

Bài 05: List Layout và Grid Layout

MỤC TIÊU

Sau bài học này sinh viên sẽ:

- Biết và sử dụng được các loại component Scaffold, List và Grid;
- Ôn lại Lamda và Higher-Order function
- Thiết kế giao diện và hiện thực các chức năng cho máy tính đơn giản.

PHƯƠNG PHÁP

- Đọc hiểu tài liệu
- Thực hành hướng dẫn trong bài học
- Làm bài tập

NỘI DUNG

1. Scaffold và các thành phần giao diện

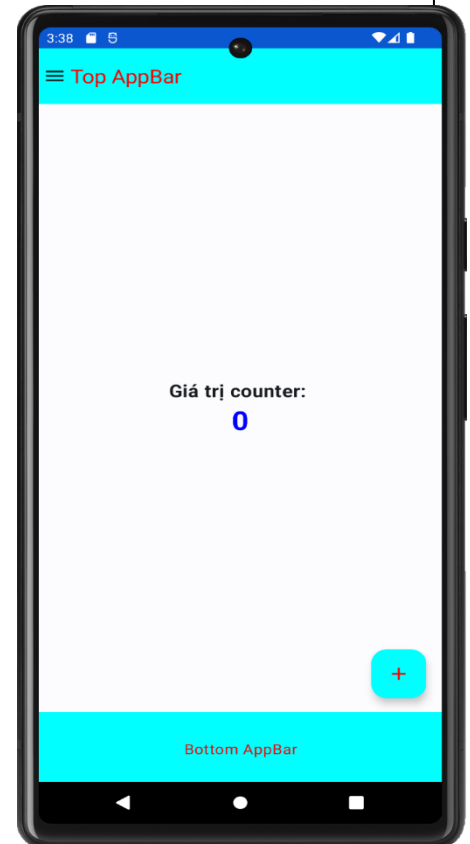
1.1. Scaffold

Scaffold là một thành phần giao diện trong Jetpack Compose, nó cung cấp cấu trúc layout chuẩn cho việc xây dựng giao diện của ứng dụng. Scaffold cung cấp API cơ bản để sử dụng, nó thực hiện theo cấu trúc thiết kế của Material Design (phiên bản tại thời điểm viết tài liệu là Material 3), các thành phần của Scaffold cấu thành bởi nhiều tham số từ các composable:

- | | |
|--|--|
| - topBar: thanh ứng dụng phía trên | hình dùng để thực hiện một tác vụ nào đó |
| - bottonBar: thanh ứng dụng phía dưới | của ứng dụng. |
| - floatingActionButton: một nút nổi trên màn | - content: thể hiện nội dung chính |

```
@ExperimentalMaterial3Api
@Composable
@ComposableInferredTarget
public fun Scaffold(
    modifier: Modifier,
    topBar: @Composable () -> Unit,
    bottomBar: @Composable () -> Unit,
    snackbarHost: @Composable () -> Unit,
    floatingActionButton: @Composable () -> Unit,
    floatingActionButtonPosition: FabPosition,
    containerColor: Color,
    contentColor: Color,
    contentWindowInsets: WindowInsets,
    content: @Composable (PaddingValues) -> Unit
): Unit
```

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun ScaffoldSampleScreen(modifier: Modifier = Modifier) {
    var counter = remember { mutableStateOf(0) }
    Scaffold(
        topBar = {
            TopAppBar(
                title = { Text(text = "Top AppBar") },
                colors = TopAppBarDefaults.smallTopAppBarColors(
                    containerColor = Color.Cyan,
                    titleContentColor = Color.Red
                ),
                navigationIcon = {
                    Icon(
                        imageVector = Icons.Sharp.Menu,
                        contentDescription = null
                    )
                }
            )
        },
        bottomBar = {
            BottomAppBar(
                containerColor = Color.Cyan,
                contentColor = Color.Red
            ) {
                Text(
                    text = "Bottom AppBar",
                    modifier = modifier.fillMaxWidth(),
                    textAlign = TextAlign.Center
                )
            }
        },
        floatingActionButton = {
            FloatingActionButton(
                onClick = { counter.value++ },
                containerColor = Color.Cyan
            ) {
                Icon(
                    imageVector = Icons.Sharp.Add,
                    contentDescription = null,
                    tint = Color.Red
                )
            }
        }
    ) {
        Column(
            modifier = modifier
                .padding(it)
                .fillMaxSize(),
            verticalArrangement = Arrangement.Center,
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            Text(text = "Giá trị counter: ",
                fontSize = 20.sp,
                fontWeight = FontWeight.Bold)
            Text(
                text = "${counter.value}",
                fontSize = 30.sp,
                fontWeight = FontWeight.Bold,
                color = Color.Blue
            )
        }
    }
}
```



Để sử dụng Scaffold trong một Composable, cần bổ sung khai báo lớp hiện thực theo Material3 Api ở phía trước hàm đó

```
@OptIn(ExperimentalMaterial3Api::class)
```

Các thành phần con của nội dung Scaffold cần sử dụng **innerPadding** (ví dụ trên dùng **it** đại diện innerPadding- sử dụng cú pháp lambda)

1.2. AppBar

App bar là container cung cấp cho người dùng các chức năng và điều hướng (navigation) ứng dụng. Có hai loại app bar: top app bar, bottom app bar.

- Top app bar: cung cấp người dùng thao tác truy cập đến các chức năng chính của ứng dụng, nó thường chứa: tiêu đề (title), các tác vụ cốt lõi, điều hướng.
- Bottom app bar: thường chứa các điều hướng ứng dụng, và các thao tác đến các chức năng khác của ứng dụng.

Để hiện thực top app bar, bottom app bar thường dùng composable: TopAppBar và BottomAppBar

```
@ExperimentalMaterial3Api
@Composable
@ComposableInferredTarget
public fun TopAppBar(
    title: @Composable () -> Unit,
    modifier: Modifier,
    navigationIcon: @Composable () -> Unit,
    actions: @Composable() (RowScope.) -> Unit,
    windowInsets: WindowInsets,
    colors: TopAppBarColors,
    scrollBehavior: TopAppBarScrollBehavior?
): Unit

@Composable
@ComposableInferredTarget
public fun BottomAppBar(
    modifier: Modifier,
    containerColor: Color,
    contentColor: Color,
    tonalElevation: Dp,
    contentPadding: PaddingValues,
    windowInsets: WindowInsets,
    content: @Composable() (RowScope.) -> Unit
): Unit
```

1.3. FloatingActionButton

Floating Action Button (FBA) là một nút nhấn nổi, nhấn mạnh cho người dùng thực hiện thao tác quan trọng trong ứng dụng. Thông thường FBA nằm nổi ở dưới góc phải của màn hình ứng dụng.

Có 4 loại FBA:

- FBA gốc: FloatingActionButton
- FBA kích thước nhỏ: SmallFloatingActionButton
- FBA kích thước lớn: LargeFloatingActionButton
- FBA mở rộng: ExtendedFloatingActionButton



Các tham số chính của các FBA:

- onClick: xử lý sự kiện nhấn vào FBA
- containerColor: màu sắc của FBA
- contentColor: màu của icon và nội dung hiển thị bên trong FBA

2. List và Grid

2.1. List

Ứng dụng có thể có nhiều màn hình, các màn hình có thể chứa danh sách các phần tử mà người dùng có thể cuộn để duyệt các phần tử đó. Compose hỗ trợ thực hiện điều đó thông qua **Column** và **Row**; để cuộn duyệt các phần tử, có thể sử dụng phương thức **verticalScroll()** ứng với **Column** của đối tượng **modifier** (phương thức **horizontalScroll()** ứng với **Row**).

Các dạng thức khác của **Column** và **Row** thể hiện cho **List (LazyList)** là **LazyColumn** và **LazyRow**. Khi danh sách phần tử quá lớn, **Column (Row)** nó sẽ không mang lại hiệu suất tốt, bởi tất cả các phần tử được bố trí bên trong nó luôn được thực thi cho việc hiển thị (render tất cả phần tử) mặc dù chúng được nhìn thấy hay không (trường hợp cuộn). Còn **LazyColumn (LazyRow)** nó thực thi hiển thị những phần tử cần thiết trong phần khung nhìn của ứng dụng (**viewport**).

Hơn thế nữa, **LazyColumn (LazyRow)** cung cấp sẵn cơ chế hỗ trợ duyệt danh sách phần tử theo chiều dọc (chiều ngang) mà không cần phải sử dụng phương thức **verticalScroll()** (**horizontalScroll()**) của **modifier** như **Column (Row)**.

Các thành phần của **LazyColumn (LazyRow)** là các **item** riêng lẻ hoặc danh sách các phần tử **items**.

Ví dụ:

```
LazyColumn {  
    item { Text(text = "Phan tu thu 1") }  
    items(5) {  
        Text(text = "Phan tu thu ${it+2}")  
    }  
}
```

LazyColumn và LazyRow:

```
@Composable
@ComposableTarget
public fun LazyColumn(
    modifier: Modifier,
    state: LazyListState,
    contentPadding: PaddingValues,
    reverseLayout: Boolean,
    verticalArrangement: Arrangement.Vertical,
    horizontalAlignment: Alignment.Horizontal,
    flingBehavior: FlingBehavior,
    userScrollEnabled: Boolean,
    content: LazyListScope.() -> Unit
): Unit
```

```
@Composable
@ComposableTarget
public fun LazyColumn(
    modifier: Modifier,
    state: LazyListState,
    contentPadding: PaddingValues,
    reverseLayout: Boolean,
    verticalArrangement: Arrangement.Vertical,
    horizontalAlignment: Alignment.Horizontal,
    flingBehavior: FlingBehavior,
    userScrollEnabled: Boolean,
    content: LazyListScope.() -> Unit
): Unit
```

2.2. Grid

Grid hiển thị danh sách các phần tử, bố trí các phần tử theo dạng lưới (theo các hàng, cột). Compose cung cấp hai Composable của Grid là **LazyVerticalGrid** và **LazyHorizontalGrid**. Trong đó, **LazyVerticalGrid** hiển thị các phần tử (item) trong một container có thể cuộn duyệt theo chiều dọc, các phần tử bố trí theo nhiều cột, còn **LazyHorizontalGrid** bố trí các phần tử theo nhiều dòng và cuộn duyệt theo chiều ngang.

Mỗi phần tử trong Grid được bố trí theo một ô (Cell), Hiện thực tham số **columns** để điều khiển bố trí các phần tử đối với **LazyVerticalGrid** và **rows** đối với **LazyHorizontalGrid**. Để hiện thực chúng có

thể sử dụng **GridCells.Adaptive()** để xác định khoảng trống tối thiểu giữa các cell, **GridCells.Fixed()** để bố trí cố định số cột hoặc số dòng.

```
@Composable
@ComposableTarget
public fun LazyVerticalGrid(
    columns: GridCells,
    modifier: Modifier,
    state: LazyGridState,
    contentPadding: PaddingValues,
    reverseLayout: Boolean,
    verticalArrangement: Arrangement.Vertical,
    horizontalArrangement: Arrangement.Horizontal,
    flingBehavior: FlingBehavior,
    userScrollEnabled: Boolean,
    content: LazyGridScope.() -> Unit
): Unit
```

```
@Composable
@ComposableTarget
public fun LazyHorizontalGrid(
    rows: GridCells,
    modifier: Modifier,
    state: LazyGridState,
    contentPadding: PaddingValues,
    reverseLayout: Boolean,
    horizontalArrangement: Arrangement.Horizontal,
    verticalArrangement: Arrangement.Vertical,
    flingBehavior: FlingBehavior,
    userScrollEnabled: Boolean,
    content: LazyGridScope.() -> Unit
): Unit
```

LazyVerticalGrid và **LazyHorizontalGrid**, các phần tử có kích thước bằng nhau. Trong trường hợp kích thước không bằng nhau, Compose cung cấp **Composable: LazyVerticalStaggeredGrid** (các phần tử có độ cao khác nhau), **LazyHorizontalStaggeredGrid** (các phần tử có độ rộng khác nhau). Để hiện thực bố trí các phần tử theo tham số **columns** hoặc **rows** dùng các phương thức tương tự grid trên là: **StaggeredGridCells.Adaptive()** và **StaggeredGridCells.Fixed()**

```
@ExperimentalFoundationApi
@Composable
@ComposableTarget
public fun LazyVerticalStaggeredGrid(
    columns: StaggeredGridCells,
    modifier: Modifier,
    state: LazyStaggeredGridState,
    contentPadding: PaddingValues,
    reverseLayout: Boolean,
    verticalItemSpacing: Dp,
    horizontalArrangement: Arrangement.Horizontal,
    flingBehavior: FlingBehavior,
    userScrollEnabled: Boolean,
    content: LazyStaggeredGridScope.() -> Unit
): Unit
```

```
@ExperimentalFoundationApi
@Composable
@ComposableTarget
public fun LazyHorizontalStaggeredGrid(
    rows: StaggeredGridCells,
    modifier: Modifier,
    state: LazyStaggeredGridState,
    contentPadding: PaddingValues,
    reverseLayout: Boolean,
    verticalArrangement: Arrangement.Vertical,
    horizontalItemSpacing: Dp,
    flingBehavior: FlingBehavior,
    userScrollEnabled: Boolean,
    content: LazyStaggeredGridScope.() -> Unit
): Unit
```

2.3. Các cách hiện thực phần tử trong List và Grid

Các phần tử (item) có thể đứng độc lập hoặc thuộc một danh sách (items), mỗi phần tử có một khoá mặc định là vị trí của phần tử trong danh sách. Khoá của phần tử có thể được chỉ định lại theo mục đích khai thác tương ứng với chức năng trên ứng dụng.

Ví dụ:

```
LazyColumn {  
    items(books, key = { it.id }) { // Khoá theo thuộc tính khoá của đối tượng book  
        //Hiện thực giao diện phần tử  
    }  
}
```

3. Hiện thực máy tính đơn giản

3.1. Yêu cầu

Xây dựng ứng dụng máy tính đơn giản. Người dùng nhập vào biểu thức từ bàn phím. Sau đó, nhấn phím =, biểu thức sẽ được tính toán và hiển thị kết quả:

Một số phím đặc biệt:

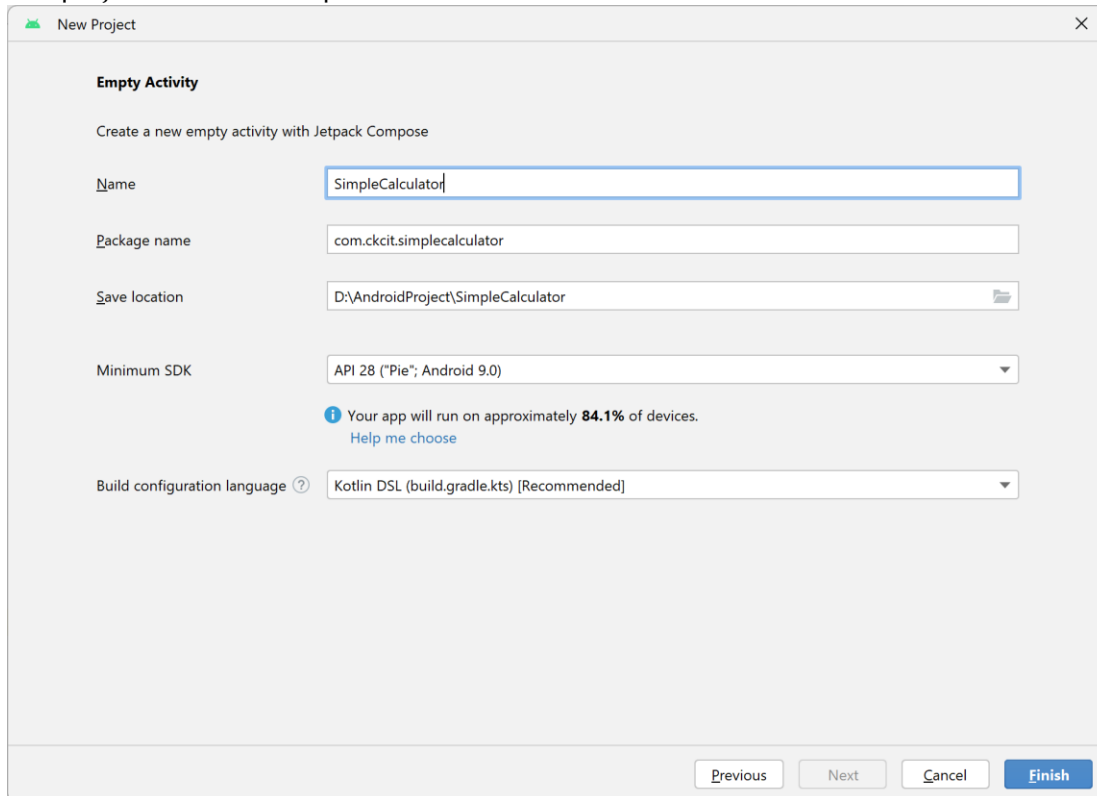
- C: xoá biểu thức
- DEL: xoá ký tự cuối của biểu thức
- %: tỉ lệ phần trăm
- +,-,x,/: cộng, trừ, nhân, chia
- Ans: lấy kết quả đưa vào biểu thức



3.2. Hướng dẫn

➤ Tạo Project và thiết kế giao diện

Tạo project có tên là Simple Calculator



Chọn template Empty Activity, xoá hàm composable mặc định, khai báo hàm **SimpleCalculatorScreen**

```
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContent {  
            SimpleCalculatorTheme {  
                Surface(  
                    modifier = Modifier.fillMaxSize(),  
                    color = MaterialTheme.colorScheme.background  
                ) {  
                    SimpleCalculatorScreen()  
                }  
            }  
        }  
    }  
}  
  
@Composable  
fun SimpleCalculatorScreen(modifier: Modifier=Modifier){  
}  
  
@Preview(showBackground = true)  
@Composable  
fun GreetingPreview() {  
    SimpleCalculatorTheme {  
        SimpleCalculatorScreen()  
    }  
}
```

Phân tích giao diện: Giao diện có thanh AppBar với tiêu đề là Text (Máy tính đơn giản), phần nội dung màn hình giao diện chia làm hai phần: **Phần Biểu Thức và Kết Quả** chứa 2 TextField, **Phần Bàn Phím** chứa các phím số và các phím chức năng (Button), giữa hai phần này có đường kẻ phân chia. Các thành phần này được bố trí theo chiều dọc của màn hình ứng dụng.

Phân tích chức năng cho các thành phần giao diện: Phần Biểu Thức và Kết Quả chỉ hiện thị nội dung khi người dùng thao tác nhấn trên các phím ở Phần Bàn Phím

- Khai báo 2 biến trạng thái gán giá trị này cho TextField biểu thức và kết quả, kèm danh sách nhấn các phím trong hàm composable **SimpleCalculatorScreen** vừa tạo

```
var _expression by remember { mutableStateOf("") }
var _result by remember { mutableStateOf("0") }
var lsKeys = listOf<String>("C", "DEL", "%", "/",
    "7", "8", "9", "x",
    "4", "5", "6", "-",
    "1", "2", "3", "+",
    "Ans", "0", ".", "="
)
```

- Thiết kế giao diện màn hình có **AppBar** (bổ sung (@OptIn(ExperimentalMaterial3Api::class)))

```
Scaffold(
    topBar = {
        TopAppBar(
            title = {
                Text(
                    text = "Máy Tính Đơn Giản",
                    fontWeight = FontWeight.Bold
                )
            },
            colors = TopAppBarDefaults.smallTopAppBarColors(
                containerColor = Color.Blue,
                titleContentColor = Color.White
            )
        )
    }
) {
    Column(modifier = modifier.padding(it)) {
    }
}
```

- Thiết kế **Textfield biểu thức** bên trong Column trên

```
TextField(
    value = _expression, onChange = { _expression = it },
    modifier = modifier.fillMaxWidth(), singleLine = true,
    readOnly = true,
    textStyle = TextStyle(fontSize = 45.sp, textAlign = TextAlign.Right),
    colors = TextFieldDefaults.textFieldColors(
        textColor = Color.Blue,
        containerColor = Color.White
    )
)
```

Tuy nhiên, TextField này mặc định có border ở dưới (bottom border), có thể tắt nó bằng cách chỉnh màu trong suốt khi focus và không focus TextField này trong **textFieldColors** composable

```
colors = TextFieldDefaults.textFieldColors(  
    textColor = Color.Blue,  
    containerColor = Color.White,  
    focusedIndicatorColor = Color.Transparent,  
    unfocusedIndicatorColor = Color.Transparent  
)
```

- Tương tự, cho phần **TextField kết quả** đặt sau TextField biểu thức (bên trong Column)

```
TextField(  
    value = _result, onValueChange = { _result = it },  
    modifier = modifier.fillMaxWidth(), singleLine = true,  
    readOnly = true,  
    textStyle = TextStyle(fontSize = 35.sp, textAlign = TextAlign.Right),  
    colors = TextFieldDefaults.textFieldColors(  
        textColor = Color.Blue,  
        containerColor = Color.White,  
        focusedIndicatorColor = Color.Transparent,  
        unfocusedIndicatorColor = Color.Transparent  
    )  
)
```

- Thêm **đường thẳng phân chia** giữa các TextField với bàn phím

```
Divider(  
    modifier=modifier.padding(top = 10.dp, bottom = 10.dp),  
    color = Color.LightGray  
)
```

- Thiết kế sơ lược phần Bàn Phím: các phím này được bố trí theo các hàng, mỗi hàng 4 thành phần cố định

```
LazyVerticalGrid(  
    modifier=modifier.fillMaxSize()  
        .padding(top=40.dp, start = 12.dp, end = 12.dp), //chỉnh theo màn hình  
    verticalArrangement = Arrangement.SpaceEvenly,  
    horizontalArrangement = Arrangement.SpaceEvenly,  
    columns = GridCells.Fixed(4),  
    content = {  
        items(lsKeys.count()) {  
            Text(text = lsKeys[it])  
        }  
    }  
)
```

Các phím trên đây chỉ tạm thời thiết kế bởi các Text, nên hiện thực chúng bởi Button và xử lý sự kiện khi nhấn chúng.

➤ Hiện thực các phím chức năng

Do các phím này có phần thiết kế và xử lý tương tự nhau, bài thực hành sẽ xây dựng hàm composable chung, với các tham số:

- o Nhấn phím
- o Màu nền của phím (nếu có)
- o Hàm xử lý sự kiện khi người dùng nhấn phím

Composable này thuộc dạng composable stateless (nó có ưu điểm có thể tái sử dụng được, composable stateful khó để tái sử dụng bởi các trạng thái của nó thường kèm xử lý giao diện).

Hàm này cũng dạng Higher-Order Function, khi tham số truyền vào bởi một hàm khác (hàm xử lý sự kiện được truyền vào dạng tham số - Nó sẽ được hiện thực trực tiếp trong hàm **SimpleCalculatorScreen** để dễ dàng cho việc set giá trị cho biểu thức và kết quả khi người dùng nhấn các phím)

```
@Composable
fun KeyFunction(
    label: String,
    containerColor: Color = Color.LightGray,
    onClick: (String) -> Unit,
    modifier: Modifier = Modifier
) {
    Button(
        modifier = modifier
            .size(70.dp, 70.dp)
            .clip(shape = CircleShape),
        onClick = { onClick(label) },
        colors = ButtonDefaults.buttonColors(
            containerColor = containerColor,
            contentColor = Color.Red
        )
    ) {
        Text(
            modifier=modifier.fillMaxWidth(),
            text = label,
            color = Color.Red,
            fontSize = 18.sp,
            textAlign = TextAlign.Center,
            fontWeight = FontWeight.Bold
        )
    }
}
```

Tiếp theo, hiện thực hàm xử lý sự kiện chung khi nhấn phím, Khai báo hiện thực hàm này dạng Lambda bên trong thân của **SimpleCalculatorScreen** (cùng với các biến trạng thái)

```
var nhanPhim: (String)->Unit = {
    when(it) {
        "C" -> { //reset lại TextField
            _expression=""
            _result="0"
        }
        "DEL" -> { //xoá ký tự cuối biểu thức
            if(_expression.length > 0){
                _expression = _expression.substring(_expression.length-1)
            }
        }
        "Ans" -> { //sao chép kết quả vào biểu thức
            _expression = _result
        }
        "=" -> { //tính giá trị biểu thức
            //sẽ hiện thực hàm tính toán và bổ sung sau
        }
        else -> { // các phím còn lại
            _expression+=it
        }
    }
}
```

➤ Xử lý tính toán biểu thức

Giá trị của TextField Biểu Thức tồn tại dạng chuỗi, cần có giải pháp chuyển (parse) chuỗi biểu thức này sang biểu thức tính toán, phân tích nó thành các toán hạng, toán tử, thứ tự ưu tiên của các phép tính rồi thực hiện duyệt và tính toán để cho kết quả cuối cùng.

Tuy nhiên, có nhiều cá nhân, tổ chức cung cấp thư viện để hỗ trợ tính toán giá trị khi đưa vào một biểu thức. Bài thực hành này sử dụng thư viện **exp4j 0.4.8** cho phần hiện thực tính giá trị biểu thức.

Thêm thư viện để biên dịch tại phần **dependencies** của file **build.gradle.kts** (Module:app)

```
implementation("net.objecthunter:exp4j:0.4.8")
```

Như hình sau:

```
dependencies { this: DependencyHandlerScope

    implementation("androidx.core:core-ktx:1.9.0")
    implementation("androidx.lifecycle:lifecycle-runtime-ktx:2.6.2")
    implementation("androidx.activity:activity-compose:1.8.0")
    implementation(platform("androidx.compose:compose-bom:2023.03.00"))
    implementation("androidx.compose.ui:ui")
    implementation("androidx.compose.ui:ui-graphics")
    implementation("androidx.compose.ui:ui-tooling-preview")
    implementation("androidx.compose.material3:material3")
    implementation("net.objecthunter:exp4j:0.4.8")
    testImplementation("junit:junit:4.13.2")
    androidTestImplementation("androidx.test.ext:junit:1.1.5")
    androidTestImplementation("androidx.test.espresso:espresso-core:3.5.1")
    androidTestImplementation(platform("androidx.compose:compose-bom:2023.03.00"))
    androidTestImplementation("androidx.compose.ui:ui-test-junit4")
    debugImplementation("androidx.compose.ui:ui-tooling")
    debugImplementation("androidx.compose.ui:ui-test-manifest")
}
```

Tại MainActivity.kt, thêm thư viện

```
import net.objecthunter.exp4j.ExpressionBuilder
```

Hiện thực hàm tính toán

```
fun calculate(exp:String):String{
    var s_exp = exp.replace("x", "*")
        .replace("%", "/100")
    val eval = ExpressionBuilder(s_exp).build()
    return eval.evaluate().toString()
}
```

Bổ sung trong hàm xử lý phím "="

```
"=" -> { //tính giá trị biểu thức
    //sẽ hiện thực hàm tính toán và bổ sung sau
    _result = calculate(_expression)
}
```

BÀI TẬP

Phần 1: Trắc Nghiệm

Phần 2: Viết Ứng Dụng

Tối ưu lại chương trình trên bằng cách bổ sung thêm phần kiểm tra khi người dùng nhập các phím sao cho đảm bảo biểu thức vẫn hợp lệ, nếu không hợp lệ thì giá trị nhập của phím đó không được thêm vào TextField của biểu thức

Ví dụ: nhập tính: $6.23 + 4..$ không hợp lệ do có 2 dấu $.$ khi đó TextField chỉ hiện $\rightarrow 6.23+4.$ là hợp lệ