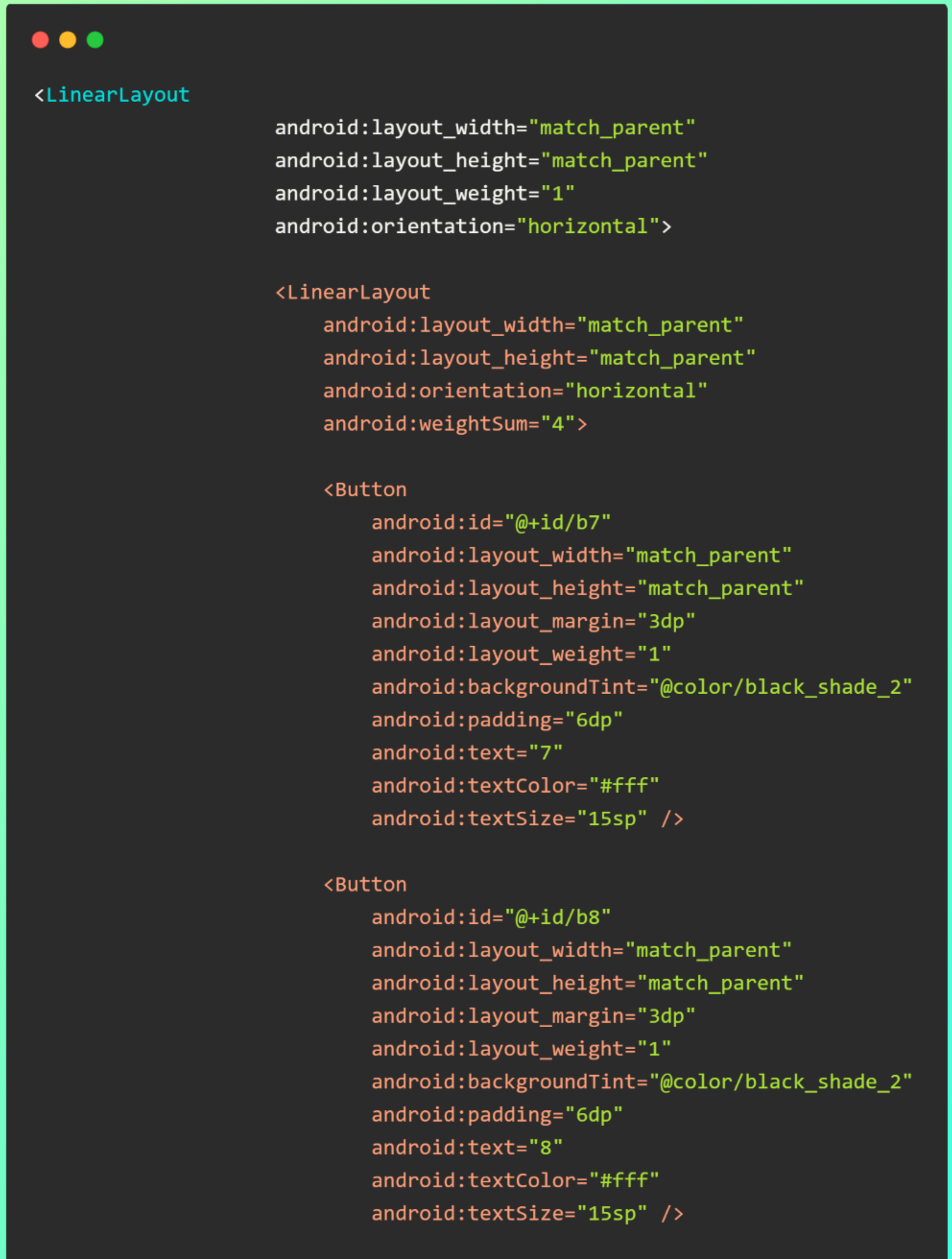
The purpose of using a Card View in Android Studio is to:

1. **Enhance Visual Presentation:** Card View adds a modern and visually appealing design to your app's user interface, making it more attractive and engaging for users.
2. **Group Related Content:** You can use Card View to group related content together, such as text, images, buttons, etc., making it easier for users to understand and interact with the information.
3. **Provide Consistency:** By using Card View throughout your app, you can maintain consistency in design and layout, creating a cohesive user experience across different screens and sections.
4. **Add Depth and Hierarchy:** The elevation effect of Card View helps to create a sense of depth and hierarchy in your app's UI, making it clear which content is more prominent or important.

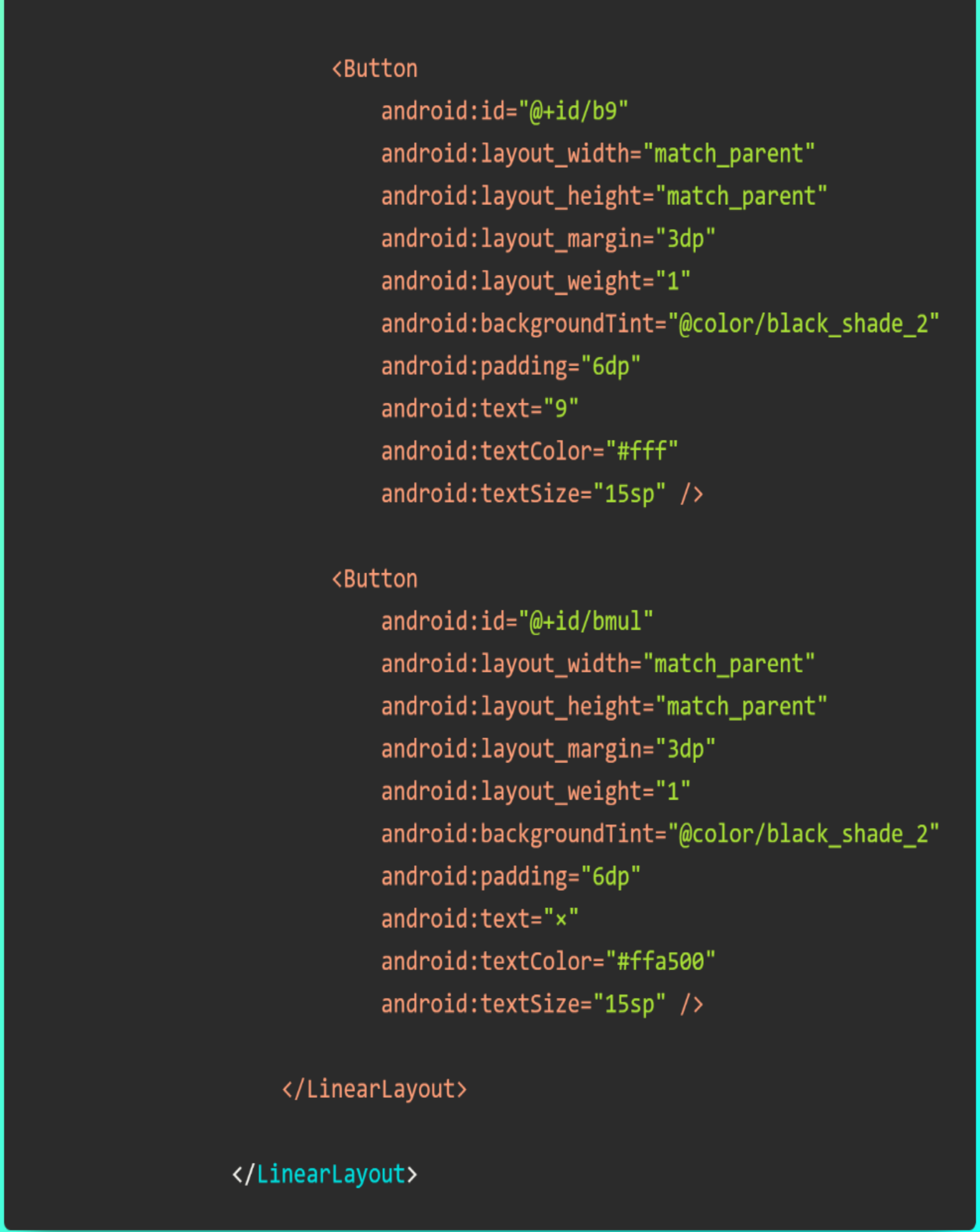




ACTIVITY LAYOUT.XML:

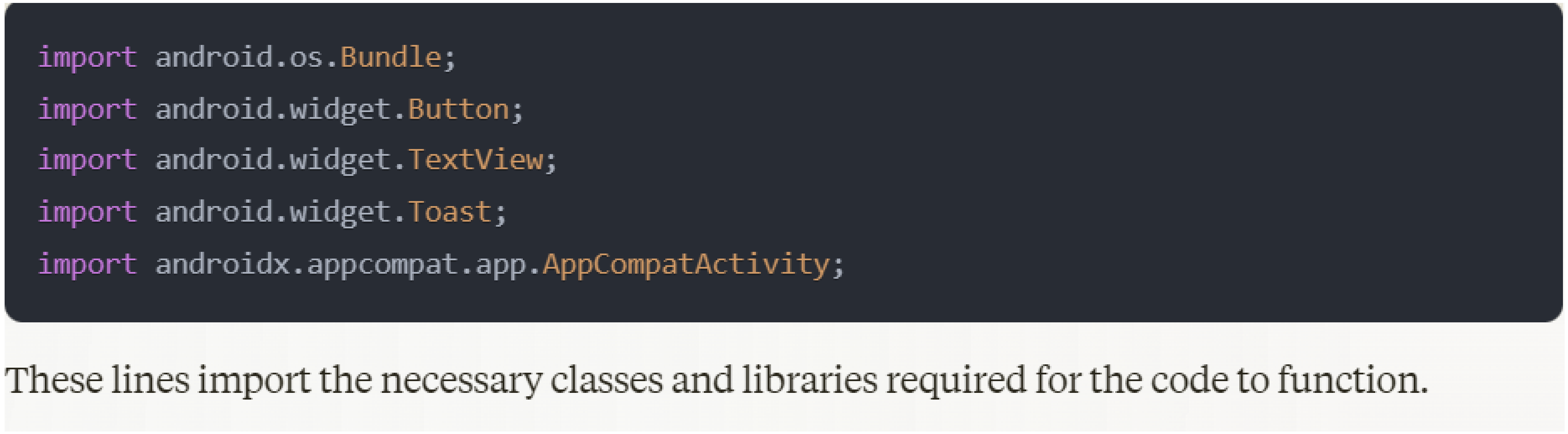


```
<LinearLayout
                    android:layout_width="match_parent"
                    android:layout_height="match_parent"
                    android:layout_weight="1"
                    android:orientation="horizontal">

                    <LinearLayout
                        android:layout_width="match_parent"
                        android:layout_height="match_parent"
                        android:orientation="horizontal"
                        android:weightSum="4">

                        <Button
                            android:id="@+id/b7"
                            android:layout_width="match_parent"
                            android:layout_height="match_parent"
                            android:layout_margin="3dp"
                            android:layout_weight="1"
                            android:backgroundTint="@color/black_shade_2"
                            android:padding="6dp"
                            android:text="7"
                            android:textColor="#fff"
                            android:textSize="15sp" />

                        <Button
                            android:id="@+id/b8"
                            android:layout_width="match_parent"
                            android:layout_height="match_parent"
                            android:layout_margin="3dp"
                            android:layout_weight="1"
                            android:backgroundTint="@color/black_shade_2"
                            android:padding="6dp"
                            android:text="8"
                            android:textColor="#fff"
                            android:textSize="15sp" />
```



```
                        <Button
                            android:id="@+id/b9"
                            android:layout_width="match_parent"
                            android:layout_height="match_parent"
                            android:layout_margin="3dp"
                            android:layout_weight="1"
                            android:backgroundTint="@color/black_shade_2"
                            android:padding="6dp"
                            android:text="9"
                            android:textColor="#fff"
                            android:textSize="15sp" />

                        <Button
                            android:id="@+id/bmul"
                            android:layout_width="match_parent"
                            android:layout_height="match_parent"
                            android:layout_margin="3dp"
                            android:layout_weight="1"
                            android:backgroundTint="@color/black_shade_2"
                            android:padding="6dp"
                            android:text="×"
                            android:textColor="#ffa500"
                            android:textSize="15sp" />

                    </LinearLayout>

                </LinearLayout>
```

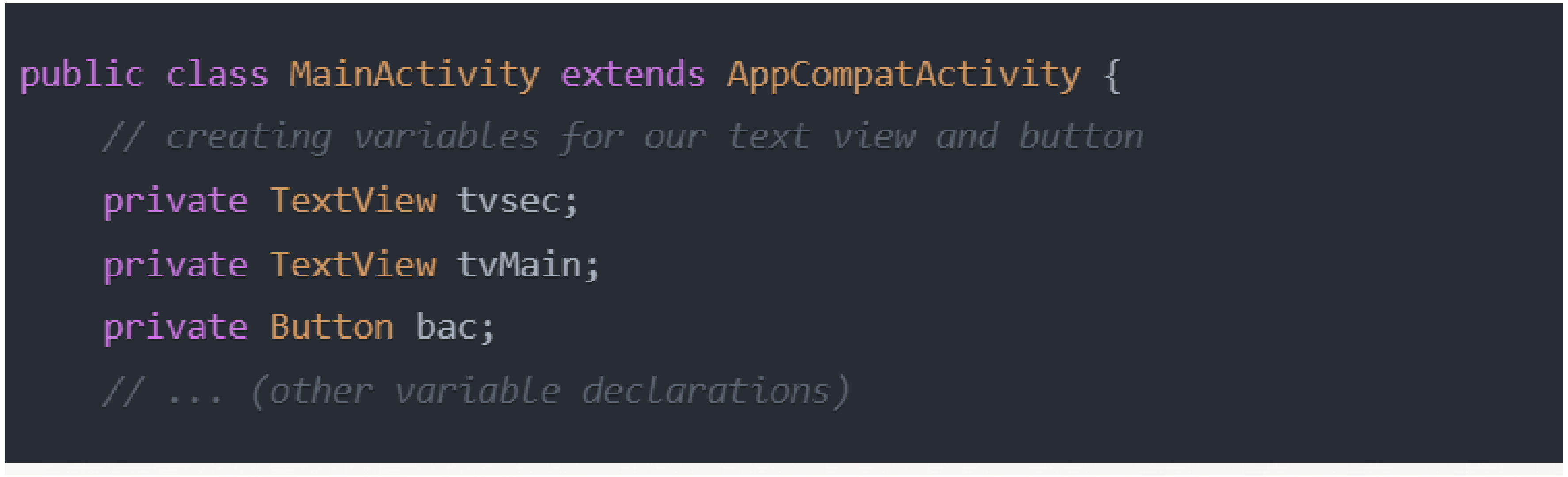
IMPORTING NECESSARY LIBRARIES:



```
import android.os.Bundle;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
```

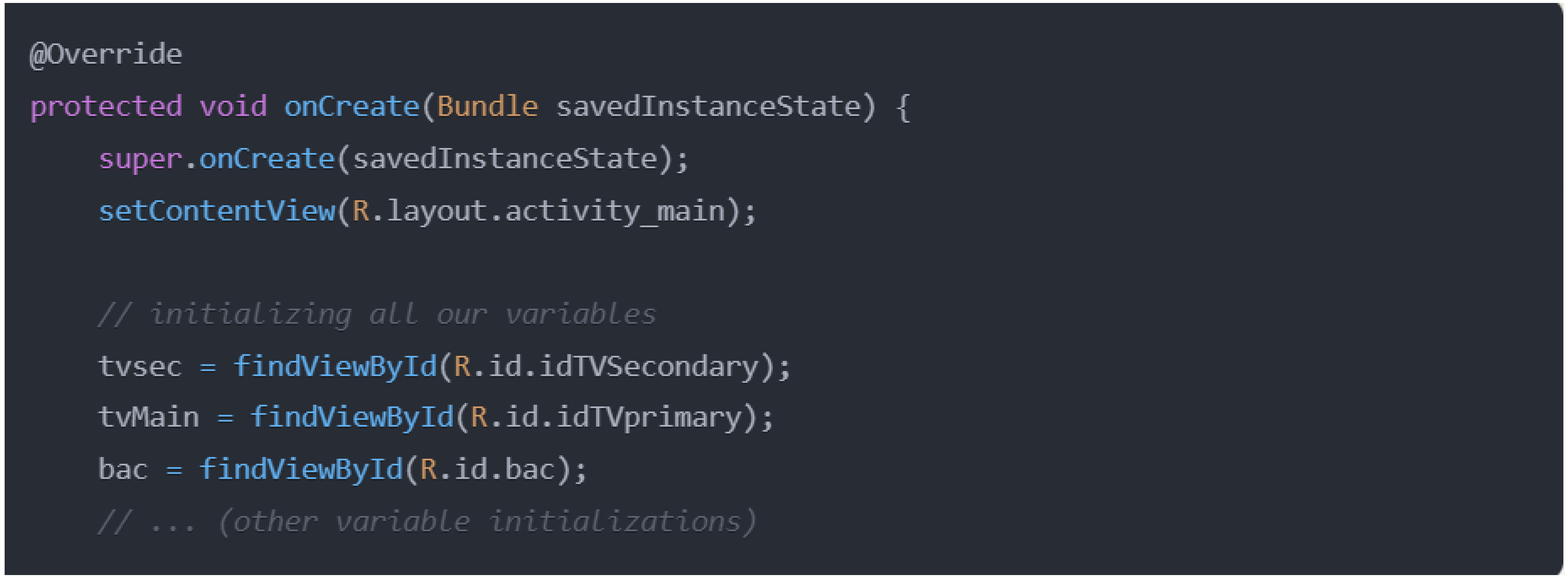
These lines import the necessary classes and libraries required for the code to function.

CREATING VARIABLES:



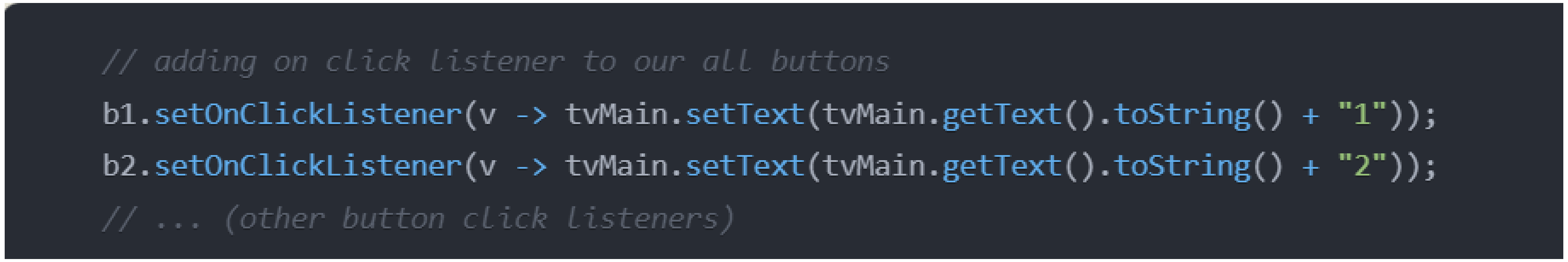
```java
public class MainActivity extends AppCompatActivity {
    // creating variables for our text view and button
    private TextView tvsec;
    private TextView tvMain;
    private Button bac;
    // ... (other variable declarations)
```

REFERENCING UI VIEWS WITH LAYOUT ID'S



```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // initializing all our variables
    tvsec = findViewById(R.id.idTVSecondary);
    tvMain = findViewById(R.id.idTVprimary);
    bac = findViewById(R.id.bac);
    // ... (other variable initializations)
```

SETUP EVENT LISTENER FOR EACH BUTTON

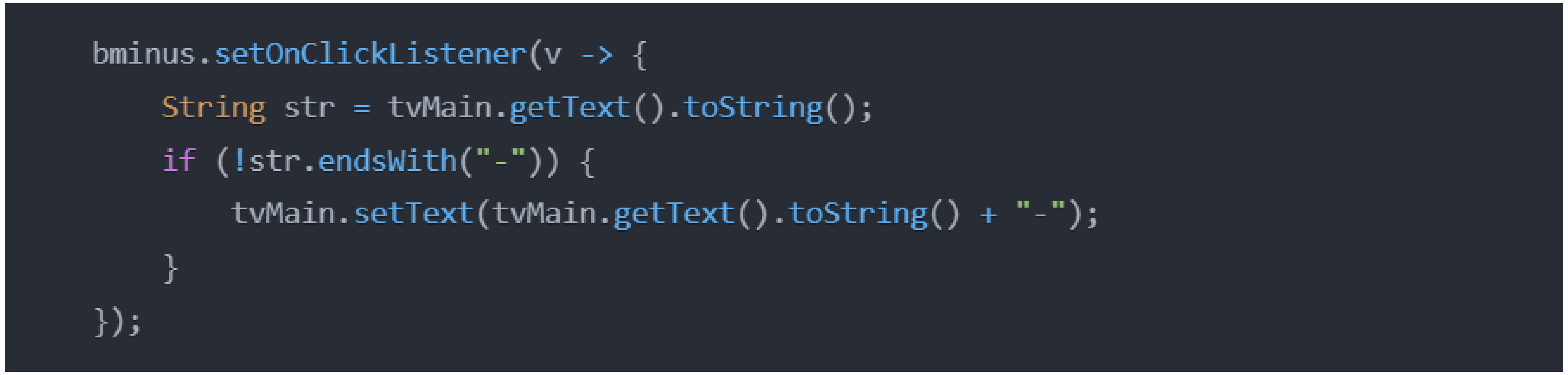


```
// adding on click listener to our all buttons
b1.setOnClickListener(v -> tvMain.setText(tvMain.getText().toString() + "1"));
b2.setOnClickListener(v -> tvMain.setText(tvMain.getText().toString() + "2"));
// ... (other button click listeners)
```

PURPOSE OF PRIMARY AND SECONDARY TEXT VIEW:

- ***tvMain*** is the main text view where the user's input and the final result are displayed, while ***tvsec*** is a secondary text view that provides additional information or displays the previous expression.

EVENT LISTENER EXAMPLE FOR MINUS BUTTON:

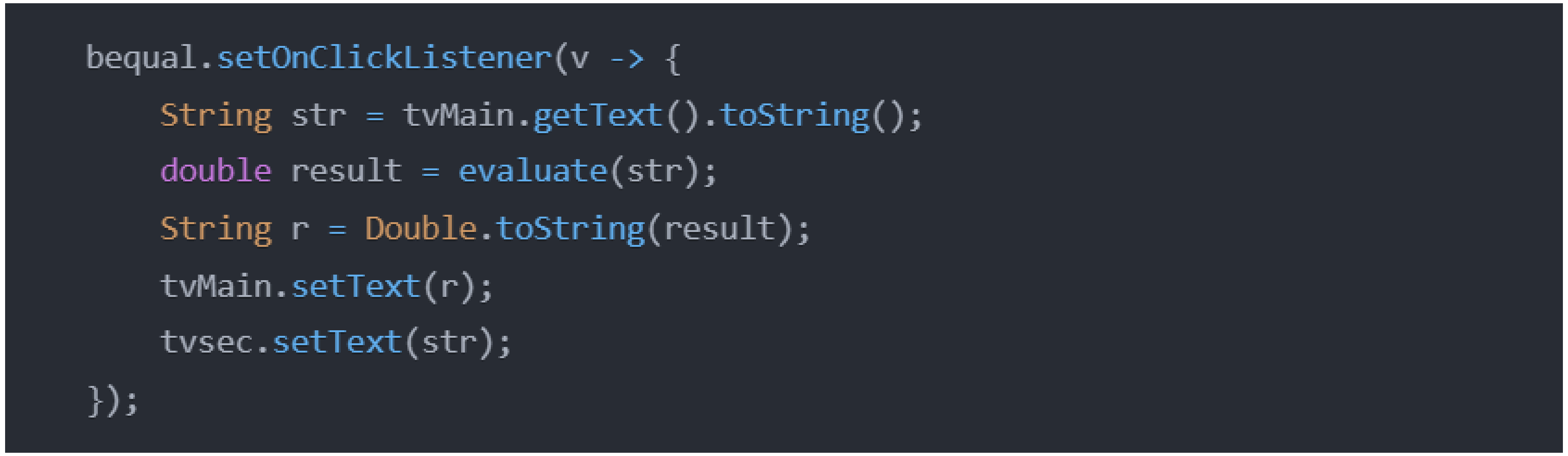


```
bminus.setOnClickListener(v -> {
    String str = tvMain.getText().toString();
    if (!str.endsWith("-")) {
        tvMain.setText(tvMain.getText().toString() + "-");
    }
});
```

EVENT LISTENER EXAMPLE FOR SQUARE ROOT BUTTON:

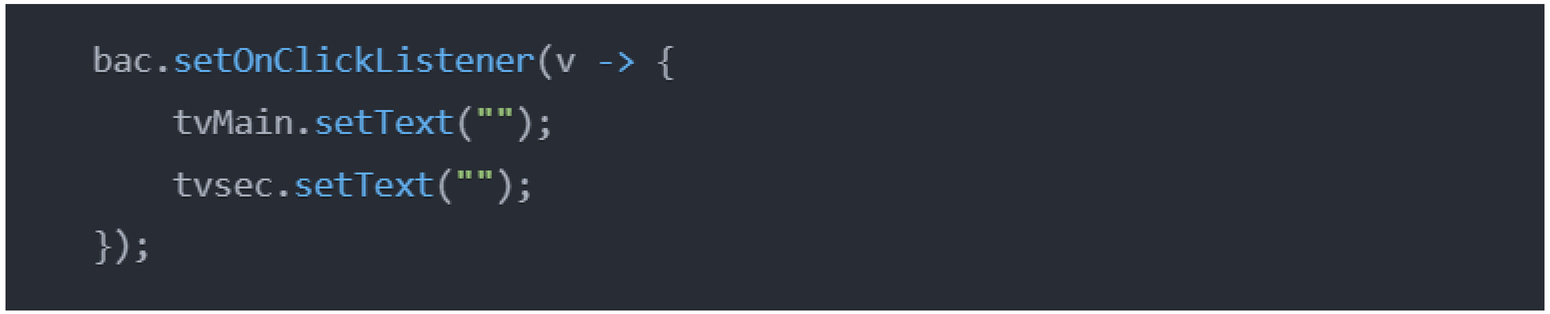
```java
bsqrt.setOnClickListener(v -> {
    if (tvMain.getText().toString().isEmpty()) {
        Toast.makeText(this, "Please enter a valid number..", Toast.LENGTH_SHORT).show();
    } else {
        String str = tvMain.getText().toString();
        double r = Math.sqrt(Double.parseDouble(str));
        String result = Double.toString(r);
        tvMain.setText(result);
    }
});
```

EVENT LISTENER EXAMPLE FOR EQUAL BUTTON:



```
bequal.setOnClickListener(v -> {
    String str = tvMain.getText().toString();
    double result = evaluate(str);
    String r = Double.toString(result);
    tvMain.setText(r);
    tvsec.setText(str);
});
```

EVENT LISTENER EXAMPLE FOR ALL CLEAR BUTTON:

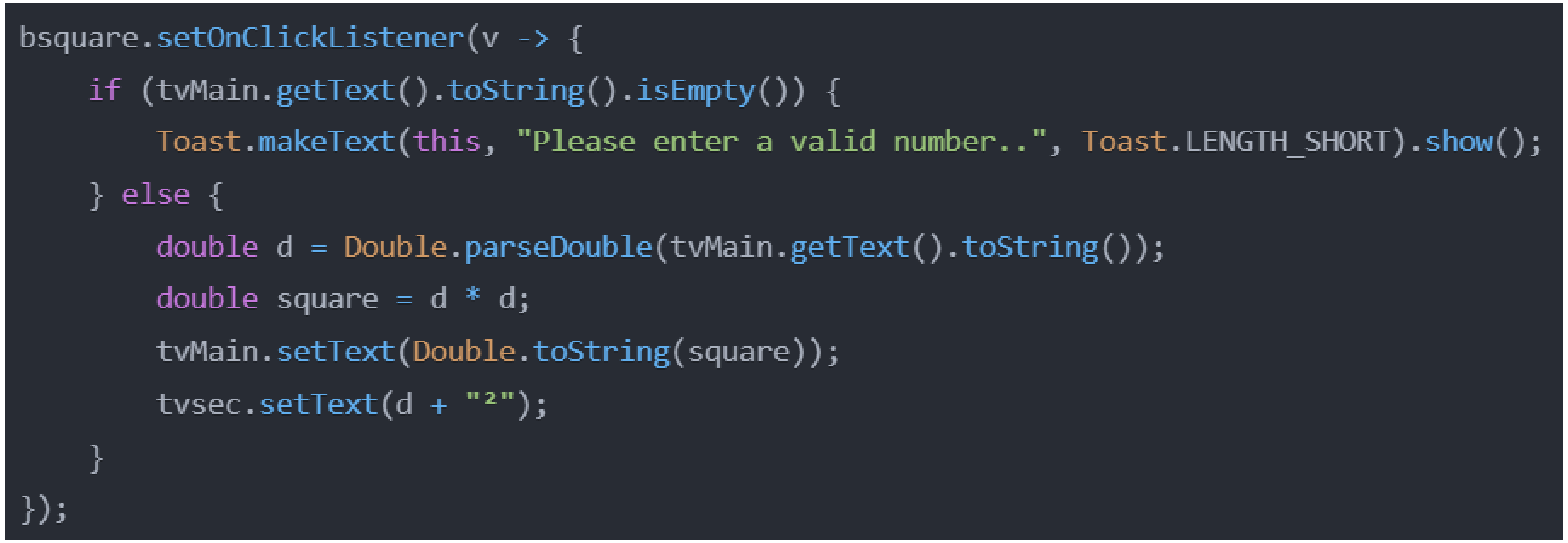


```
bac.setOnClickListener(v -> {
    tvMain.setText("");
    tvsec.setText("");
});
```

EVENT LISTENER EXAMPLE FOR CLEAR BUTTON:

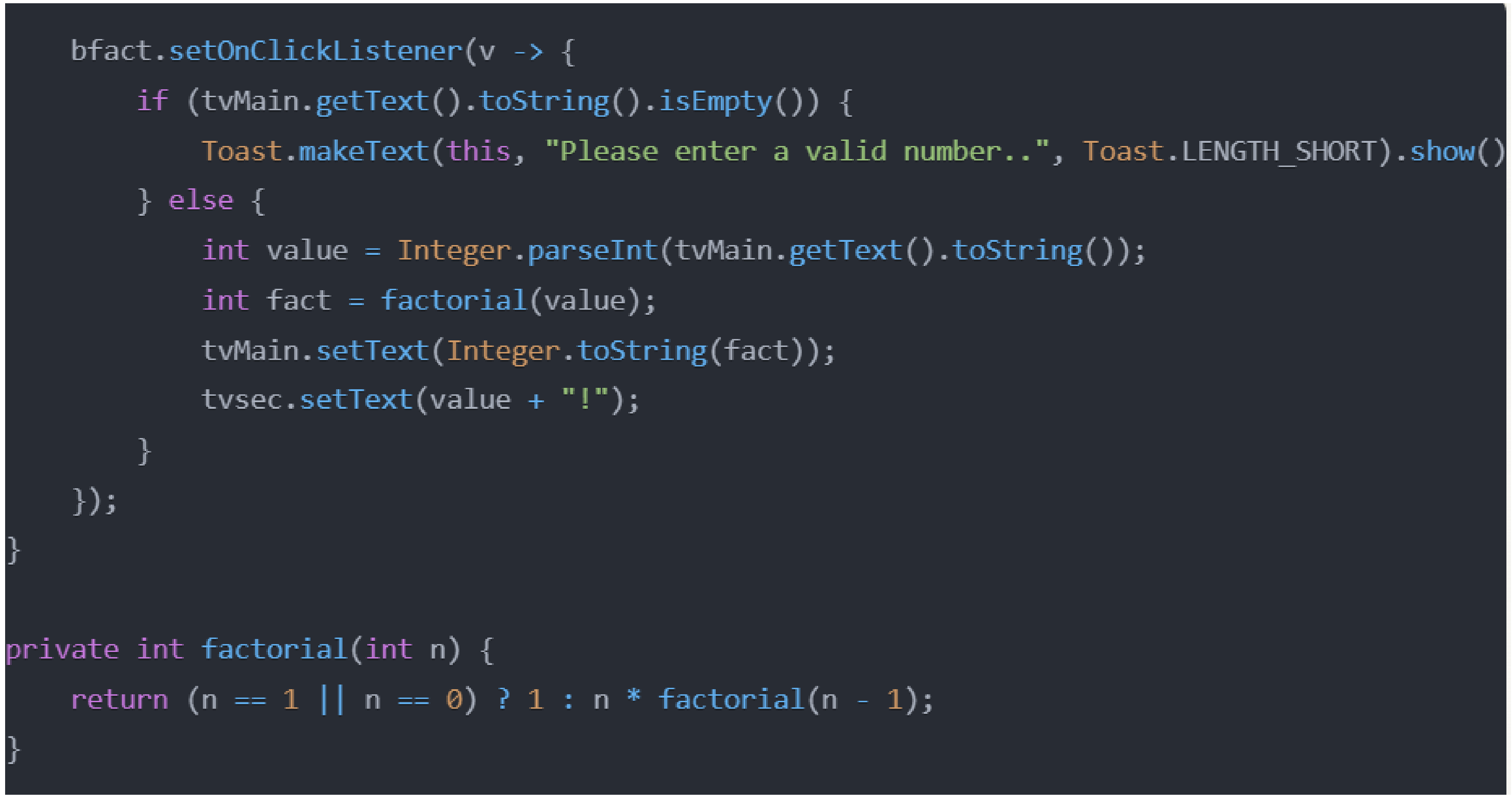
```java
bc.setOnClickListener(v -> {
    String str = tvMain.getText().toString();
    if (!str.isEmpty()) {
        str = str.substring(0, str.length() - 1);
        tvMain.setText(str);
    }
});
```

EVENT LISTENER EXAMPLE FOR SQUARE BUTTON:



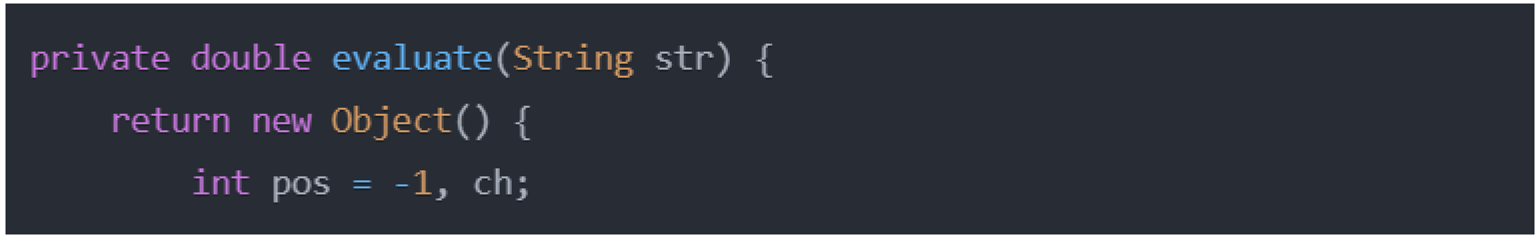
```java
bsquare.setOnClickListener(v -> {
    if (tvMain.getText().toString().isEmpty()) {
        Toast.makeText(this, "Please enter a valid number..", Toast.LENGTH_SHORT).show();
    } else {
        double d = Double.parseDouble(tvMain.getText().toString());
        double square = d * d;
        tvMain.setText(Double.toString(square));
        tvsec.setText(d + "²");
    }
});
```

EVENT LISTENER EXAMPLE FOR FACTORIAL BUTTON:



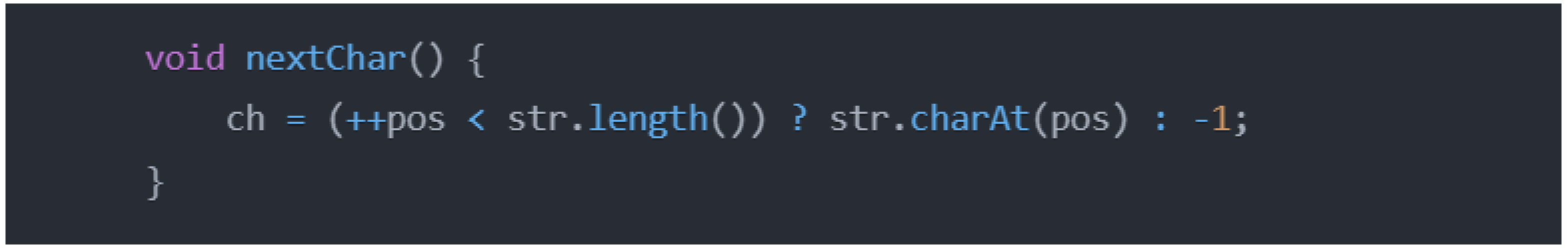
```java
    bfact.setOnClickListener(v -> {
        if (tvMain.getText().toString().isEmpty()) {
            Toast.makeText(this, "Please enter a valid number..", Toast.LENGTH_SHORT).show()
        } else {
            int value = Integer.parseInt(tvMain.getText().toString());
            int fact = factorial(value);
            tvMain.setText(Integer.toString(fact));
            tvsec.setText(value + "!");
        }
    });
}

private int factorial(int n) {
    return (n == 1 || n == 0) ? 1 : n * factorial(n - 1);
}
```

EVALUATE FUNCTION INITIALIZATION:



```
private double evaluate(String str) {
    return new Object() {
        int pos = -1, ch;
```

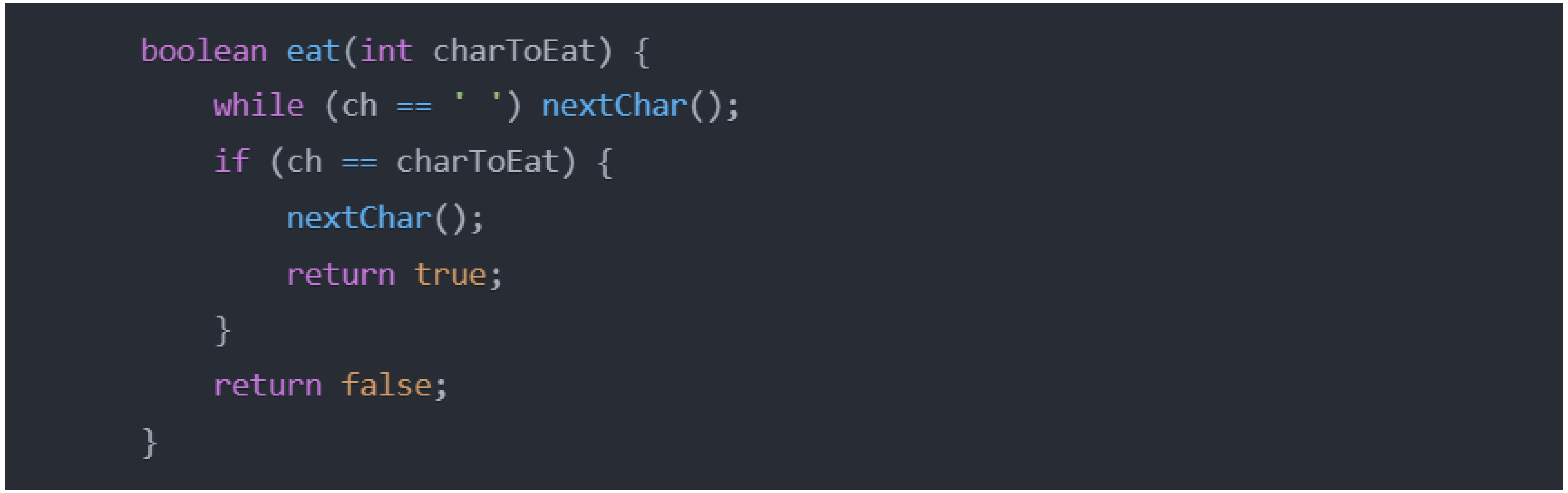
NEXT CHAR FUNCTION:



```java
void nextChar() {
    ch = (++pos < str.length()) ? str.charAt(pos) : -1;
}
```

The nextChar() method is a helper method that moves the position (pos) to the next character in the input string str. If the new position is within the string's bounds, it assigns the character at that position to ch. Otherwise, it assigns -1 to ch, indicating the end of the string.

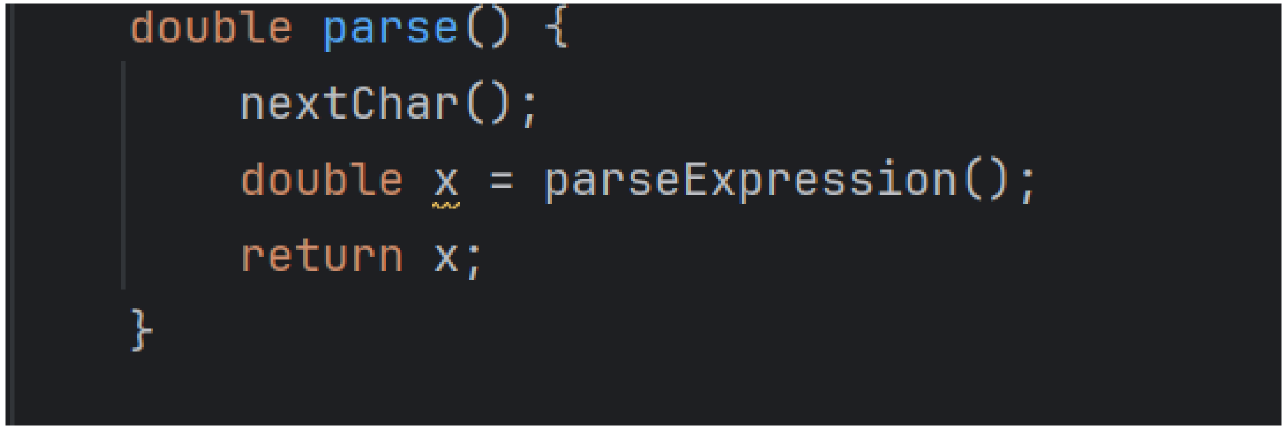
EAT FUNCTION TO CONSUME OPERATORS:



```java
boolean eat(int charToEat) {
    while (ch == ' ') nextChar();
    if (ch == charToEat) {
        nextChar();
        return true;
    }
    return false;
}
```

The eat() method is another helper method that checks if the current character (ch) matches the given charToEat. It first skips any whitespace characters by calling nextChar() until it encounters a non-whitespace character. If the current character matches charToEat, it moves to the next character by calling nextChar() and returns true. Otherwise, it returns false.

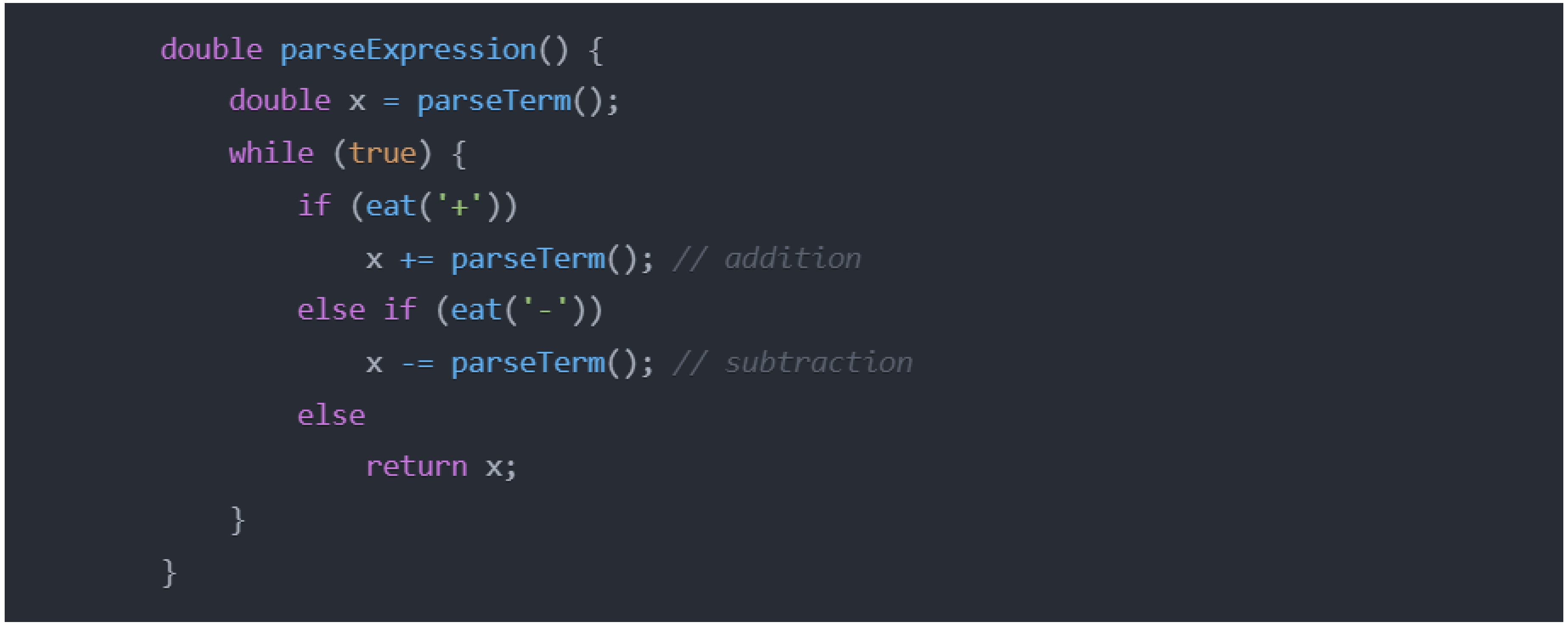
PARSE FUNCTION TO PARSE STRINGS:



```
double parse() {
    nextChar();
    double x = parseExpression();
    return x;
}
```

The parse() method is the entry point of the parser. It calls nextChar() to move to the first character of the input string. Then, it calls parseExpression() to parse the expression and returns the result.

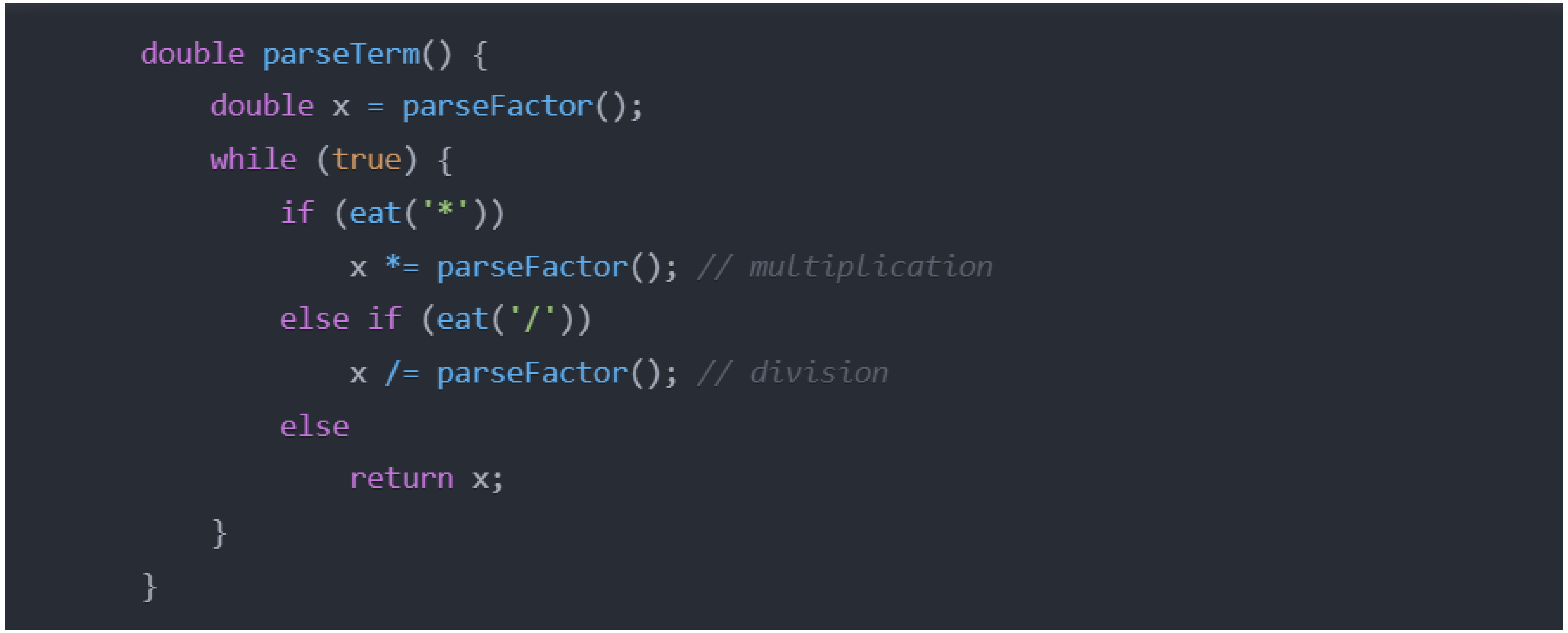
PARSE EXPRESSION METHOD TO HANDLE (+ & -) :



```
double parseExpression() {
    double x = parseTerm();
    while (true) {
        if (eat('+'))
            x += parseTerm(); // addition
        else if (eat('-'))
            x -= parseTerm(); // subtraction
        else
            return x;
    }
}
```

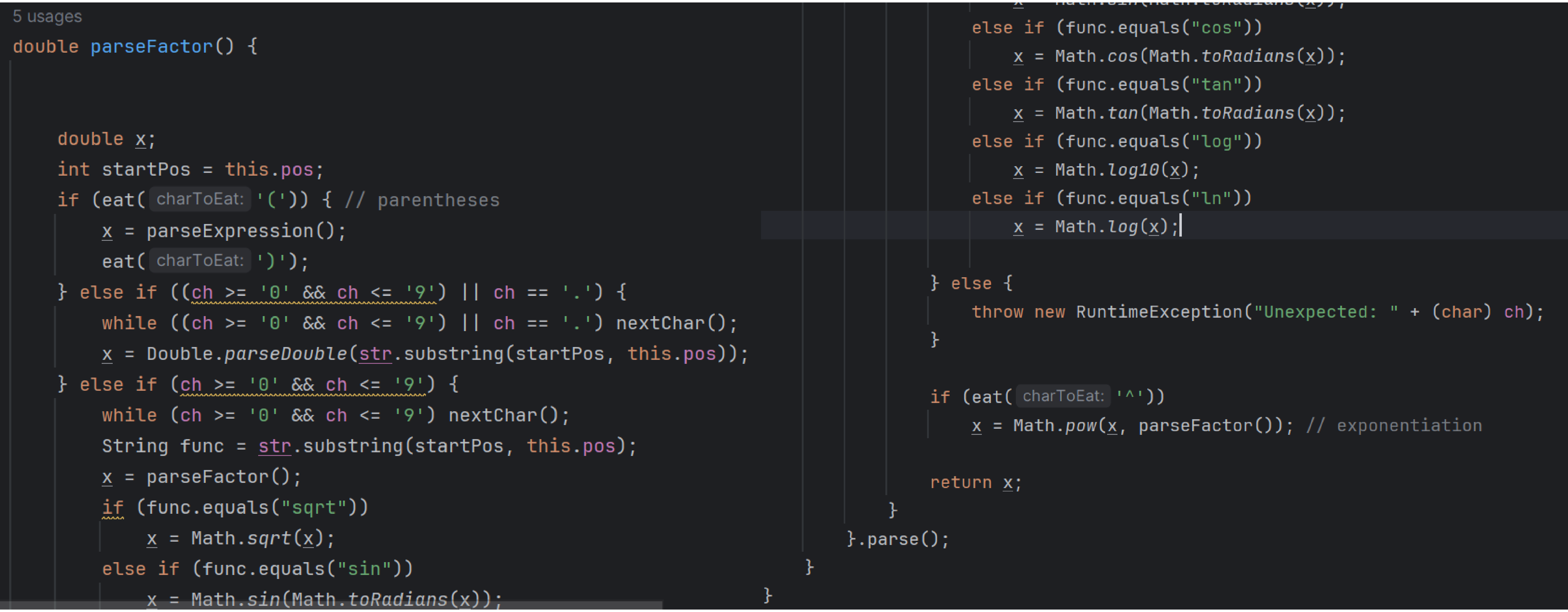
The parseExpression() method handles addition and subtraction operations. It first calls parseTerm() to parse the first term of the expression and assigns the result to x. Then, it enters a loop where it checks for the + or - operator using the eat() method. If the operator is found, it performs the corresponding operation (addition or subtraction) with the result of calling parseTerm() again. The loop continues until neither + nor - is found, at which point it returns x.

PARSE TERM METHOD TO HANDLE (* & /)

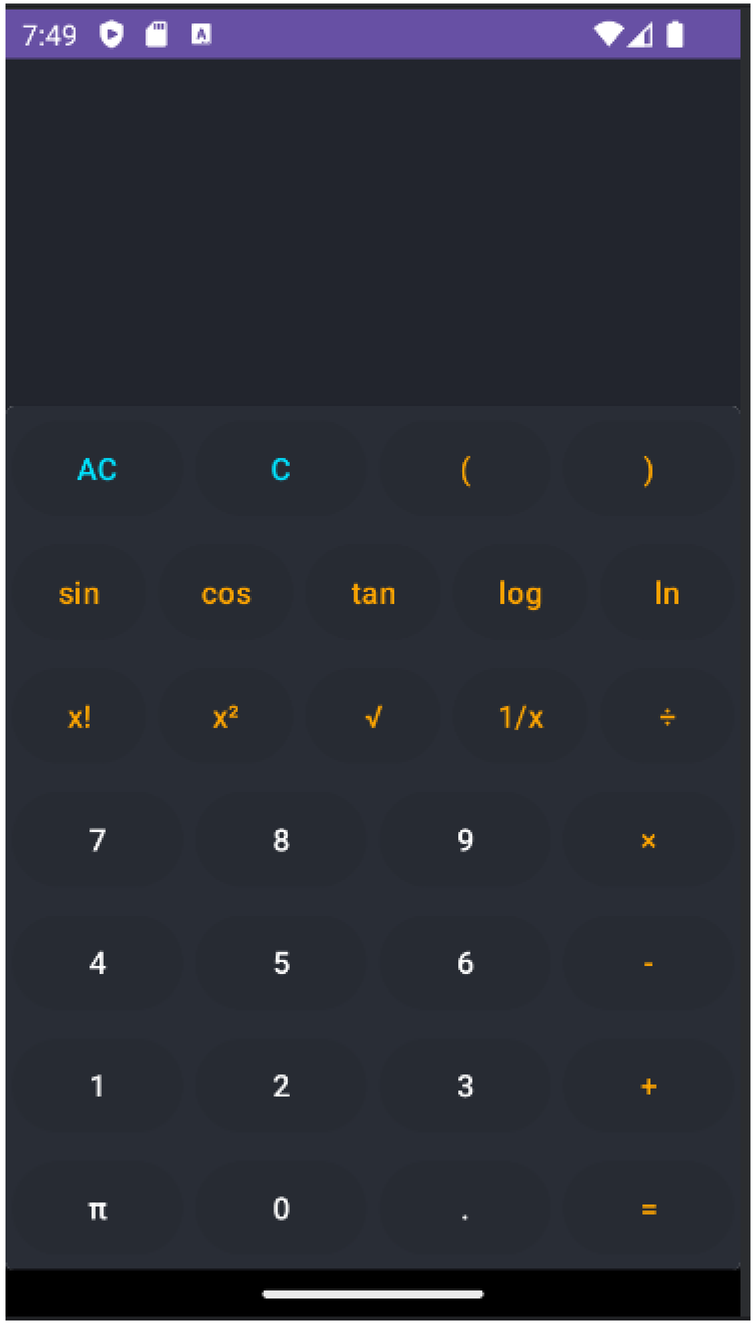


```
double parseTerm() {
    double x = parseFactor();
    while (true) {
        if (eat('*'))
            x *= parseFactor(); // multiplication
        else if (eat('/'))
            x /= parseFactor(); // division
        else
            return x;
    }
}
```

PARSE FACTOR METHOD TO HANDLE INDIVIDUAL CHARACTER:



```java
            5 usages
            double parseFactor() {


                double x;
                int startPos = this.pos;
                if (eat( charToEat: '(')) { // parentheses
                    x = parseExpression();
                    eat( charToEat: ')');
                } else if ((ch >= '0' && ch <= '9') || ch == '.') {
                    while ((ch >= '0' && ch <= '9') || ch == '.') nextChar();
                    x = Double.parseDouble(str.substring(startPos, this.pos));
                } else if (ch >= '0' && ch <= '9') {
                    while (ch >= '0' && ch <= '9') nextChar();
                    String func = str.substring(startPos, this.pos);
                    x = parseFactor();
                    if (func.equals("sqrt"))
                        x = Math.sqrt(x);
                    else if (func.equals("sin"))
                        x = Math.sin(Math.toRadians(x));
                    else if (func.equals("cos"))
                        x = Math.cos(Math.toRadians(x));
                    else if (func.equals("tan"))
                        x = Math.tan(Math.toRadians(x));
                    else if (func.equals("log"))
                        x = Math.log10(x);
                    else if (func.equals("ln"))
                        x = Math.log(x);

                } else {
                    throw new RuntimeException("Unexpected: " + (char) ch);
                }

                if (eat( charToEat: '^'))
                    x = Math.pow(x, parseFactor()); // exponentiation

                return x;
            }
        }.parse();
    }
}
```



TASKS:

- Implement the Fully Functional Calculator in Android Studio As per given Picture: