gis

Kotlin Jetpack Compose

บทนำและพื้นฐาน (Introduction and Fundamentals)

1. Jetpack Compose คืออะไร?

- Jetpack Compose เป็น Modern Toolkit ที่ Google พัฒนาขึ้นมาเพื่อการสร้าง UI บน Android โดยเฉพาะ มันถูกออกแบบมาให้ใช้งานง่าย รวดเร็ว และมีประสิทธิภาพสูงกว่าการสร้าง UI แบบเดิมที่ใช้ XML layout

บทนำและพื้นฐาน (Introduction and Fundamentals)

2. ทำไมถึงควรใช้ Jetpack Compose?

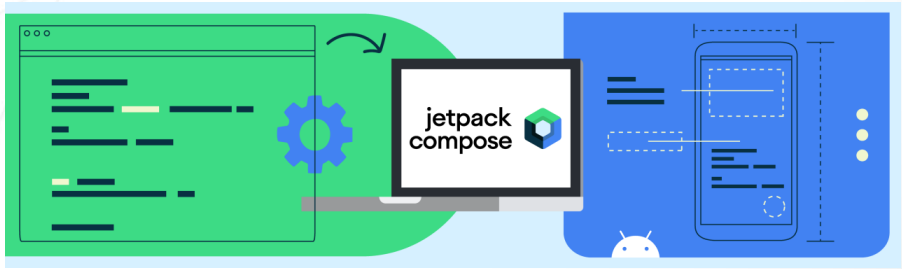
- เขียนโค้ดน้อยลง: Jetpack Compose ใช้ภาษา Kotlin ที่กระชับและเข้าใจง่าย ทำให้เขียนโค้ดได้น้อยลง และลดความซับซ้อนของโครงสร้างโปรเจกต์

- UI ที่สอดคล้องกัน: Jetpack Compose ช่วยให้ UI มีความสอดคล้องกันมากขึ้น เพราะ UI ถูกกำหนดด้วยโค้ด ทำให้หลีกเลี่ยงปัญหาที่เกิดจากการแก้ไข XML layout หลายๆ ไฟล์

- Preview แบบ Real-time: คุณสามารถเห็นผลลัพธ์ของ UI ที่กำลังพัฒนาได้ทันที ใน Android Studio โดยไม่ต้องคอมไพล์หรือรันแอปพลิเคชันใหม่ทุกครั้ง

- ประสิทธิภาพสูง: Jetpack Compose ถูกออกแบบมาให้มีประสิทธิภาพสูง โดยเฉพาะอย่างยิ่งเมื่อทำงานร่วมกับ UI ที่ซับซ้อน

- ความนิยมที่เพิ่มขึ้น: Jetpack Compose กำลังเป็นที่นิยมอย่างมากในหมู่นักพัฒนา Android และมีแนวโน้มว่าจะกลายเป็นมาตรฐานใหม่ในการพัฒนา UI บน Android



บทนำและพื้นฐาน (Introduction and Fundamentals)

3. Jetpack Compose VS XML Layout

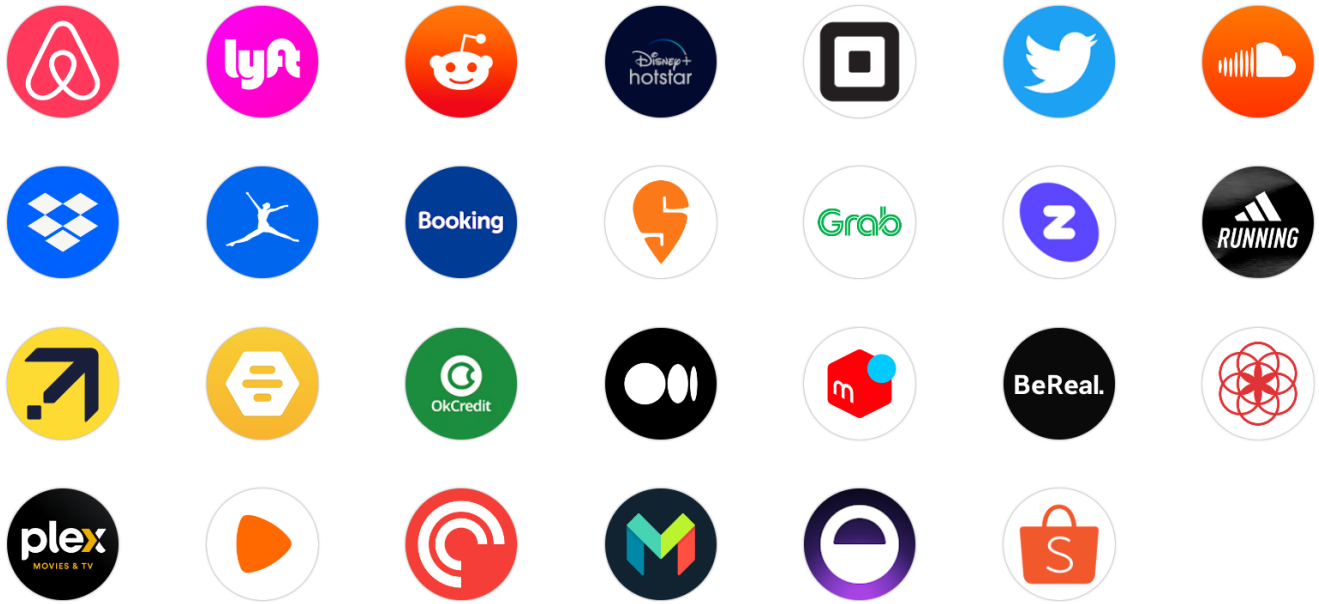
	XML Layout	Jetpack Compose
ปริมาณโค้ด	มาก	น้อยกว่า
ความยืดหยุ่น	น้อยกว่า	มากกว่า
การจัดการ State	ยากกว่า	ง่ายกว่า
ประสิทธิภาพ	ต่ำกว่า	สูงกว่า
การเรียนรู้	ง่ายสำหรับผู้เริ่มต้น	ต้องใช้เวลาทำความเข้าใจ
เหมาะสำหรับ	โครงสร้าง UI แบบคงที่, โปรเจกต์ที่มีอยู่แล้ว	UI ที่ซับซ้อน, โปรเจกต์ใหม่, ต้องการประสิทธิภาพสูง
Preview	ไม่มี	มี

บทนำและพื้นฐาน (Introduction and Fundamentals)

4. ความนิยมในปัจจุบัน

- ความนิยมของ Jetpack Compose ในปัจจุบันกำลังเพิ่มขึ้นอย่างต่อเนื่อง แม้ว่าจะยังเป็นเทคโนโลยีที่ค่อนข้างใหม่เมื่อเทียบกับ XML Layout แต่ก็ได้รับความสนใจและการยอมรับจากนักพัฒนา Android มากขึ้นอย่างต่อเนื่อง

ตัวอย่าง แอปฯ ที่พัฒนาด้วย Jetpack Compose :



บทนำและพื้นฐาน (Introduction and Fundamentals)

5. พื้นฐาน Kotlin

5.1 Variables (ตัวแปร)

- var (Mutable Variable): ใช้เมื่อต้องการประกาศตัวแปรที่สามารถเปลี่ยนแปลงค่าได้ในภายหลัง
- val (Immutable Variable): ใช้เมื่อต้องการประกาศตัวแปรที่ไม่สามารถเปลี่ยนแปลงค่าได้ (คล้ายกับ final ใน Java)

```
var greeting: String = "Hello" // สามารถเปลี่ยนแปลงค่าได้ เช่น greeting = "Hi"
val pi: Double = 3.14159 // ไม่สามารถเปลี่ยนแปลงค่าได้
```

บทนำและพื้นฐาน (Introduction and Fundamentals)

5. พื้นฐาน Kotlin

5.2 Lists (ลิสต์)

- listOf: สร้าง immutable list (ไม่สามารถเปลี่ยนแปลงค่าได้)
- mutableListOf: สร้าง mutable list (สามารถเปลี่ยนแปลงค่าได้)

```kotlin
val numbers = listOf(1, 2, 3, 4, 5)
val mutableNumbers = mutableListOf(1, 2, 3)
mutableNumbers.add(4) // เพิ่ม element เข้าไปใน list
```

บทนำและพื้นฐาน (Introduction and Fundamentals)

5. พื้นฐาน Kotlin

5.3 Functions (ฟังก์ชัน)

- Named Function: ฟังก์ชันที่มีชื่อเรียก สามารถเรียกใช้ได้จากที่ต่างๆ ในโค้ด
- Lambda Expression: ฟังก์ชันที่ไม่มีชื่อ นิยมใช้ใน Jetpack Compose เพื่อกำหนด UI

```kotlin
// Named Function
fun greet(name: String): String {
    return "Hello, $name!"
}

// Lambda Expression
val greet: (String) -> String = { name -> "Hello, $name!" }
```

บทนำและพื้นฐาน (Introduction and Fundamentals)

5. พื้นฐาน Kotlin

5.4 Classes (คลาส)

- Data Class: คลาสที่ใช้เก็บข้อมูล โดย Kotlin จะสร้างฟังก์ชัน equals(), hashCode(), toString() และ copy() ให้โดยอัตโนมัติ
- Class: คลาสทั่วไปที่สามารถมี properties และ functions ได้

```kotlin
// Data Class
data class User(val name: String, val age: Int)

// Class
class Car(val brand: String, val model: String) {
    fun startEngine() {
        println("Engine started!")
    }
}
```

บทนำและพื้นฐาน (Introduction and Fundamentals)

5. พื้นฐาน Kotlin

5.5 Conditionals (เงื่อนไข)

- if-else: ตรวจสอบเงื่อนไข ถ้าเป็นจริงจะทำงานในบล็อก if ถ้าเป็นเท็จจะทำงานในบล็อก else

- when: ตรวจสอบค่าของตัวแปรหรือ expression แล้วทำงานใน case ที่ตรงกัน

```kotlin
// if-else
val number = 10
if (number > 0) {
    println("Positive")
} else if (number < 0) {
    println("Negative")
} else {
    println("Zero")
}

// when
val dayOfWeek = "Monday"
when (dayOfWeek) {
    "Monday" -> println("Start of the week")
    "Friday" -> println("End of the week")
    else -> println("Middle of the week")
}
```

บทนำและพื้นฐาน (Introduction and Fundamentals)

5. พื้นฐาน Kotlin

5.6 Loops (การวนซ้ำ)

- for loop: วนซ้ำตามจำนวนครั้งที่กำหนด หรือวนซ้ำใน collection
- while loop: วนซ้ำจนกว่าเงื่อนไขจะเป็นเท็จ

```kotlin
// for loop
for (i in 1..5) {
    println(i)
}

val fruits = listOf("apple", "banana", "orange")
for (fruit in fruits) {
    println(fruit)
}

// while loop
var count = 0
while (count < 5) {
    println(count)
    count++
}
```

บทนำและพื้นฐาน (Introduction and Fundamentals)

5. พื้นฐาน Kotlin

5.7 Lambdas (แลมบ์ดา)

- Higher-Order Function: ฟังก์ชันที่รับฟังก์ชันอื่นเป็น argument หรือ return ฟังก์ชัน
- Lambda Expression: นิพจน์ที่แทนฟังก์ชัน นิยมใช้ใน Jetpack Compose

```kotlin
// Higher-Order Function
fun calculate(x: Int, y: Int, operation: (Int, Int) -> Int): Int {
    return operation(x, y)
}

val sum = calculate(5, 3) { a, b -> a + b } // ใช้ lambda expression เป็น argument
val product = calculate(5, 3) { a, b -> a * b }
```

บทนำและพื้นฐาน (Introduction and Fundamentals)

6. Declarative UI

6.1 Declarative UI คืออะไร?

- Declarative UI (Jetpack Compose) เป็นแนวคิดใหม่ในการสร้าง UI ที่ช่วยลดความซับซ้อนของโค้ดและเพิ่มความยืดหยุ่นในการพัฒนา UI เมื่อเทียบกับ Imperative UI (XML layout) แม้ว่า Jetpack Compose จะยังเป็นเทคโนโลยีที่ค่อนข้างใหม่ แต่ก็มีแนวโน้มที่จะกลายเป็นมาตรฐานใหม่ในการพัฒนา UI บน Android ในอนาคต

บทนำและพื้นฐาน (Introduction and Fundamentals)

6. Declarative UI

6.2ตัวอย่างโค้ดเปรียบเทียบ Jetpack Compose กับ XML layout

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"

    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:layout_width="wrap_content"

        android:layout_height="wrap_content"
        android:text="Hello, World!" />

</LinearLayout>
```

XML Layout

```kotlin
@Composable
fun Greeting() {
    Column {
        Text(text = "Hello, World!")
    }
}
```

Jetpack Compose

บทนำและพื้นฐาน (Introduction and Fundamentals)

7. Composable Function

7.1 **Composable Function** คืออะไร?

Composable Functions เป็นฟังก์ชันพิเศษใน Jetpack Compose ที่ใช้สำหรับสร้างและจัดการส่วนประกอบต่างๆ ของ UI (User Interface) โดยมีคุณสมบัติสำคัญดังนี้

- Annotate ด้วย @Composable: ฟังก์ชันเหล่านี้จะต้องถูกทำเครื่องหมายด้วย annotation @Composable เพื่อบอกให้คอมไพเลอร์ทราบว่าฟังก์ชันนี้ใช้สำหรับสร้าง UI
- คืนค่าเป็น Unit: Composable Functions ไม่คืนค่าใดๆ (คืนค่าเป็น Unit) เพราะหน้าที่หลักของมันคือการอธิบายว่า UI ควรมีหน้าตาอย่างไร
- สามารถเรียกใช้ Composable Functions อื่นๆ: Composable Functions สามารถเรียกใช้ Composable Functions อื่นๆ ได้ ทำให้สามารถสร้าง UI ที่ซับซ้อนได้ง่าย

บทนำและพื้นฐาน (Introduction and Fundamentals)

7. Composable Function

7.2 **State, Recomposition, Side-effects** คืออะไร

- State: State คือข้อมูลที่ใช้ในการแสดงผล UI เมื่อ State เปลี่ยนแปลง UI ก็จะถูกอัปเดตตามไปด้วย
- Recomposition: Recomposition คือกระบวนการที่ Jetpack Compose ใช้ในการอัปเดต UI เมื่อ State เปลี่ยนแปลง โดยจะทำการเรียกใช้ Composable Functions ที่เกี่ยวข้องกับ State นั้นซ้ำอีกครั้ง
- Side-effects: Side-effects คือผลข้างเคียงที่เกิดขึ้นจากการทำงานของ Composable Functions เช่น การเรียกใช้ API หรือการเปลี่ยนแปลงข้อมูลภายนอก

บทนำและพื้นฐาน (Introduction and Fundamentals)

7. Composable Function

7.2 **State, Recomposition, Side-effects** คืออะไร

- จากตัวอย่างโค้ดนี้ count คือ State ที่ใช้ในการแสดงผล UI เมื่อกดปุ่ม "Increment" ค่า count จะเพิ่มขึ้น 1 ทำให้เกิด Recomposition และ UI จะถูกอัปเดตเพื่อแสดงค่า count ใหม่

```
@Composable
fun Counter() {
    var count by remember { mutableStateOf(0) } // State (ข้อมูล)

    Column {
        Text(text = "Count: $count") // UI ที่แสดงผลตาม State

        Button(onClick = { count++ }) { // เมื่อกดปุ่ม State จะเปลี่ยนแปลง
            Text(text = "Increment")
        }
    }
}
```

บทนำและพื้นฐาน (Introduction and Fundamentals)

Todo App

ลิงค์ Notion : https://www.notion.so/Jetpack-Compose-26533bc1a91c478d8a6ba05a853b8b62?pvs=4

การจัดการข้อมูล (State Management)

State Management เป็นส่วนสำคัญในการสร้างแอปพลิเคชันที่ตอบสนองต่อการเปลี่ยนแปลงข้อมูลและมี UI ที่อัปเดตอยู่เสมอ ใน Jetpack Compose การจัดการ State ทำได้อย่างมีประสิทธิภาพ

1. **State และ State HoistingState**: คือข้อมูลที่ UI ใช้ในการแสดงผล เช่น ข้อความ, สี, หรือสถานะของปุ่ม

- State: คือข้อมูลที่ UI ใช้ในการแสดงผล เช่น ข้อความที่แสดงในปุ่ม, จำนวนครั้งที่ผู้ใช้กดปุ่ม หรือค่าที่ผู้ใช้กรอกในช่อง TextField เมื่อ State เปลี่ยนแปลง Jetpack Compose จะทำการ recompose (วาดใหม่) เฉพาะส่วนของ UI ที่เกี่ยวข้องกับ State นั้น ทำให้ UI มีประสิทธิภาพและตอบสนองได้ดี
- State Hoisting: คือเทคนิคการย้าย State ขึ้นไปอยู่ใน Composable Function ที่ระดับสูงกว่า เพื่อให้สามารถแชร์ State ระหว่าง Composable หลายๆ ตัวได้ เช่น ถ้ามีหลายปุ่มที่ต้องการอัปเดตข้อความเดียวกัน เราสามารถสร้าง State ใน Composable ที่อยู่เหนือปุ่มทั้งหมด แล้วส่ง State นั้นไปยังปุ่มต่างๆ เพื่อให้ปุ่มแต่ละปุ่มสามารถอ่านและอัปเดต State ได้

การจัดการข้อมูล (State Management)

2. การใช้ State และ MutableState ใน Compose

- remember: เป็นฟังก์ชันที่ใช้ในการสร้าง State ที่จะคงอยู่ตราบใดที่ Composable Function นั้นยังอยู่ใน Composition (ยังถูกแสดงผลอยู่) ถ้า Composable Function ถูกทำลาย State ที่สร้างด้วย remember ก็จะหายไปด้วย
- mutableStateOf: ใช้ในการสร้าง State ที่สามารถเปลี่ยนแปลงได้ โดยจะคืนค่า MutableState object ซึ่งมี property value ที่เราสามารถเปลี่ยนแปลงได้ เมื่อ value เปลี่ยนแปลง Jetpack Compose จะรู้และทำการ recompose UI ที่เกี่ยวข้อง

การจัดการข้อมูล (State Management)

3. การจัดการ State ระหว่าง Components

- การส่งผ่าน State ระหว่าง Composable Functions:

เราสามารถส่ง State และฟังก์ชันสำหรับอัปเดต State ผ่านพารามิเตอร์ของ Composable Function ได้ เช่น ส่ง State count และฟังก์ชัน onCountChange ไปยัง Composable อื่น

- การใช้งาน ViewModel กับ Compose:

ViewModel เหมาะสำหรับเก็บ State ที่ซับซ้อนหรือ State ที่ต้องคงอยู่แม้ว่า UI จะถูกทำลายและสร้างใหม่ (เช่น เมื่อหมุนหน้าจอ) เราสามารถใช้ viewModel() function เพื่อสร้างหรือดึง ViewModel มาใช้งานใน Composable Function และใช้ collectAsState() หรือ observeAsState() เพื่อสังเกตการเปลี่ยนแปลงของ State ใน ViewModel

การเปลี่ยนหน้าจอ (Navigation)

การเปลี่ยนหน้าจอ (Navigation) เป็นองค์ประกอบสำคัญในการสร้างแอปพลิเคชันที่มีหลายหน้าจอ Jetpack Compose มี Navigation Component ที่ช่วยให้การจัดการการนำทางระหว่างหน้าจอเป็นเรื่องง่ายและมีประสิทธิภาพ

1. Navigation Component

Navigation Component เป็นไลบรารีที่ช่วยจัดการการนำทางในแอปพลิเคชัน Android โดยมีองค์ประกอบหลัก 3 ส่วน:

- NavHost: คอนเทนเนอร์ที่แสดงผล Composable Destinations (หน้าจอต่างๆ) ตามเส้นทาง (route) ที่กำหนดใน NavGraph. NavHost จะคอยดูแล stack ของหน้าจอ และแสดงผลหน้าจอที่อยู่บนสุดของ stack เสมอ
- NavController: ทำหน้าที่ควบคุมการนำทางระหว่างหน้าจอต่างๆ โดยมีฟังก์ชันหลักๆ ดังนี้:
  - navigate(route): นำทางไปยังหน้าจอใหม่ตาม route ที่กำหนด
  - popBackStack(): ย้อนกลับไปยังหน้าจอก่อนหน้า
  - navigateUp(): นำทางกลับตามลำดับชั้นของ NavGraph
- Composable Destinations: หน้าจอต่างๆ ในแอปพลิเคชันที่ประกาศด้วย composable() ภายใน NavHost

การเปลี่ยนหน้าจอ (Navigation)

2. การประกาศ NavGraph

NavGraph เป็นไฟล์ XML ที่กำหนดเส้นทางการนำทางระหว่างหน้าจอต่างๆ ในแอปพลิเคชัน โดยแต่ละหน้าจอจะถูกกำหนดด้วย route ที่ไม่ซ้ำกัน ตัวอย่างดังรูป

```
<navigation xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/nav_graph"
    app:startDestination="@id/homeFragment">

    <fragment
        android:id="@+id/homeFragment"
        android:name="com.example.myapp.HomeFragment"
        android:label="Home" />

    <fragment
        android:id="@+id/detailsFragment"
        android:name="com.example.myapp.DetailsFragment"
        android:label="Details">
        <argument
            android:name="itemId"
            app:argType="string" />
    </fragment>
</navigation>
```

การเปลี่ยนหน้าจอ (Navigation)

3 . การส่งข้อมูลระหว่างหน้าจอ (Arguments)

เราสามารถส่งข้อมูลระหว่างหน้าจอได้โดยการเพิ่ม arguments ใน route ของ destination และใช้ navController.navigate() เพื่อเปลี่ยนหน้าจอพร้อมส่งข้อมูลไปด้วย ตัวอย่างดังรูป

```
// ส่งค่า itemId ไปยังหน้าจอ Details
navController.navigate("details/$itemId")

// รับค่า itemId ในหน้าจอ Details
composable("details/{itemId}") { backStackEntry ->
    val itemId = backStackEntry.arguments?.getString("itemId")
}
```

การเปลี่ยนหน้าจอ (Navigation)

4. Nested Navigation

Nested Navigation คือการนำทางแบบซ้อนหลายชั้น โดยแต่ละ NavGraph สามารถมี NavGraph ย่อยได้ ทำให้สามารถจัดกลุ่มหน้าจอที่เกี่ยวข้องกันได้อย่างเป็นระเบียบ และช่วยให้การจัดการการนำทางในแอปพลิเคชันขนาดใหญ่ทำได้ง่ายขึ้น

สร้าง UI ที่รองรับทุกอุปกรณ์ (Responsive UI)

Responsive UI เป็นแนวคิดการออกแบบ UI ที่สามารถปรับตัวเข้ากับขนาดหน้าจอและอุปกรณ์ที่หลากหลายได้อย่างมีประสิทธิภาพ Jetpack Compose มีเครื่องมือและแนวคิดที่ช่วยให้เราสร้าง Responsive UI ได้อย่างง่ายดาย

1. Modifiers ที่ใช้ในการปรับขนาด UI

Modifiers เป็นเครื่องมือสำคัญในการสร้าง Responsive UI ใน Jetpack Compose โดยมี Modifiers ที่เกี่ยวข้องกับการปรับขนาด UI ดังนี้:

- fillMaxWidth(), fillMaxHeight(), fillMaxSize(): ขยาย UI element ให้เต็มความกว้าง, ความสูง, หรือทั้งความกว้างและความสูงของพื้นที่ว่าง
- width(), height(), size(): กำหนดขนาดคงที่ของ UI element
- weight(): กำหนดสัดส่วนของ UI element ใน layout แบบ Row หรือ Column
- padding(): เพิ่มระยะห่างรอบๆ UI element

สร้าง UI ที่รองรับทุกอุปกรณ์ (Responsive UI)

2. ConstraintLayout

ConstraintLayout เป็น layout ที่ทรงพลังใน Jetpack Compose ช่วยให้คุณจัดวางองค์ประกอบ UI (Composables) ได้อย่างอิสระและยืดหยุ่น โดยกำหนดความสัมพันธ์ (Constraints) ระหว่างองค์ประกอบต่างๆ เช่น การวางชิดขอบ, การจัดกึ่งกลาง, การวางห่างจากองค์ประกอบอื่นๆ เป็นต้น

```
ConstraintLayout {
    val (button, text) = createRefs()

    Button(
        onClick = { /* ... */ },
        modifier = Modifier.constrainAs(button) {
            top.linkTo(parent.top)
            start.linkTo(parent.start)
        }
    ) {
        Text("Click me")
    }

    Text(
        text = "Hello World!",
        modifier = Modifier.constrainAs(text) {
            top.linkTo(button.bottom)
            start.linkTo(parent.start)
        }
    )
}
```

สร้าง UI ที่รองรับทุกอุปกรณ์ (Responsive UI)

3. Window Insets

Window Insets คือพื้นที่รอบๆ หน้าจอของอุปกรณ์ Android ที่ถูกใช้โดยส่วนประกอบของระบบ เช่น Status bar, Navigation bar, IME (Input Method Editor หรือ คีย์บอร์ด), Cutout (รอยบากบนหน้าจอ) เป็นต้น Window Insets จะบอกให้แอปพลิเคชันรู้ว่าพื้นที่ส่วนไหนของหน้าจอที่ไม่ควรวาง UI elements เพื่อไม่ให้ถูกบังหรือทับซ้อน

สร้าง UI ที่รองรับทุกอุปกรณ์ (Responsive UI)

4. WindowSizeClass

WindowSizeClass เป็นส่วนหนึ่งของ Jetpack Compose ที่ช่วยให้เราสามารถปรับแต่ง UI ให้เหมาะสมกับขนาดหน้าจอของอุปกรณ์ต่างๆ ได้อย่างง่ายดาย โดย WindowSizeClass จะแบ่งขนาดหน้าจอออกเป็นกลุ่มๆ ตามความกว้าง (Width) และความสูง (Height) ทำให้เราสามารถเขียนโค้ด UI ที่ตอบสนองต่อขนาดหน้าจอที่แตกต่างกันได้

```
val windowSizeClass = calculateWindowSizeClass()

when (windowSizeClass.widthSizeClass) {
    WindowWidthSizeClass.Compact -> { /* UI for small screens */ }
    WindowWidthSizeClass.Medium -> { /* UI for medium screens */ }
    WindowWidthSizeClass.Expanded -> { /* UI for large screens */ }
}
```

สร้าง UI ที่รองรับทุกอุปกรณ์ (Responsive UI)

สรุป

การสร้าง Responsive UI ใน Jetpack Compose ต้องอาศัยการใช้ Modifiers, ConstraintLayout, Material Design, Window Insets, และ WindowSizeClass ร่วมกัน เพื่อให้ UI สามารถปรับตัวเข้ากับขนาดหน้าจอและอุปกรณ์ที่หลากหลายได้อย่างมีประสิทธิภาพ

Material Design

Material Design คือแนวทางการออกแบบ UI (User Interface) ที่ Google พัฒนาขึ้น เพื่อให้แอปพลิเคชันมีหน้าตาที่สวยงาม ทันสมัย สอดคล้องกัน และใช้งานง่ายบนทุกอุปกรณ์ Jetpack Compose รองรับ Material Design อย่างเต็มที่ ทำให้เราสามารถสร้าง UI ที่สวยงามและมีประสิทธิภาพ

1. Material Design คืออะไร?

Material Design เป็นเหมือนคู่มือการออกแบบที่เน้น:

- ความเรียบง่าย: ใช้พื้นที่ว่าง (whitespace) อย่างเหมาะสม และหลีกเลี่ยงองค์ประกอบที่ไม่จำเป็น
- ความชัดเจน: ใช้ Typography ที่อ่านง่าย และสีสันที่ตัดกันอย่างชัดเจน
- การเคลื่อนไหวที่มีความหมาย: ใช้ Animation เพื่อเน้นการกระทำของผู้ใช้ และทำให้ UI มีชีวิตชีวามากขึ้น
- ความสอดคล้อง: ใช้ Component และรูปแบบที่สอดคล้องกันทั้งแอปพลิเคชัน

Material Design

2. Material Theme ใน Jetpack Compose

Material Theme เป็นเหมือนชุดรูปแบบ (Theme) ที่กำหนดสี รูปแบบตัวอักษร และรูปทรงต่างๆ ของ UI ใน Jetpack Compose เราสามารถใช้ MaterialTheme Composable เพื่อครอบ UI ทั้งหมดของแอปพลิเคชัน และกำหนดธีมให้กับแอปพลิเคชันได้ง่ายๆ และเราสามารถปรับแต่ง Material Theme ได้ตามต้องการ เช่น เปลี่ยนสีหลัก (primary), สีรอง (secondary), รูปแบบตัวอักษร (typography), หรือรูปทรงขององค์ประกอบ UI (shapes)

```
MaterialTheme { // ครอบ UI ทั้งหมดด้วย MaterialTheme
    // ... UI elements ของแอปพลิเคชัน
}
```

Material Design

3. คอมโพเนนต์หลักของ Material Design

Jetpack Compose มี Material Components ที่พร้อมใช้งานมากมาย เช่น:

- Scaffold: โครงสร้างพื้นฐานของหน้าจอแอปพลิเคชัน ประกอบด้วย AppBar, BottomAppBar, FAB (Floating Action Button), Drawer, Snackbar, etc.
- AppBar: แถบด้านบนของหน้าจอ มักใช้แสดงชื่อแอปพลิเคชัน ปุ่มนำทาง หรือ actions อื่นๆ
- BottomNavigation: แถบนำทางด้านล่างของหน้าจอ
- Button: ปุ่มสำหรับให้ผู้ใช้กด มีหลายประเภท เช่น TextButton, OutlinedButton, ElevatedButton
- Card: พื้นที่สำหรับแสดงข้อมูลที่เกี่ยวข้องกัน มักใช้แสดงรูปภาพและข้อความ
- Text: องค์ประกอบสำหรับแสดงข้อความ
- TextField: ช่องสำหรับให้ผู้ใช้กรอกข้อมูล
- AlertDialog: กล่องโต้ตอบสำหรับแจ้งเตือนหรือยืนยันการกระทำ
- และอื่นๆ อีกมากมาย: เช่น Switch, Checkbox, RadioButton, Slider, ProgressIndicator, etc.

การทำงานกับข้อมูลภายนอก

ในแอปพลิเคชัน Android ส่วนใหญ่ มักจะต้องทำงานกับข้อมูลจากแหล่งภายนอก เช่น API, ฐานข้อมูล หรือไฟล์ Jetpack Compose มีเครื่องมือและแนวคิดที่ช่วยให้เราจัดการข้อมูลภายนอกได้อย่างมีประสิทธิภาพและสอดคล้องกับหลักการของ Jetpack Compose

1. การดึงข้อมูลจาก API

การดึงข้อมูลจาก API (Application Programming Interface) เป็นเรื่องที่พบเจอบ่อยในการพัฒนาแอปพลิเคชัน Android โดยทั่วไปเราจะใช้ไลบรารีอย่าง Retrofit, Volley หรือ Ktor เพื่อจัดการการสื่อสารกับ API

```
interface ApiService {
    @GET("users")
    suspend fun getUsers(): List<User>
}

@Composable
fun MyScreen(viewModel: MyViewModel = viewModel()) {
    val users by viewModel.users.collectAsState()

    LazyColumn {
        items(users) { user ->
            Text(text = user.name)
        }
    }
}
```

การทำงานกับข้อมูลภายนอก

2. การใช้ Repository Pattern

Repository Pattern เป็นรูปแบบการออกแบบซอฟต์แวร์ (Design Pattern) ที่ช่วยให้เราแยกส่วนการจัดการข้อมูลออกจาก UI (User Interface) โดย Repository ทำหน้าที่เป็นตัวกลางในการดึงข้อมูลจากแหล่งต่างๆ เช่น ฐานข้อมูล, API, หรือไฟล์ แล้วส่งต่อข้อมูลที่ประมวลผลแล้วให้กับ ViewModel หรือส่วนอื่นๆ ที่ต้องการใช้งาน

การทำงานกับข้อมูลภายนอก

3. การจัดเก็บข้อมูลใน Local Database

Local Database เป็นส่วนสำคัญในการพัฒนาแอปพลิเคชัน Android ที่ต้องการจัดเก็บข้อมูลไว้ในเครื่องของผู้ใช้ Room Persistence Library เป็นไลบรารีจาก Jetpack ที่ช่วยให้การทำงานกับฐานข้อมูล SQLite เป็นเรื่องง่ายและมีประสิทธิภาพ

องค์ประกอบหลักของ Room

1.Entity: เป็นคลาสที่แทนตารางในฐานข้อมูล แต่ละ property ในคลาสจะถูกแมปกับคอลัมน์ในตาราง

2.DAO (Data Access Object): เป็น interface ที่กำหนดเมธอดสำหรับเข้าถึงข้อมูลในฐานข้อมูล เช่น การเพิ่ม ลบ แก้ไข หรือค้นหาข้อมูล

3.Database: เป็นคลาส abstract ที่สืบทอดมาจาก RoomDatabase ทำหน้าที่เป็นจุดเข้าถึงฐานข้อมูล และต้องมี abstract method ที่คืนค่า DAO

การทำงานกับข้อมูลภายนอก

4. การจัดการ State ที่มาจากข้อมูลภายนอก

เมื่อดึงข้อมูลจาก API หรือฐานข้อมูล เราต้องจัดการ State ที่มาจากข้อมูลภายนอกเหล่านี้ใน Jetpack Compose โดยทั่วไปเราจะใช้ StateFlow หรือ LiveData เพื่อเก็บ State และ collectAsState หรือ observeAsState เพื่อ observe การเปลี่ยนแปลงของ State

หลักการประกาศตัวแปรใน Jetpack Compose

Jetpack Compose เน้นการใช้ immutable state (สถานะที่ไม่สามารถเปลี่ยนแปลงได้) เพื่อให้ UI อัปเดตเมื่อ state เปลี่ยนแปลง ซึ่งแตกต่างจากการเขียน UI แบบเดิมที่ใช้ XML ซึ่งมักจะใช้ mutable state (สถานะที่เปลี่ยนแปลงได้)

ArcGIS Maps

การ Setup Project ใน Android Studio

การ Setup Project เพื่อเตรียมพร้อมสำหรับการพัฒนาแอปพลิเคชัน แผนที่ ด้วย ArcGIS Maps SDK for Kotlin

1 .สร้าง Project ใหม่

1.1. เปิด Android Studio และสร้าง Empty Compose Activity project ใหม่

1.2. ตั้งชื่อโครงการ (เช่น "Display a map")

1.3. กำหนด Package name (เช่น "com.example.app" หรือตามชื่อองค์กร)

1.4. เลือกตำแหน่งบันทึกโครงการ (ควรเป็นโฟลเดอร์ใหม่)

1.5. เลือก Minimum SDK เป็น API 26 ("Oreo"; Android 8.0)

1.6. เลือก Build configuration language เป็น Kotlin DSL (build.gradle.kts)

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/display-a-map/

การ Setup Project ใน Android Studio

2 . แก้ไขไฟล์ Gradle

2.1. เปิดไฟล์ build.gradle.kts (Project: Display_a_map) และแทนที่เนื้อหาด้วยโค้ดที่กำหนดไว้ในรูปด้านล่าง ซึ่งจะกำหนดเวอร์ชันของปลั๊กอินที่ใช้ในโครงการ

build.gradle.kts (Project: Display_a_map)

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.
plugins {
    id("com.android.application") version "8.2.0" apply false
    id("org.jetbrains.kotlin.android") version "1.9.10" apply false
}
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/display-a-map/

การ Setup Project ใน Android Studio

2 . แก้ไขไฟล์ Gradle

2.2. เปิดไฟล์ build.gradle.kts (Module: app) และแทนที่เนื้อหาด้วยโค้ดที่กำหนดไว้ดังรูปในสไลด์ถัดไป ซึ่งจะ

- กำหนด Kotlin JVM target version
- ยกเว้นไฟล์บางไฟล์ในการ packaging
- กำหนด namespace ของแอปพลิเคชัน
- เพิ่ม dependencies สำหรับ ArcGIS Maps SDK for Kotlin, Toolkit และ Jetpack Compose

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/display-a-map/

การ Setup Project ใน Android Studio

build.gradle.kts (Module: app)

Expand

```
    kotlinOptions {
        jvmTarget = "17"
    }

    packaging {
        resources {
            excludes += "/META-INF/DEPENDENCIES"
        }
    }

    namespace = "com.example.app"
}

dependencies {
    implementation("androidx.core:core-ktx:1.12.0")
    implementation("androidx.lifecycle:lifecycle-runtime-ktx:2.7.0")
    implementation("androidx.activity:activity-compose:1.8.2")
    // Jetpack Compose Bill of Materials
    implementation(platform("androidx.compose:compose-bom:2023.10.01"))
    // Jetpack Compose dependencies
    implementation("androidx.compose.ui:ui")
    implementation("androidx.compose.material3:material3")
    // ArcGIS Map Kotlin SDK dependencies
    implementation("com.esri:arcgis-maps-kotlin:200.4.0")
    // Toolkit dependencies
    implementation(platform("com.esri:arcgis-maps-kotlin-toolkit-bom:200.4.0"))
    implementation("com.esri:arcgis-maps-kotlin-toolkit-geoview-compose")
}
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/display-a-map/

การ Setup Project ใน Android Studio

2 . แก้ไขไฟล์ Gradle

2.3. เปิดไฟล์ settings.gradle.kts และแทนที่เนื้อหาด้วยโค้ดดังรูป ซึ่งจะกำหนด repositories สำหรับ Gradle

settings.gradle.kts (Display a map)

```
pluginManagement {
    repositories {
        google()
        mavenCentral()
        gradlePluginPortal()
    }
}

dependencyResolutionManagement {
    @Suppress("UnstableApiUsage")
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    @Suppress("UnstableApiUsage")
    repositories {
        google()
        mavenCentral()
        maven { url = uri("https://esri.jfrog.io/artifactory/arcgis") }
    }
}

rootProject.name = "Display a map"
include(":app")
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/display-a-map/

การ Setup Project ใน Android Studio

3 . ซิงค์ Gradle

- คลิกที่ปุ่ม "Sync Now" หรือไอคอน refresh เพื่อซิงค์การเปลี่ยนแปลงใน Gradle ซึ่งอาจใช้เวลาสักครู่

4 . อนุญาตการเข้าถึงอินเทอร์เน็ต

- เปิดไฟล์ AndroidManifest.xml และเพิ่ม code ตามบรรทัดที่ 6 ดังรูปด้านขวา เพื่อให้แอปพลิเคชันสามารถเข้าถึงอินเทอร์เน็ตได้ ซึ่งจำเป็นสำหรับการดึงข้อมูลแผนที่จาก ArcGIS

AndroidManifest.xml

Expand

```
3  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:tools="http://schemas.android.com/tools">
5
6      <uses-permission android:name="android.permission.INTERNET" />
7
8      <application
9          android:allowBackup="true"
10         android:dataExtractionRules="@xml/data_extraction_rules"
11         android:fullBackupContent="@xml/backup_rules"
12         android:icon="@mipmap/ic_launcher"
13         android:label="@string/app_name"
14         android:roundIcon="@mipmap/ic_launcher_round"
15         android:supportsRtl="true"
16         android:theme="@style/Theme.DisplayAMap"
17         tools:targetApi="31">
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/display-a-map/

การสร้างแผนที่

การสร้างออบเจ็กต์แผนที่ (ArcGISMap) เพื่อเตรียมพร้อมสำหรับการแสดงผลบน UI ของแอปพลิเคชัน Android

1.สร้างแพ็กเกจและไฟล์สำหรับ UI

1.1.สร้างแพ็กเกจใหม่ชื่อ com.example.app.screens เพื่อจัดเก็บไฟล์ที่เกี่ยวข้องกับส่วนต่อประสานผู้ใช้ (UI) ของแอปพลิเคชัน

1.2. สร้างไฟล์ Kotlin ใหม่ชื่อ MainScreen.kt ภายในแพ็กเกจ screens ซึ่งจะเป็นไฟล์หลักสำหรับสร้างและจัดการ UI ของแผนที่

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/display-a-map/

การสร้างแผนที่

2. เตรียมไฟล์ MainScreen.kt

2.1.เปิดไฟล์ MainScreen.kt และลบโค้ดเริ่มต้นที่ Android Studio สร้างให้

2.2. เพิ่มส่วน import เพื่อนำเข้าคลาสและฟังก์ชันที่จำเป็นจาก ArcGIS Maps SDK และ Jetpack Compose รวมถึงคลาสสำหรับจัดการ UI อื่นๆ

2.-3. เพิ่ม annotation @file:OptIn(ExperimentalMaterial3Api::class) เพื่อเปิดใช้งานการใช้ Material Design 3 ซึ่งเป็นชุดคอมโพเนนต์ UI ใหม่

MainScreen.kt

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

package com.example.app.screens

import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Text
import androidx.compose.material3.TopAppBar
import androidx.compose.runtime.Composable
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.stringResource
import com.arcgismaps.mapping.ArcGISMap
import com.arcgismaps.mapping.BasemapStyle
import com.arcgismaps.mapping.Viewpoint
import com.arcgismaps.toolkit.geoviewcompose.MapView
import com.example.app.R
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/display-a-map/

การสร้างแผนที่

3. สร้างฟังก์ชัน createMap()

3.1. สร้างฟังก์ชัน createMap() ที่จะทำหน้าที่สร้างออบเจ็กต์ ArcGISMap และคืนค่ากลับมา ตัวอย่างดังรูป

MainScreen.kt

```
fun createMap(): ArcGISMap {

}
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/display-a-map/

การสร้างแผนที่

3.2. ภายในฟังก์ชัน createMap() สร้างออบเจ็กต์ ArcGISMap โดยใช้ BasemapStyle.ArcGISTopographic ซึ่งเป็นBase Map ที่แสดงภูมิประเทศ ตัวอย่างดังรูป

MainScreen.kt

Expand

```
fun createMap(): ArcGISMap {

    return ArcGISMap(BasemapStyle.ArcGISTopographic).apply {

    }

}
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/display-a-map/

การสร้างแผนที่

3.3. ใช้บล็อก apply เพื่อกำหนดค่าเริ่มต้นของแผนที่:

1 สร้างออบเจ็กต์ Viewpoint ซึ่งประกอบด้วย:

- latitude: ละติจูดของตำแหน่งที่จะแสดงบนแผนที่ (ในที่นี้คือ 34.0270)
- longitude: ลองจิจูดของตำแหน่งที่จะแสดงบนแผนที่ (ในที่นี้คือ -118.8050)
- scale: ระดับการซูมของแผนที่ (ในที่นี้คือ 72000.0)

2 กำหนดค่า Viewpoint ที่สร้างขึ้นให้กับ initialViewpoint ของออบเจ็กต์ ArcGISMap ตัวอย่างดังรูป

MainScreen.kt

Expand

```
fun createMap(): ArcGISMap {

    return ArcGISMap(BasemapStyle.ArcGISTopographic).apply {

        initialViewpoint = Viewpoint(
            latitude = 34.0270,
            longitude = -118.8050,

            scale = 72000.0

        )

    }

}
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/display-a-map/

การสร้างหน้าจอหลักเพื่อแสดงแผนที่

การแสดงแผนที่ที่เราสร้างไว้ก่อนหน้านี้บนหน้าจอ

1. สร้าง Composable Function MainScreen()

1.1.ภายในไฟล์ MainScreen.kt สร้าง Composable function ชื่อ MainScreen() โดยใช้ annotation @Composable ซึ่งจะทำหน้าที่สร้าง UI สำหรับหน้าจอหลักของแอปพลิเคชัน ตัวอย่างดังรูป

MainScreen.kt

Expand

```
@Composable
fun MainScreen() {


}
```

Expand

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/display-a-map/

การสร้างหน้าจอหลักเพื่อแสดงแผนที่

2. สร้างและเก็บ Instance ของ ArcGISMap

2.1.ภายใน MainScreen() ใช้ remember block เพื่อสร้างและเก็บ instance ของ ArcGISMap ที่ได้จากฟังก์ชัน createMap() ที่สร้างไว้ก่อนหน้านี้ (remember ทำให้ instance ของ ArcGISMap คงอยู่แม้จะมีการ recompose ของ UI ช่วยให้ไม่ต้องสร้างแผนที่ใหม่ทุกครั้งที่ UI มีการอัปเดต)

MainScreen.kt

```
@Composable
fun MainScreen() {

    val map = remember {
        createMap()
    }

}
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/display-a-map/

การสร้างหน้าจอหลักเพื่อแสดงแผนที่

3. สร้าง Layout ด้วย Scaffold

3.1ใช้ Composable function Scaffold จาก Jetpack Compose เพื่อสร้างโครงสร้างพื้นฐานของหน้าจอ ซึ่งในที่นี้ประกอบด้วย

- TopAppBar: แถบด้านบนของหน้าจอ โดยในตัวอย่างนี้จะแสดงชื่อแอปพลิเคชัน (ดึงมาจาก stringResource(id = R.string.app_name))
- trailing lambda: ส่วนเนื้อหาหลักของหน้าจอที่จะใช้สำหรับแสดงแผนที่

MainScreen.kt

Expand

```
@Composable
fun MainScreen() {

    val map = remember {
        createMap()
    }

    Scaffold(
        topBar = { TopAppBar(title = { Text(text = stringResource(id = R.string.app_name)) }) }
    ) {

    }

}
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/display-a-map/

การสร้างหน้าจอหลักเพื่อแสดงแผนที่

4. แสดงแผนที่ด้วย MapView

4.1 ภายใน trailing lambda ของ Scaffold, เรียกใช้ Composable function MapView จาก ArcGIS Maps SDK for Kotlin Toolkit ซึ่งเป็นคอมโพเนนต์ที่ใช้แสดงแผนที่

4.2 กำหนด Modifier ให้กับ MapView ดังนี้ :

- fillMaxSize(): ทำให้แผนที่ขยายเต็มพื้นที่ที่ว่างอยู่
- padding(it): เพิ่มระยะห่างระหว่างแผนที่กับขอบของหน้าจอ โดยใช้ padding ที่ Scaffold กำหนดไว้

4.3 ส่ง instance ของ ArcGISMap (ที่เก็บอยู่ในตัวแปร map) ให้กับพารามิเตอร์ arcGISMap ของ MapView เพื่อให้ MapView รู้ว่าจะต้องแสดงแผนที่ใด

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/display-a-map/

การสร้างหน้าจอหลักเพื่อแสดงแผนที่

4. แสดงแผนที่ด้วย MapView

ตัวอย่างโค้ด

MainScreen.kt

Expand

```
@Composable
fun MainScreen() {

    val map = remember {
        createMap()
    }

    Scaffold(
        topBar = { TopAppBar(title = { Text(text = stringResource(id = R.string.app_name)) }) }
    ) {

        MapView(
            modifier = Modifier.fillMaxSize().padding(it),
            arcGISMap = map
        )

    }

}
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/display-a-map/

การเรียกใช้ MainScreen ใน MainActivity

ในส่วนนี้จะสอนวิธีการเชื่อมต่อ Composable function MainScreen() ที่เราสร้างไว้ก่อนหน้านี้ เข้ากับ Activity หลักของแอปพลิเคชัน (MainActivity) เพื่อให้ UI ของแผ
ที่แสดงผลบนหน้าจอเมื่อแอปพลิเคชันเริ่มทำงาน

1 . เตรียมไฟล์ MainActivity.kt

1.1. เปิดไฟล์ MainActivity.kt ซึ่งเป็น Activity หลักของแอปพลิเคชัน

1.2.ลบโค้ดทั้งหมดที่ไม่จำเป็นออก เหลือไว้เพียงแค่ package declaration และ class definition ของ MainActivity

MainActivity.kt

```
package com.example.app

class MainActivity : ComponentActivity() {

}
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/display-a-map/

การเรียกใช้ MainScreen ใน MainActivity

2. เพิ่ม Imports ที่จำเป็น

2.1. เพิ่มส่วน import เพื่อนำเข้าคลาสและฟังก์ชันที่จำเป็นสำหรับการทำงานร่วมกับ Jetpack Compose และ ArcGIS Maps SDK

MainActivity.kt

```kotlin
package com.example.app

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import com.arcgismaps.ApiKey
import com.arcgismaps.ArcGISEnvironment
import com.example.app.screens.MainScreen
import com.example.app.ui.theme.DisplayAMapTheme

class MainActivity : ComponentActivity() {

}
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/display-a-map/

การเรียกใช้ MainScreen ใน MainActivity

3. เรียกใช้ MainScreen ใน onCreate()

3.1. ภายในฟังก์ชัน onCreate() ซึ่งเป็น lifecycle method ของ Activity ที่จะถูกเรียกใช้เมื่อ Activity เริ่มทำงาน โดยให้ทำดังนี้

1. เรียก super.onCreate(savedInstanceState) เพื่อเรียกใช้ onCreate() ของคลาส parent (ComponentActivity)

2. เรียกใช้ฟังก์ชัน setContent เพื่อกำหนด composable content ให้กับ Activity

- ภายใน setContent ใช้ DisplayAMapTheme ซึ่งเป็น theme ของแอปพลิเคชัน เพื่อห่อหุ้ม MainScreen()

- การห่อหุ้ม MainScreen() ด้วย DisplayAMapTheme จะทำให้ UI ของ Map ที่ถูกสร้างขึ้นมาใช้ theme ของแอปพลิเคชัน ซึ่งช่วยให้ UI มีความสอดคล้องกัน

MainActivity.kt

Expand

```kotlin
class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        setContent {
            DisplayAMapTheme {
                MainScreen()
            }
        }

    }

}
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/display-a-map/

การ Set API Key

ในส่วนนี้จะสอนวิธีการตั้งค่า API key เพื่อให้ ArcGIS Maps SDK for Kotlin สามารถใช้งานฟังก์ชันต่างๆ ได้อย่างถูกต้อง

1. สร้างเมธอด setApiKey()

1.1. ภายในคลาส MainActivity, สร้างเมธอด private ชื่อ setApiKey()

2. กำหนดค่า ArcGISEnvironment.apiKey

2.1. ภายในเมธอด setApiKey() กำหนดค่าให้กับ ArcGISEnvironment.apiKey โดยใช้ ApiKey.create() และส่ง access token ของคุณ (API key) เป็นสตริง

2.2. อย่าลืมใส่ access token ไว้ภายในเครื่องหมาย double quotes ("")

MainActivity.kt

Expand

```kotlin
class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        setContent {
            DisplayAMapTheme {
                MainScreen()
            }
        }

    }

    private fun setApiKey() {

        ArcGISEnvironment.apiKey = ApiKey.create("YOUR_ACCESS_TOKEN")

    }

}
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/display-a-map/

การ Set API Key

ข้อควรระวัง

- การเก็บ access token ไว้ในโค้ดโดยตรงไม่ใช่แนวทางปฏิบัติที่ดี (best practice) เนื่องจากมีความเสี่ยงด้านความปลอดภัย
- ในแอปพลิเคชันจริง ควรเก็บ access token อย่างปลอดภัย เช่น การเข้ารหัส หรือการใช้ Keychain (สำหรับ iOS) หรือ KeyStore (สำหรับ Android)

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/display-a-map/

การ Set API Key

3. เรียกใช้ setApiKey() ใน onCreate()

3.1. ภายในเมธอด onCreate() ของ MainActivity เรียกใช้เมธอด setApiKey() ก่อนที่จะเรียกใช้ setContent block

3.2.การเรียกใช้ setApiKey() ก่อนจะทำให้ ArcGIS Environment ได้รับ API key ก่อนที่จะเริ่มสร้าง UI ของแผนที่

MainActivity.kt

Expand

```
class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        setApiKey()

        setContent {
            DisplayAMapTheme {
                MainScreen()
            }
        }

    }

    private fun setApiKey() {

        ArcGISEnvironment.apiKey = ApiKey.create("YOUR_ACCESS_TOKEN")

    }

}
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/display-a-map/

การ Run Application

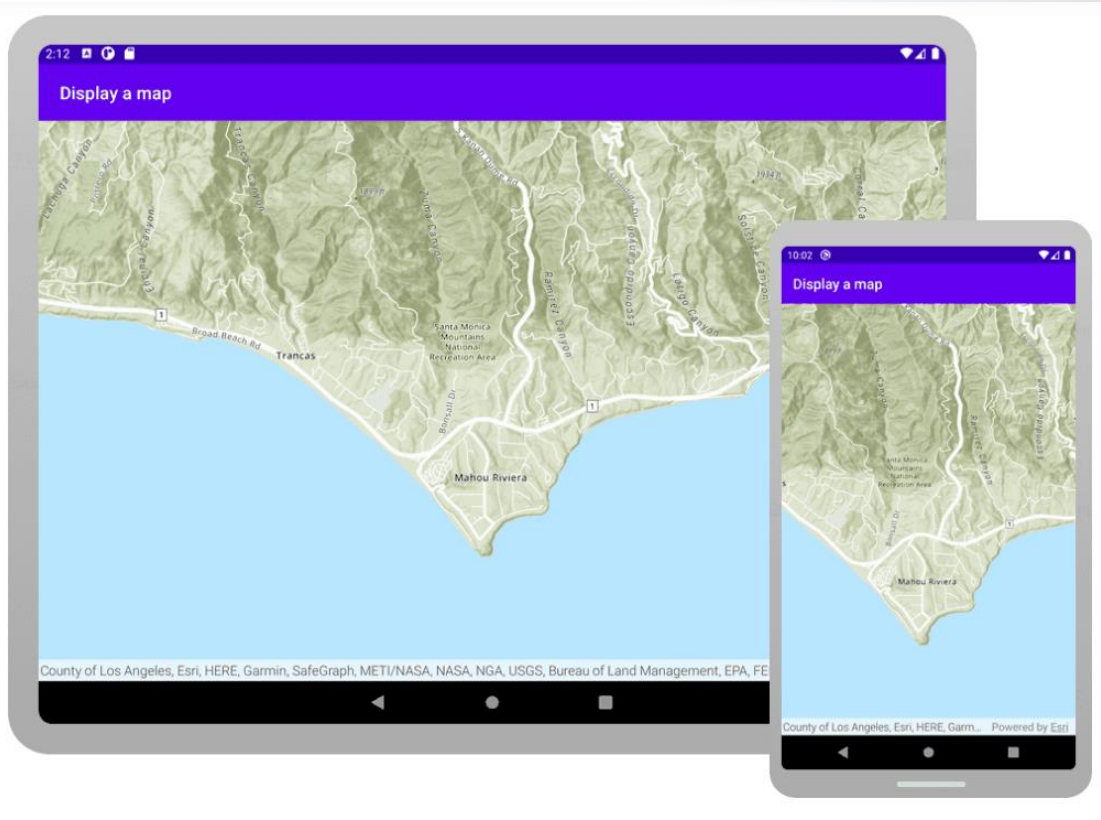
หลังจากที่เราได้สร้างและ Setup Project รวมถึงเขียนโค้ดสำหรับแสดงแผนที่แล้ว ขั้นตอนสุดท้ายคือการ Run Application เพื่อดูผลลัพธ์

1. Run Application

1.1. ใน Android Studio คลิกที่เมนู Run > Run 'app'

1.2. Application จะถูก build และติดตั้งลงบนอุปกรณ์หรือ emulator ที่เลือกไว้

1.3. เมื่อ Application เริ่มทำงาน จะเห็นแผนที่แสดงบนหน้าจอ



ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/display-a-map/

การเพิ่ม Graphics Overlay

ในส่วนนี้จะสอนวิธีการเพิ่ม Graphics Overlay ซึ่งเป็นองค์ประกอบสำคัญในการแสดงผล Graphics ต่างๆ เช่น Point, Line และ Polygon บนแผนที่ในแอปพลิเคชัน Android โดยใช้ Jetpack Compose และ ArcGIS Maps SDK for Kotlin

1. เพิ่ม Imports ที่จำเป็น

1.1. ในไฟล์ MainScreen.kt ตรวจสอบว่ามีการ import คลาสและฟังก์ชันที่จำเป็นสำหรับการทำงานกับ Graphics Overlay

- com.arcgismaps.mapping.view.Graphic: แทนข้อมูลของ graphic แต่ละตัว
- com.arcgismaps.mapping.view.GraphicsOverlay: เป็น container สำหรับเก็บ graphic หลายๆ ตัว
- ในส่วนของ imports อื่นๆ ที่เกี่ยวข้องกับการสร้าง geometry และ symbol ซึ่งจะใช้ในภายหลัง

MainScreen.kt

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

package com.example.app.screens

import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Text
import androidx.compose.material3.TopAppBar
import androidx.compose.runtime.Composable
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.stringResource
import com.arcgismaps.Color
import com.arcgismaps.geometry.Point
import com.arcgismaps.geometry.PolygonBuilder
import com.arcgismaps.geometry.PolylineBuilder
import com.arcgismaps.geometry.SpatialReference
import com.arcgismaps.mapping.ArcGISMap
import com.arcgismaps.mapping.BasemapStyle
import com.arcgismaps.mapping.Viewpoint
import com.arcgismaps.mapping.symbology.SimpleFillSymbol
import com.arcgismaps.mapping.symbology.SimpleFillSymbolStyle
import com.arcgismaps.mapping.symbology.SimpleLineSymbol
import com.arcgismaps.mapping.symbology.SimpleLineSymbolStyle
import com.arcgismaps.mapping.symbology.SimpleMarkerSymbol
import com.arcgismaps.mapping.symbology.SimpleMarkerSymbolStyle
import com.arcgismaps.mapping.view.Graphic
import com.arcgismaps.mapping.view.GraphicsOverlay
import com.arcgismaps.toolkit.geoviewcompose.MapView
import com.example.app.R
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

การเพิ่ม Graphics Overlay

2. สร้าง Graphics Overlay

2.1. ภายใน Composable function MainScreen() ใช้ remember block เพื่อสร้าง instance ของ GraphicsOverlay

2.2. remember ทำให้ instance นี้คงอยู่ตลอดอายุการใช้งานของ Composable นี้ ป้องกันการสร้างซ้ำโดยไม่จำเป็นและประหยัดทรัพยากร

MainScreen.kt

Expand

```
@Composable
fun MainScreen() {
    // create an ArcGISMap
    val map = remember {
        createMap()
    }

    // Create a graphics overlay.
    val graphicsOverlay = remember { GraphicsOverlay() }

    Scaffold(
        topBar = { TopAppBar(title = { Text(text = stringResource(id = R.string.app_name)) }) }
    ) {
        MapView(
            modifier = Modifier.fillMaxSize().padding(it),
            arcGISMap = map,

        )
    }
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

การเพิ่ม Graphics Overlay

3. สร้าง List ของ Graphics Overlays

3.1. สร้าง List ที่มี graphicsOverlay เพียงตัวเดียว ที่เราสร้างขึ้นในขั้นตอนที่แล้ว (โดยทั่วไป แอปพลิเคชันอาจมี Graphics Overlay หลายตัวเพื่อแยกประเภทหรือจัดการ Graphics ต่างๆ แต่ในตัวอย่างนี้ใช้เพียงอัน[illegible])

MainScreen.kt

Expand

```
@Composable
fun MainScreen() {
    // create an ArcGISMap
    val map = remember {
        createMap()
    }

    // Create a graphics overlay.
    val graphicsOverlay = remember { GraphicsOverlay() }

    // Create a list of graphics overlays used by the MapView
    val graphicsOverlays = remember { listOf(graphicsOverlay) }

    Scaffold(
        topBar = { TopAppBar(title = { Text(text = stringResource(id = R.string.app_name)) }) }
    ) {
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

การเพิ่ม Graphics Overlay

4. ส่ง Graphics Overlays ไปยัง MapView

4.1. ใน composable function MapView เพิ่ม argument graphicsOverlays

4.2. ส่ง List graphicsOverlays (ที่สร้างขึ้นในขั้นตอนที่ 3) ไปยัง MapView

4.3. MapView จะรับ List นี้และแสดงผล Graphics ที่อยู่ใน graphicsOverlay บนแผนที่

MainScreen.kt

Expand

```
Scaffold(
    topBar = { TopAppBar(title = { Text(text = stringResource(id = R.string.app_name)) }) }
) {
    MapView(
        modifier = Modifier.fillMaxSize().padding(it),
        arcGISMap = map,

        graphicsOverlays = graphicsOverlays

    )
}
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

การเพิ่ม Graphics Overlay

เหตุผลที่ต้องใช้ List ในการส่งเข้าไปใน graphicsOverlays

- MapView ถูกออกแบบมาให้รองรับ Graphics Overlay หลายๆ อัน
- การส่ง List ช่วยให้มีความยืดหยุ่นในการเพิ่มหรือลบ Graphics Overlay ในภายหลังได้ง่าย
- แม้ว่าในตัวอย่างนี้ใช้ List ที่มีสมาชิกเพียงตัวเดียว แต่ในแอปพลิเคชันที่ซับซ้อนกว่า อาจต้องการแยก Graphics Overlay ตามประเภทของข้อมูล เช่น แยก Graphics Overlay สำหรับ marker, polyline, และ polygon

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

การเพิ่ม Point Graphic

ในส่วนนี้จะสอนวิธีการเพิ่มจุด (point) ลงบนแผนที่ในแอปพลิเคชัน Android โดยใช้ ArcGIS Maps SDK for Kotlin และ Jetpack Compose

1. สร้างสัญลักษณ์สำหรับเส้นขอบจุด (Outline)

1.1. ภายในไฟล์ MainScreen.kt สร้าง property ระดับบนสุด ชื่อ blueOutlineSymbol โดยใช้ by lazy ซึ่งจะทำให้สัญลักษณ์ถูกสร้างขึ้นเมื่อมีการใช้งานครั้งแรก

1.2. ภายใน lazy block สร้าง SimpleLineSymbol ที่มีลักษณะตามลำดับดังนี้:

- style: เป็นเส้นทึบ (SimpleLineSymbolStyle.Solid)
- color: สีน้ำเงิน (Color.fromRgba(0, 0, 255))
- width: ความหนา 2 จุด (2f)

MainScreen.kt

Expand

```
// Create a blue outline symbol.
private val blueOutlineSymbol by lazy {
    SimpleLineSymbol(SimpleLineSymbolStyle.Solid, Color.fromRgba(0, 0, 255), 2f)
}
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

การเพิ่ม Point Graphic

2. สร้าง Point Graphic

2.1.สร้าง property ระดับบนสุดชื่อ pointGraphic โดยใช้ by lazy

MainScreen.kt

Expand

```
private val pointGraphic by lazy {

}
```

Expand

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

การเพิ่ม Point Graphic

2. สร้าง Point Graphic

2.2. ภายใน lazy block

2.2.1สร้าง Point โดยกำหนดค่าดังนี้

- x: ลองจิจูด (ในที่นี้คือ -118.8065)
- y: ละติจูด (ในที่นี้คือ 34.0005)
- spatialReference: ระบบอ้างอิงพิกัด (ในที่นี้คือ WGS84 ซึ่งใช้สำหรับละติจูดและลองจิจูด)

MainScreen.kt

Expand

```
private val pointGraphic by lazy {

    // Create a point geometry with a location and spatial reference.
    // Point(latitude, longitude, spatial reference)
    val point = Point(
        x = -118.8065,
        y = 34.0005,
        spatialReference = SpatialReference.wgs84()
    )

    // Create a point symbol that is an small red circle and assign the blue outline symbol to its outline
    val simpleMarkerSymbol = SimpleMarkerSymbol(
        style = SimpleMarkerSymbolStyle.Circle,
        color = Color.red,
        size = 10f
    )
    simpleMarkerSymbol.outline = blueOutlineSymbol

}
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

การเพิ่ม Point Graphic

2. สร้าง Point Graphic

2.2. ภายใน lazy block

2.2.2 สร้าง SimpleMarkerSymbol ที่มีลักษณะดังนี้

- style: รูปวงกลม (SimpleMarkerSymbolStyle.Circle)
- color: สีแดง (Color.red)
- size: ขนาด 10 จุด (10f)
- outline: กำหนดให้ใช้ blueOutlineSymbol ที่สร้างไว้ก่อน

หน้านี้เป็นเส้นขอบ

MainScreen.kt

Expand

```kotlin
private val pointGraphic by lazy {

    // Create a point geometry with a location and spatial reference.
    // Point(latitude, longitude, spatial reference)
    val point = Point(
        x = -118.8065,
        y = 34.0005,
        spatialReference = SpatialReference.wgs84()
    )

    // Create a point symbol that is an small red circle and assign the blue outline symbol to its outline
    val simpleMarkerSymbol = SimpleMarkerSymbol(
        style = SimpleMarkerSymbolStyle.Circle,
        color = Color.red,
        size = 10f
    )
    simpleMarkerSymbol.outline = blueOutlineSymbol

}
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

การเพิ่ม Point Graphic

2. สร้าง Point Graphic

2.2. ภายใน lazy block

2.2.3 สร้าง Graphic โดยส่ง Point และ SimpleMarkerSymbol ที่สร้างขึ้นเข้าไป

MainScreen.kt

Expand

```kotlin
private val pointGraphic by lazy {

    // Create a point geometry with a location and spatial reference.
    // Point(latitude, longitude, spatial reference)
    val point = Point(
        x = -118.8065,
        y = 34.0005,
        spatialReference = SpatialReference.wgs84()
    )

    // Create a point symbol that is an small red circle and assign the blue outline symbol to its outline property.
    val simpleMarkerSymbol = SimpleMarkerSymbol(
        style = SimpleMarkerSymbolStyle.Circle,
        color = Color.red,
        size = 10f
    )
    simpleMarkerSymbol.outline = blueOutlineSymbol

    // Create a graphic with the point geometry and symbol.
    Graphic(
        geometry = point,
        symbol = simpleMarkerSymbol
    )
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

การเพิ่ม Point Graphic

3. เพิ่ม Point Graphic ไปยัง Graphics Overlay

3.1. ภายใน MainScreen เพิ่ม pointGraphic ที่สร้างขึ้นไปยัง graphics property ของ graphicsOverlay

MainScreen.kt

Expand

```kotlin
@Composable
fun MainScreen() {
    // create an ArcGISMap
    val map = remember {
        createMap()
    }

    // Create a graphics overlay.
    val graphicsOverlay = remember { GraphicsOverlay() }

    // Add the point graphic to the graphics overlay.
    graphicsOverlay.graphics.add(pointGraphic)

    // Create a list of graphics overlays used by the MapView
    val graphicsOverlays = remember { listOf(graphicsOverlay) }
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

การเพิ่ม Point Graphic

4. Run Application

4.1. คลิก Run > Run 'app' เพื่อ Run Application

4.2. เมื่อ Run เสร็จแล้วจะเห็นจุดสีแดงที่มีเส้นขอบสีน้ำเงินปรากฏขึ้นบนแผนที่ ตรง Point Dume State Beach (ตำแหน่งที่ได้กำหนดไว้)

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

การเพิ่ม Line Graphic

ในส่วนนี้จะสอนวิธีการเพิ่มเส้น (polyline) ลงบนแผนที่ในแอปพลิเคชัน Android โดยใช้ ArcGIS Maps SDK for Kotlin และ Jetpack Compose

1. สร้าง Polyline Graphic

1.1. ภายในไฟล์ MainScreen.kt สร้าง property ระดับบนสุด ชื่อ polylineGraphic โดยใช้ by lazy ซึ่งจะทำให้ graphic ถูกสร้างขึ้นเมื่อมีการใช้งานครั้งแรก

MainScreen.kt

Expand

```
private val polylineGraphic by lazy {

}
```

Expand

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

การเพิ่ม Line Graphic

1. สร้าง Polyline Graphic

1.2. ภายใน lazy block

1.2.1 สร้าง SimpleLineSymbol เพื่อกำหนดลักษณะของเส้น

- style: เป็นเส้นทึบ (SimpleLineSymbolStyle.Solid)
- color: สีน้ำเงิน (Color.fromRgba(0, 0, 255))
- width: ความหนา 3 จุด (3f)

MainScreen.kt

Expand

```
private val polylineGraphic by lazy {

    // Create a blue line symbol for the polyline.
    val polylineSymbol = SimpleLineSymbol(
        style = SimpleLineSymbolStyle.Solid,
        color = Color.fromRgba(0, 0, 255),
        width = 3f
    )

    // Create a polylineBuilder with a spatial reference and add three points to it.
    val polylineBuilder = PolylineBuilder(SpatialReference.wgs84()) {
        addPoint(-118.8215, 34.0139)
        addPoint(-118.8148, 34.0080)
        addPoint(-118.8088, 34.0016)
    }
    // then get the polyline from the polyline builder
    val polyline = polylineBuilder.toGeometry()

}
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

การเพิ่ม Line Graphic

1. สร้าง Polyline Graphic

1.2. ภายใน lazy block

1.2.2 สร้าง PolylineBuilder เพื่อสร้าง polyline

- กำหนด SpatialReference เป็น WGS84

- ใช้ addPoint เพื่อเพิ่มจุด (ละติจูด, ลองจิจูด) เข้าไปใน polyline

1.2.3 ใช้ toGeometry() เพื่อแปลง PolylineBuilder เป็น Polyline object ซึ่งเป็น geometry ของเส้น

1.2.4 สร้าง Graphic โดยส่ง Polyline และ SimpleLineSymbol ที่สร้างขึ้นเข้าไป

MainScreen.kt

Expand

```kotlin
private val polylineGraphic by lazy {

    // Create a blue line symbol for the polyline.
    val polylineSymbol = SimpleLineSymbol(
        style = SimpleLineSymbolStyle.Solid,
        color = Color.fromRgba(0, 0, 255),
        width = 3f
    )

    // Create a polylineBuilder with a spatial reference and add three points to it.
    val polylineBuilder = PolylineBuilder(SpatialReference.wgs84()) {
        addPoint(-118.8215, 34.0139)
        addPoint(-118.8148, 34.0080)
        addPoint(-118.8088, 34.0016)
    }
    // then get the polyline from the polyline builder
    val polyline = polylineBuilder.toGeometry()

}
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

การเพิ่ม Line Graphic

1. สร้าง Polyline Graphic

1.2. ภายใน lazy block

1.2.4 สร้าง Graphic โดยส่ง Polyline และ SimpleLineSymbol ที่สร้างขึ้นเข้าไป

MainScreen.kt

Expand

```
private val polylineGraphic by lazy {

    // Create a blue line symbol for the polyline.
    val polylineSymbol = SimpleLineSymbol(
        style = SimpleLineSymbolStyle.Solid,
        color = Color.fromRgba(0, 0, 255),
        width = 3f
    )

    // Create a polylineBuilder with a spatial reference and add three points to it.
    val polylineBuilder = PolylineBuilder(SpatialReference.wgs84()) {
        addPoint(-118.8215, 34.0139)
        addPoint(-118.8148, 34.0080)
        addPoint(-118.8088, 34.0016)
    }
    // then get the polyline from the polyline builder
    val polyline = polylineBuilder.toGeometry()

    // Create a polyline graphic with the polyline geometry and symbol.
    Graphic(
        geometry = polyline,
        symbol = polylineSymbol
    )

}
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

การเพิ่ม Line Graphic

2. เพิ่ม Polyline Graphic ไปยัง Graphics Overlay

2.1.ภายใน MainScreen เพิ่ม polylineGraphic ที่สร้างขึ้นไปยัง graphics property ของ graphicsOverlay เช่นเดียวกับที่ทำกับ pointGraphic

MainScreen.kt

Expand

```kotlin
@Composable
fun MainScreen() {
    // create an ArcGISMap
    val map = remember {
        createMap()
    }

    // Create a graphics overlay.
    val graphicsOverlay = remember { GraphicsOverlay() }

    // Add the point graphic to the graphics overlay.
    graphicsOverlay.graphics.add(pointGraphic)

    // Add the polyline graphic to the graphics overlay.
    graphicsOverlay.graphics.add(polylineGraphic)

    // Create a list of graphics overlays used by the MapView
    val graphicsOverlays = remember { listOf(graphicsOverlay) }
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

การเพิ่ม Line Graphic

3. Run Application

3.1. คลิก Run > Run 'app' เพื่อ Run Application

3.2. เมื่อ Run เสร็จแล้วจะเห็นเส้นสีน้ำเงินปรากฏขึ้นบนแผนที่ เชื่อมต่อจุดที่คุณกำหนดไว้ใน PolylineBuilder

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

การเพิ่ม Polygon Graphic

ในส่วนนี้จะสอนวิธีการเพิ่มรูปหลายเหลี่ยม (polygon)ลงบนแผนที่ในแอปพลิเคชัน Android โดยใช้ ArcGIS Maps SDK for Kotlin และ Jetpack Compose

1. สร้าง Polygon Graphic

1.1. ภายในไฟล์ MainScreen.kt สร้าง property ระดับบนสุด ชื่อ polygonGraphic โดยใช้ by lazy ซึ่งจะทำให้ graphic ถูกสร้างขึ้นเมื่อมีการใช้งานครั้งแรก

MainScreen.kt

Expand

```
private val polygonGraphic by lazy {

}
```

Expand

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

การเพิ่ม Polygon Graphic

1. สร้าง Polygon Graphic

1.2. ภายใน lazy block

1.2.1 สร้าง PolygonBuilder เพื่อสร้าง polygon

- กำหนด SpatialReference เป็น WGS84 (ระบบอ้างอิงพิกัดสำหรับละติจูดและลองจิจูด)
- ใช้ addPoint เพื่อเพิ่มจุด (ละติจูด, ลองจิจูด) ที่เป็นมุมของ polygon เข้าไป

1.2.2 ใช้ toGeometry() เพื่อแปลง PolygonBuilder เป็น Polygon object ซึ่งเป็น geometry ของรูปหลายเหลี่ยม

MainScreen.kt

Expand

```kotlin
private val polygonGraphic by lazy {

    // Create a polygon builder with a spatial reference and add five vertices (points) to it.
    // Then get the polygon from the polygon builder.
    val polygonBuilder = PolygonBuilder(SpatialReference.wgs84()) {
        addPoint(-118.8189, 34.0137)
        addPoint(-118.8067, 34.0215)
        addPoint(-118.7914, 34.0163)
        addPoint(-118.7959, 34.0085)
        addPoint(-118.8085, 34.0035)
    }
    val polygon = polygonBuilder.toGeometry()

    // Create a red fill symbol with an alpha component of 128: values can run from 0 to 255).
    val polygonFillSymbol = SimpleFillSymbol(
        style = SimpleFillSymbolStyle.Solid,
        color = Color.fromRgba(255, 0, 0, 128),
        outline = blueOutlineSymbol
    )

}
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

การเพิ่ม Polygon Graphic

1. สร้าง Polygon Graphic

1.2. ภายใน lazy block

1.2.3 สร้าง SimpleFillSymbol เพื่อกำหนดลักษณะของพื้นที่ภายใน polygon

- style: เป็นพื้นที่ทึบ (SimpleFillSymbolStyle.Solid)
- color: สีแดงที่มีความโปร่งแสง (Color.fromRgba(255, 0, 0, 128)) โดย alpha 128 หมายถึงความโปร่งแสง 50%
- outline: กำหนดให้ใช้ blueOutlineSymbol ที่สร้างไว้ก่อนหน้านี้เป็นเส้นขอบ

MainScreen.kt

Expand

```kotlin
private val polygonGraphic by lazy {

    // Create a polygon builder with a spatial reference and add five vertices (points) to it.
    // Then get the polygon from the polygon builder.
    val polygonBuilder = PolygonBuilder(SpatialReference.wgs84()) {
        addPoint(-118.8189, 34.0137)
        addPoint(-118.8067, 34.0215)
        addPoint(-118.7914, 34.0163)
        addPoint(-118.7959, 34.0085)
        addPoint(-118.8085, 34.0035)
    }
    val polygon = polygonBuilder.toGeometry()

    // Create a red fill symbol with an alpha component of 128: values can run from 0 to 255).
    val polygonFillSymbol = SimpleFillSymbol(
        style = SimpleFillSymbolStyle.Solid,
        color = Color.fromRgba(255, 0, 0, 128),
        outline = blueOutlineSymbol
    )

}
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

การเพิ่ม Polygon Graphic

1. สร้าง Polygon Graphic

1.2. ภายใน lazy block

1.2.4 สร้าง Graphic โดยส่ง Polygon และ SimpleFillSymbol ที่สร้างขึ้นเข้าไป

MainScreen.kt

Expand

```kotlin
private val polygonGraphic by lazy {

    // Create a polygon builder with a spatial reference and add five vertices (points) to it.
    // Then get the polygon from the polygon builder.
    val polygonBuilder = PolygonBuilder(SpatialReference.wgs84()) {
        addPoint(-118.8189, 34.0137)
        addPoint(-118.8067, 34.0215)
        addPoint(-118.7914, 34.0163)
        addPoint(-118.7959, 34.0085)
        addPoint(-118.8085, 34.0035)
    }
    val polygon = polygonBuilder.toGeometry()

    // Create a red fill symbol with an alpha component of 128: values can run from 0 to 255).
    val polygonFillSymbol = SimpleFillSymbol(
        style = SimpleFillSymbolStyle.Solid,
        color = Color.fromRgba(255, 0, 0, 128),
        outline = blueOutlineSymbol
    )

    // Create a polygon graphic from the polygon geometry and symbol.
    Graphic(
        geometry = polygon,
        symbol = polygonFillSymbol
    )

}
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

การเพิ่ม Polygon Graphic

2. เพิ่ม Polygon Graphic ไปยัง Graphics Overlay

2.1. ภายใน MainScreen, เพิ่ม polygonGraphic ที่สร้างขึ้นไปยัง graphics property ของ graphicsOverlay เช่นเดียวกับที่ทำกับ pointGraphic และ polylineGraphic

MainScreen.kt

Expand

```
@Composable
fun MainScreen() {
    // create an ArcGISMap
    val map = remember {
        createMap()
    }

    // Create a graphics overlay.
    val graphicsOverlay = remember { GraphicsOverlay() }

    // Add the point graphic to the graphics overlay.
    graphicsOverlay.graphics.add(pointGraphic)

    // Add the polyline graphic to the graphics overlay.
    graphicsOverlay.graphics.add(polylineGraphic)

    // Add the polygon graphic to the graphics overlay.
    graphicsOverlay.graphics.add(polygonGraphic)

    // Create a list of graphics overlays used by the MapView
    val graphicsOverlays = remember { listOf(graphicsOverlay) }
```

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

การเพิ่ม Polygon Graphic

3. Run Application

3.1. คลิก Run > Run 'app' เพื่อ Run Application

3.2. เมื่อ Run เสร็จแล้วจะเห็นรูปหลายเหลี่ยมสีแดงโปร่งแสงที่มีเส้นขอบสีน้ำเงินปรากฏขึ้นบนแผนที่ครอบคลุมพื้นที่ที่กำหนดจุดไว้ใน PolygonBuilder

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

Layer Types

1.Feature Layer

1.1 คืออะไร?

- Feature Layer คือเลเยอร์ที่ใช้แสดงข้อมูลเชิงพื้นที่ (spatial data) ในรูปแบบของ features ซึ่งประกอบด้วยรูปทรงเรขาคณิต (geometry) และคุณสมบัติ (attributes)

1.2 ลักษณะ

- รูปทรงเรขาคณิต: จุด (point), เส้น (polyline), รูปหลายเหลี่ยม (polygon)
- คุณสมบัติ: ข้อมูลเพิ่มเติมที่อธิบายแต่ละ feature เช่น ชื่อ, ประเภท, หรือค่าอื่นๆ
- แหล่งข้อมูล: เชื่อมต่อกับ feature service ซึ่งให้บริการข้อมูลเชิงพื้นที่และไม่ใช่เชิงพื้นที่

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

Layer Types

1.Feature Layer

1.3 การใช้งาน

- เหมาะสำหรับการแสดงข้อมูลที่ต้องการการโต้ตอบกับผู้ใช้ เช่น การเลือก feature เพื่อดูรายละเอียด การแก้ไขข้อมูล หรือการวิเคราะห์ข้อมูลเชิงพื้นที่
- เหมาะสำหรับข้อมูลที่เปลี่ยนแปลงบ่อย เช่น ตำแหน่งของยานพาหนะ หรือข้อมูลสภาพอากาศ

1.4 ตัวอย่างการใช้งาน

- แสดงตำแหน่งร้านค้าบนแผนที่ พร้อมข้อมูลรายละเอียดของร้านค้าเมื่อคลิก
- แสดงขอบเขตการปกครองของจังหวัดต่างๆ ในประเทศไทย
- แสดงเส้นทางการเดินรถประจำทาง พร้อมข้อมูลตารางเวลา

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

Layer Types

2. Map Image Layer

2.1 คืออะไร?

- เลเยอร์ที่แสดงผลภาพแผนที่ที่สร้างไว้ล่วงหน้า (pre-rendered map images)

2.2 ลักษณะ

- ภาพแผนที่: ภาพที่สร้างไว้ล่วงหน้า มีความละเอียดสูง
- แหล่งข้อมูล: เชื่อมต่อกับ map service ซึ่งให้บริการภาพแผนที่ในรูปแบบต่างๆ

2.3 การใช้งาน

- เหมาะสำหรับการแสดงภาพแผนที่พื้นหลัง เช่น แผนที่ฐาน (basemap) หรือแผนที่แสดงภาพถ่ายดาวเทียม
- เหมาะสำหรับข้อมูลที่ไม่ต้องการการโต้ตอบกับผู้ใช้มากนัก เช่น ข้อมูลภูมิประเทศ หรือข้อมูลการใช้ที่ดิน

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

Layer Types

2. Map Image Layer

2.4 ตัวอย่างการใช้งาน

- แสดงภาพถ่ายดาวเทียมของโลกเป็นพื้นหลังของแผนที่
- แสดงแผนที่ถนนในระดับประเทศหรือภูมิภาค
- แสดงแผนที่ภูมิประเทศของภูเขาหรือพื้นที่ธรรมชาติ

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

Layer Types

3. Tile Layer

3.1 คืออะไร?

- เลเยอร์ที่แสดงผลภาพแผนที่ที่ถูกแบ่งเป็น tiles (ชิ้นส่วนเล็กๆ) ซึ่งจะถูกโหลดตามความจำเป็นเมื่อผู้ใช้ซูมหรือเลื่อนแผนที่

3.2 ลักษณะ

- ภาพแผนที่: แบ่งเป็น tiles ขนาดเล็ก
- แหล่งข้อมูล: เชื่อมต่อกับ tile service ซึ่งให้บริการ tiles ของแผนที่

3.3 การใช้งาน

- เหมาะสำหรับการแสดงข้อมูลขนาดใหญ่ เช่น แผนที่ฐานที่มีรายละเอียดสูง หรือแผนที่แสดงข้อมูลจำนวนมาก
- เหมาะสำหรับข้อมูลที่ต้องการประสิทธิภาพในการแสดงผลที่ดี เนื่องจากโหลดเฉพาะ tiles ที่จำเป็น

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

Layer Types

3. Tile Layer

3.4 ตัวอย่างการใช้งาน

- แสดงแผนที่ถนนแบบละเอียด พร้อมข้อมูลชื่อถนนและสถานที่สำคัญ
- แสดงภาพถ่ายทางอากาศที่มีความละเอียดสูง
- แสดงแผนที่ภูมิประเทศแบบเวกเตอร์ที่มีความละเอียดสูง

ข้อมูลเพิ่มเติม: https://developers.arcgis.com/kotlin/maps-2d/tutorials/add-a-point-line-and-polygon/

END