บทที่ 7

การสร้าง UI และการจัดการโต้ตอบใน Android

อ.ดร.พงษ์ดนัย จิตตวิสุทธิกุล

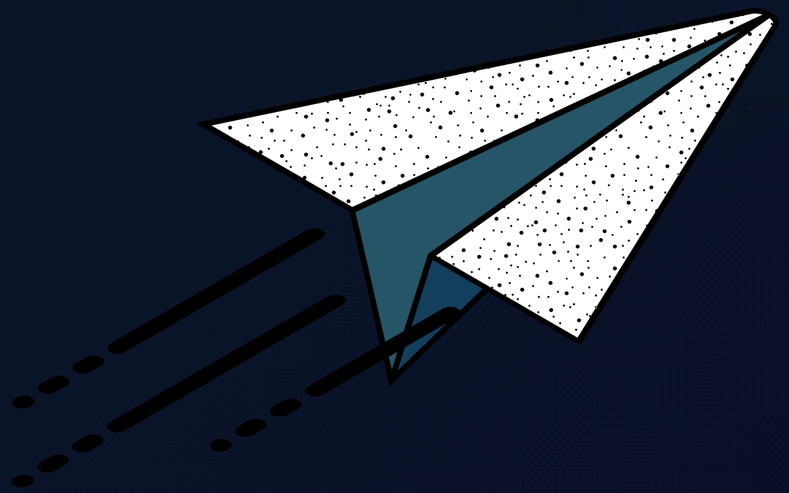
Topic

3 หัวข้อหลัก:

- การสร้าง UI ด้วย Jetpack Compose
- วงจรชีวิตของ Activity (Activity Lifecycle)
- ปฏิบัติการ: สร้างแอปคำนวณทิป

Objective behavior

3 วัตถุประสงค์เชิงพฤติกรรม:



ระบุหน้าที่ของ Composable function พื้นฐานและ Layout Composables ในการสร้าง UI ด้วย Jetpack Compose ได้

อธิบายแนวคิดของ State และบทบาทของ remember กับ mutableStateOf ในการสร้าง UI ที่โต้ตอบได้

ระบุลำดับการเรียกของ Callback Methods หลักใน Activity Lifecycle ตามสถานการณ์ต่าง ๆ ที่กำหนดได้

การสร้าง UI ด้วย Jetpack Compose

เปลี่ยนจากการแสดงข้อความธรรมดา
สู่การสร้าง "หน้าจอ" ที่สมบูรณ์แบบ
ด้วยการประกอบ "ตัวต่อเลโก้" ดิจิทัล

องค์ประกอบพื้นฐาน: Composable Functions

ใน Jetpack Compose ทุกสิ่งบนหน้าจอคือฟังก์ชัน Kotlin ที่ทำหน้าที่ "วาด" UI เปรียบเสมือนตัวต่อเลโก้แต่ละชิ้น:

1. Text (แสดงข้อความ)

```
Text(text = "Welcome!")
```

2. TextField (รับข้อมูล)

```
TextField(

        value = "",

        onValueChange = {}

       )
```

3. Button (ปุ่มกด)

```
Button(onClick = { /* Do action */ }) {
Text("Click Me")

        }
```

การจัดวางองค์ประกอบ & Modifiers

Layout Composables ("แผ่นรองเลโก้")

ใช้เพื่อจัดเรียง Composable ชิ้นอื่นๆ ให้อยู่ในลำดับชั้น (Hierarchy)

↕ Column: จัดเรียงใน แนวตั้ง (บนลงล่าง)

↔ Row: จัดเรียงใน แนวนอน (ซ้ายไปขวา)

Modifiers ("คำคุณศัพท์")

ใช้ปรับแต่งคุณสมบัติ เช่น ขนาด, สี, หรือระยะห่าง

```
Text( text = "Styled Text", modifier =
Modifier.padding(16.dp) )
```

วงจรชีวิตของ Activity (Activity Lifecycle)

ทำความเข้าใจ "บ้าน" ที่ UI ของเราอาศัยอยู่ หน้าจอแอปเกิดขึ้น ทำงาน และถูกทำลายไป อย่างไร?

ทำไมต้องเข้าใจ Activity Lifecycle?

ระบบปฏิบัติการจะเรียกใช้ Callback Methods เปลี่ยนสถานะหน้าจออัตโนมัติ ซึ่งสำคัญต่อแอปดังนี้:

จัดการทรัพยากร

รู้ว่าเมื่อใดควรหยุดการทำงานเบื้องหลัง (เช่น หยุดเล่นวิดีโอ) เมื่อแอปถูกซ่อน เพื่อช่วยประหยัดแบตเตอรี่และหน่วยความจำเครื่อง

ป้องกันข้อมูลสูญหาย

ช่วยให้เราทราบจังหวะที่ควรบันทึกข้อมูลชั่วคราว (เช่น ข้อความที่กำลังพิมพ์) ก่อนที่หน้าจอจะถูกทำลาย เช่น ตอนสลับแอปหรือหมุนจอ

สร้าง UX ที่ดี

ป้องกันไม่ให้แอปพลิเคชันแครช (Crash) หรือทำงานผิดปกติเมื่อเกิดเหตุการณ์แทรกซ้อน เช่น มีสายโทรศัพท์เข้าขณะใช้งาน

สถานะหลักของ Lifecycle

เราใช้ Log.d() เป็น "กล้องวงจรปิด" ดูการทำงาน:

- **เมื่อเปิดแอป (Created & Visible)**
  onCreate() → onStart() → onResume()
  แอปถูกสร้าง ปรากฏบนจอ และพร้อมรับการกดปุ่ม

- **เมื่อกดปุ่ม Home (Hidden)**
  onPause() → onStop()
  แอปสูญเสีย Focus และถูกซ่อนจากหน้าจอ (หยุดชั่วคราว)

สถานะหลักของ Lifecycle

เราใช้ Log.d() เป็น "กล้องวงจรปิด" ดูการทำงาน:

เมื่อกลับเข้าแอปอีกครั้ง (Return)
onStart() → onResume()
(ไม่เรียก onCreate ซ้ำ เพราะแอปยังไม่ถูกทำลาย)

เมื่อกดปุ่ม Back (Destroyed)
onPause() → onStop() → onDestroy()
หน้าจอถูกทำลายและลบออกจากหน่วยความจำ

ปฏิบัติการ: การติดตามวงจรชีวิตด้วย Logcat

"ติดกล้องวงจรปิด" ให้แอปพลิเคชัน เราจะใช้ Log.d() เพื่อสั่งให้แอป "ตะโกน" ข้อความออกมา และใช้ Logcat เป็น "ห้องควบคุม" คอยดักฟังและกรองด้วยคำว่า Lifecycle

```
override fun onStart() {
    super.onStart()
    Log.d("Lifecycle", "onStart Called")
}

override fun onPause() {
    super.onPause()
    Log.d("Lifecycle", "onPause Called")
}
// ทำเช่นเดียวกันกับ onResume, onStop, onDestroy
```

ผลลัพธ์จาก 4 สถานการณ์ (Scenarios)

A: เปิดแอปพลิเคชัน
> onCreate -> onStart -> onResume
แอปถูกสร้าง ปรากฏบนจอ และพร้อมรับการใช้งาน

B: กดปุ่ม Home
> onPause -> onStop
แอปไม่ได้อยู่หน้าจอแล้ว ระบบสั่งหยุดชั่วคราว

C: กลับเข้าแอปอีกครั้ง
> onStart -> onResume
สั่งให้แอปกลับมาทำงานต่อได้เลย

D: กดปุ่ม Back (ออกจากแอป)
> onPause -> onStop -> onDestroy
ผู้ใช้บอกว่าเลิกใช้งานแล้ว ระบบจึงสั่ง "ทำลาย" ทิ้งออกจากหน่วยความจำ

ปฏิบัติการสร้างแอปคำนวณทิป (Tip Calculator)

นำความรู้ทั้งหมดมาสร้างแอปพลิเคชันแรก
ที่มีการโต้ตอบกับผู้ใช้ (Interactive UI)

แนวคิดสำคัญ: "State" (สถานะและความจำของ UI)

UI คือกระดานไวท์บอร์ด!

Composable Function ทำหน้าที่วาด UI แต่ตัวมันเอง "ไม่มีความจำ" (ลบวาดใหม่ข้อมูลจะหาย)

State คือกล่อง "ความจำพิเศษ" ที่สั่งให้ UI จดจำข้อมูลไว้ เช่น สิ่งที่ผู้ใช้พิมพ์เข้ามาใน TextField

```
var amountInput by remember {
mutableStateOf("") }
```

เมื่อค่าใน State เปลี่ยนแปลง Compose จะวาดใหม่ (Recomposition) ทันที ทำให้ UI อัปเดต!

โครงสร้างการทำงานแอปคำนวณทิป

1.สร้าง State (กล่องความจำ):

สำหรับเก็บ amountInput (ยอดอาหาร), tipAmount (ยอดทิป), totalAmount (ยอดรวม)

2.รับข้อมูลผู้ใช้:

ใช้ อัปเดตค่า amountInput เมื่อมีการพิมพ์ (onValueChange)

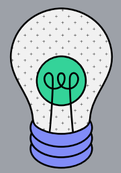
3.ประมวลผล (Logic):

เมื่อกด Button ให้แปลงข้อความกรอกเป็นตัวเลขด้วย toDoubleOrNull() แล้วคำนวณ 15%

4.แสดงผลลัพธ์:

นำค่าผลลัพธ์ไปแสดงใน Text และจัดรูปแบบทศนิยมด้วย String.format("%.2f")

Questions & Discussion



วันนี้เราได้เรียนรู้การประกอบ UI
ด้วย Compose, เข้าใจความสำคัญของ
Activity Lifecycle
และทำให้แอปมีชีวิตด้วย State