최종 수업 과제로 사용되는 OpenKnightsApp은 최신 Android 앱 구조(Compose + MVI + Clean Architecture + Firebase)를 모두 아우르는 훌륭한 예제입니다. 이 앱을 Android Studio 2025.1.1 + targetSDK=35 + navigation-3 + Jetpack Compose 기반으로 단계적으로 개발하면서 기존 수업의 주요 챕터들을 효과적으로 커버할 수 있습니다.

# ✓ OpenKnightsApp 개발 과정과 매핑되는 수업 챕터 정리

	9 11 "		
단계	주요 기능	관련 챕터	챕터 내용
<b>1</b> 단계디자인 시스템 구축	core-designsyst emKnightsTheme, KnightsButton 등	<b>Ch 1~6 (</b> 슬라이드 기반 <b>)</b>	Theme, Color, Typography 정의공통 컴포넌트 제작: Button, Card, TopAppBar 등
<b>2</b> 단계Navigat ion 구조 도입	화면 전환: 신청 → 제출 → 상세 등	Navigation-3 연계+ Ch 13 Activity & Intent	기존 Intent/Activity 구조 제거, Navigation-3의 screen(route) 함수로 타입 안전한 화면 전환
<b>3</b> 단계화면 상태관리 + MVI	UiState / UiEffect / UiIntent 구조 확립	Ch 14~16 Component+ Compose MVI 실습	ViewModel → Composable 흐름 실습, Toast / Navigation / Dialog를 UiEffect로 처리
<b>4</b> 단계신청 화면 구현	신청 팀 목록 불러오기 / 상세로 이동	Ch 17 데이터 저장	Room optional. Firestore 데이터와 UI 연동+ LazyColumn, TeamCard 구성
<b>5</b> 단계Firebas e Auth 연동	참가자 이메일 로그인운영자 구분	Ch 20 Firebase Auth	Firebase 인증 연동 (email 기반)Composable + ViewModel 로그인 화면 구성
<b>6</b> 단계Firestor e 연동	신청 데이터 저장, 심사 결과 저장	Ch 21-1 Firestore	Repository 패턴 + ViewModel → Firebase 연동
<b>7</b> 단계 <b>Storag</b> e 연동	앱 스크린샷 이미지 업로드/다운로드	Ch 21-2 Firebase Storage	putFile() + AsyncImage(Coil)로 이미지 처리
8단계위치/지 도 연동 (선택)	대회 장소, 위치 기반 지도 (선택 사항)	Ch 19 Google Maps API	GoogleMap() Compose API 적용
<b>9</b> 단계 <b>FCM</b> 연동 (선택)	푸시 알림 기능 (예선 결과 발표)	Ch 21-4 FCM	FirebaseMessagingService로 토큰 수신 및 로깅
<b>10</b> 단계앱 통합 + 배포	전체 기능 통합 + 최종 앱 빌드	수업 마무리 단계	모든 모듈 통합: app-openknights, core-* 정리Navigation 루트 통일, 디자인 일관성 검토

### 🔽 navigation-3 활용 시 강의 효과 강화 포인트

기존 방식

navigation-3 도입 효과

"detail/{id}" 같은 문자열 기반<br/>routeDestination.Detail(teamId) 처럼 함수 기반, IDE<br/>자동완성arguments?.getString("id")매개변수 함수 인자로 직접 받기 (타입 안전)

하나의 NavHost 안에서 화면 구성 모듈별 NavGraph 구성 → feature-team, feature-result 독립 가능  $\rightarrow$  특히 OpenKnightsApp은 신청 목록  $\rightarrow$  상세 보기  $\rightarrow$  제출 완료  $\rightarrow$  결과 보기 등 다단계 흐름 구조가 많아 navigation-3 도입 효과가 큽니다.

# ☑ 수업 로드맵 예시 (OpenKnightsApp 기준)

주차	수업 내용	실습 예시
1–2주	디자인 시스템 만들기	core-designsystem 제작, <b>KnightsTheme</b> 적용
3–4주	공통 컴포넌트 제작	KnightsButton, KnightsCard, KnightsTopAppBar
5-6주	화면 상태 관리 + Navigation-3	UiState / UiEffect 구성, 화면 전환 구조 설정
7-8주	Firebase Auth 연동	참가자 이메일 로그인 화면 제작
9–10주	Firestore + 팀 신청 화면	LazyColumn으로 팀 목록 구성, 데이터 바인딩
11–12주	이미지 업로드 (Storage)	팀 스크린샷 등록 기능, Coil 이미지 표시
13주	기능 통합 및 앱 완성	전체 Navigation 연결, 디자인 점검
14주	최종 테스트 및 발표	APK 빌드, 발표용 시연 및 코드 리뷰

# ☑ 결론

OpenKnightsApp은 단일 프로젝트로 Compose UI 수업의 모든 내용을 체계적으로 학습, 적용, 확장할 수 있는 완성도 높은 실전 과제입니다.

- navigation-3 기반으로 초기부터 설계하면 화면 구조와 전환이 명확해지고
- 기존 챕터 내용을 단계적으로 실습에 흡수할 수 있어 이론-실습 일관성이 높아집니다.

#### 필요하시다면 다음도 제공 가능합니다:

- 🧩 OpenKnightsApp 각 feature에 대한 navigation-3 기반 템플릿 코드
- 📝 전체 수업을 연결한 슬라이드/마크다운 요약
- 📁 각 주차별 실습 Starter/Final Code 저장소 구조

첨부한 pdf 파일을 수정해서 studio 2025.1.1 및 targetSDK=35 기준 Compose UI 수업에 사용하려고 해. Compose와 관련이 없는 부분을 정리해 주고, 이 pdf 내용과 관련된 Compose UI 관련 내용을 설명해줘.

#### 아주 좋은 질문입니다!

첨부하신 \*\*「11 제트팩 라이브러리.pdf」\*\*의 내용을 기준으로 Jetpack Compose UI 개발에서도 Jetpack 라이브러리를 왜, 언제, 어떤 앱에서 사용하는지를 구체적으로 설명드리겠습니다.

### 🔽 1. Compose UI 개발에서도 Jetpack 라이브러리를 쓰는 이유

Jetpack Compose는 UI 계층만 선언형으로 대체한 것입니다. 앱의 구조, 상태관리, 데이터 저장, 화면 이동, 권한 처리 등은 여전히 Jetpack 라이브러리를 통해 해결합니다.

- ⑥ Compose는 \*\*화면(View)\*\*만 바뀐 것이지,
- → 앱 구성 전반의 아키텍처와 기능은 여전히 **Jetpack**을 활용합니다.

### 🧱 2. PDF에 언급된 Jetpack 라이브러리 중 Compose에서 실제 사용하는 경우

라이브러리	Compose에서 사용 여부	용도
<pre>ViewModel (androidx.lifecycle.viewm odel)</pre>	✔ 매우 중요	화면 상태 보존, 비즈니스 로직 처리
Navigation (androidx.navigation.compose)	✔ 사용	화면 간 전환 (NavHost, NavController)
Room (androidx.room)	✔ 사용	로컬 DB 관리 (Compose에서도 Flow, LiveData 연동 가능)
DataStore (androidx.datastore)	✔ 사용	설정값 저장, SharedPreferences 대체
WorkManager	✓ 사용	백그라운드 작업 (Compose와 무관하지만 앱 전반에서 필요)
<pre>Paging 3 (androidx.paging.compose)</pre>	✔ 사용	대용량 목록 데이터 효율적 로딩
Hilt, Dagger	☑ 사용	의존성 주입 (Compose도 지원)
CameraX, Media3, ActivityResult API	✓ 사용	카메라, 미디어 재생, 파일 선택 등 Compose 외부 기능과 연동할 때 필요

### 3. 어떤 앱 개발 시 Jetpack 라이브러리를 Compose와 함께 쓰게 되는가?

#### 📌 (1) 실무 앱 전반

- 복잡한 앱 구조 필요 → ViewModel, Navigation, Hilt, Room, WorkManager 필수
- 예시:
  - ㅇ 커머스 앱
  - 채팅/메신저 앱
  - 뉴스/콘텐츠 앱
  - 사용자 설정/알림을 갖는 앱

#### 📌 (2) 데이터 기반 앱

- DB 저장 필요 → Room, DataStore
- 목록 무한 스크롤 → Paging3

### 📌 (3) 미디어 앱

- 이미지/비디오 촬영 → CameraX
- 오디오/비디오 재생 → Media3

### 📌 (4) 권한, 외부 연동 앱

● 위치, 파일, 사진 접근 → ActivityResult, Permissions, Sensors, Bluetooth

### 🎯 정리: Compose UI 개발과 Jetpack 라이브러리의 관계

구분 역할

**Jetpack Compose** 화면 UI를 선언형으로 구성

Jetpack 라이브러리 UI 외의 전반적인 앱 기능 처리 (데이터, 로직, 구조, 백엔드 등)

Jetpack은 "UI + 비 UI 계층 전체를 관리하는 Android 개발 표준"이며, Compose는 그중 "UI 구성 방법"만 대체한 모듈입니다.



### 결론 요약

- Compose UI를 사용한다고 해서 Jetpack을 안 쓰는 게 아님
- 실제로는 Compose + ViewModel + Room + Navigation + DataStore 등이 기본 조합
- 앱이 복잡해질수록 Jetpack 라이브러리의 사용은 거의 필수

원하시면 이 내용을 기반으로 한 Compose + Jetpack 구조도, 실제 예제 앱 구조 파일, 또는 슬라이드 형태로 요약해 드릴 수 있습니다.

첨부하신 PDF \*\*「12 머티리얼 라이브러리」\*\*는 기존 **View** 기반 안드로이드 **UI** 개발에 초점을 맞추고 있습니다. 하지만 **Compose UI** 수업에 활용하려면 다음 두 가지 작업이 필요합니다:

# ✓ 1. 정리 대상 (Compose와 관련 없는 내용)

Compose에서는 기존의 XML 레이아웃이나 뷰 계층(View hierarchy)을 사용하지 않기 때문에, 아래 항목들은 수업 자료에서 제거하거나 Compose 대응으로 대체해야 합니다.

제거 대상 이유

AppBarLayout, 모두 View 기반 레이아웃이며 Compose에서는 CollapsingToolbarLayout, TopAppBar, LazyColumn, scrollBehavior 등으로 대체

TabLayout, TabItem, ViewPager2, Compose에서는 TabRow, ScrollableTabRow, TabLayoutMediator Pager 등으로 대체

DrawerLayout, NavigationView, Compose에서는 ModalDrawer, DrawerState, menu.xml rememberDrawerState 등 사용

컴포저블 사용

activity\_main.xml, Compose에서는 XML 없이 setContent {}

MainActivity.kt에서 블록 내부에 컴포저블 선언

# ☑ 2. Compose 기반으로 바꿔서 설명할 수 있는 내용 정리

아래는 PDF에서 설명하는 각 항목을 Jetpack Compose에 맞게 다시 설명한 버전입니다:

### 📌 12-1. 앱바 사용하기 (Compose 방식)

setContentView, findViewById 등

NestedScrollView

정의 방식)

View 방식 Compose 방식

AppBarLayout TopAppBar() 또는 CenterAlignedTopAppBar()

CollapsingToolbarLayout TopAppBarScrollBehavior()+

LargeTopAppBar() + nestedScroll()

NestedScrollView + LazvColumn +

CoordinatorLayout nestedScroll(scrollBehavior.nestedScrollCo

nnection)

툴바 title, 스크롤 flag 설정 TopAppBar(title = { Text("제목") },

scrollBehavior = ...)

📌 예시 코드:

```
val scrollBehavior = TopAppBarDefaults.enterAlwaysScrollBehavior()
Scaffold(
  topBar = {
    LargeTopAppBar(
      title = { Text("Compose 앱바") },
      scrollBehavior = scrollBehavior
  },
  modifier = Modifier.nestedScroll(scrollBehavior.nestedScrollConnection)
  LazyColumn(contentPadding = it) {
    items(100) { i ->
       Text("아이템 $i", modifier = Modifier.padding(16.dp))
  }
}
📌 12-2. 탭 레이아웃 구성 (Compose 방식)
    View 방식
                                             Compose 방식
TabLayout,
                  TabRow, ScrollableTabRow
 TabItem
 ViewPager2
                  HorizontalPager() (accompanist-pager 라이브러리 또는 Compose
                  Pager API)
                  직접 상태로 연결 (selectedTabIndex & onClick)
 TabLayoutMed
 iator
📌 예시 코드:
val tabs = listOf("홈", "설정", "정보")
var selectedTab by remember { mutableStateOf(0) }
Column {
  TabRow(selectedTabIndex = selectedTab) {
    tabs.forEachIndexed { index, title ->
      Tab(
         selected = selectedTab == index,
         onClick = { selectedTab = index },
         text = { Text(title) }
    }
  when (selectedTab) {
    0 -> Text("홈 콘텐츠")
    1 -> Text("설정 콘텐츠")
    2 -> Text("정보 콘텐츠")
```

}

```
View 방식
```

#### Compose 방식

```
DrawerLayout,
                            ModalNavigationDrawer()
 NavigationView
 메뉴 xml 설정
                            직접 컴포저블로 구성하거나 navController로 라우팅 처리
📌 예시 코드:
val drawerState = rememberDrawerState(DrawerValue.Closed)
val scope = rememberCoroutineScope()
ModalNavigationDrawer(
  drawerState = drawerState,
  drawerContent = {
    Column {
       Text("训 뉴 1", modifier = Modifier.clickable { /* TODO */ }.padding(16.dp))
       Text("메뉴2", modifier = Modifier.clickable { /* TODO */ }.padding(16.dp))
  }
  Scaffold(
    topBar = {
       TopAppBar(
         title = { Text("드로어 예시") },
         navigationIcon = {
           IconButton(onClick = {
              scope.launch { drawerState.open() }
           }){
              lcon(lcons.Default.Menu, contentDescription = null)
      )
    }
  ) {
    Text("메인 콘텐츠", modifier = Modifier.padding(it))
}
```

### 📌 12-4. 확장된 FAB (Compose 방식)

View 방식

Compose 방식

```
FloatingActionButton + XML 속성 FloatingActionButton(onClick = ...)
확장형 FAB ExtendedFloatingActionButton( ) 사용 가능

에서 코드:

ExtendedFloatingActionButton( onClick = { /* TODO */ }, icon = { Icon(Icons.Default.Add, contentDescription = null) }, text = { Text("추가") }
```

### 📘 결론 및 수업 방향 제안

- 기존 PDF의 내용을 Compose UI 구성요소 중심으로 재구성하는 것이 효과적입니다.
- 각 항목을 Compose에서 어떻게 구성하는지 코드 예제와 함께 소개하는 슬라이드나 핸드아웃을 만들면 좋습니다.
- View 기반 XML은 설명 대상이 아니라 "이제 Compose에서는 이렇게 바뀌었습니다"의 비교 대상으로만 제시하세요.

원하신다면 위 내용을 기반으로 **Compose** 수업용 마크다운(**Marp** 형식) 슬라이드나 정리 문서도 만들어드릴 수 있습니다. 필요하신 형식을 말씀해 주세요!

첨부하신 PDF \*\*「13 액티비티 컴포넌트」\*\*는 기존 View 기반 Android 앱 개발을 설명하고 있으며, Jetpack Compose UI 수업용으로는 일부 내용을 수정하고 Compose 방식으로 재설명할 필요가 있습니다. 아래에 따라 정리 및 변환을 제안드립니다.

### ✓ 1. Compose UI 수업 기준으로 제거 또는 보완할 항목

기존 항목

Activity)

Compose UI 기준에서의 처리 방향

startActivity(), Intent 유지하되 Compose NavController와 함께 설명 (단방향 관련 코드 이동만 설명할 수도 있음) ActivityResultLauncher, Compose 기반 Navigation에서는 대부분 navBackStackEntry.arguments 또는 ViewModel로 Intent.putExtra 등 상태 전달 layout.xml. Compose에서는 XML 없이 Composable 함수로 UI 구성. ViewBinding 사용 안함 AddActivity.kt, MainActivity.kt Bundle. Compose에서는 rememberSaveable()로 대체 onSaveInstanceState() ANR, Handler, Thread 예제 유지 가능. 하지만 CoroutineScope와 LaunchedEffect로 다시 설명 Compose에서는 setContent {} 기반으로 Activity 생성 설명 (Empty

# ☑ 2. Compose UI 방식으로 다시 설명할 내용 (2025.1.1 + targetSdk=35 기준)

MainActivity 하나로 시작

```
📌 [13-1] 인텐트와 화면 전환: Compose 기준
```

기존 Intent를 사용한 화면 이동은 유지하되, Compose에서는 Navigation 라이브러리를 쓰는 것이 권장됩니다.

```
● 여시: Compose Navigation 화면 전환

②Composable
fun AppNavHost(navController: NavHostController) {
    NavHost(navController, startDestination = "home") {
        composable("home") { HomeScreen(navController) }
        composable("add") { AddTodoScreen(navController) }
    }
}

// HomeScreen
Button(onClick = { navController.navigate("add") }) {
    Text("할 일 추가")
}
```

☑ 데이터 전달: Bundle → navArguments

```
composable("detail/{id}") { backStackEntry ->
  val id = backStackEntry.arguments?.getString("id")
📌 [13-2] 액티비티 생명주기와 상태 저장
    Compose는 Lifecycle0wner를 내장하고 있으므로, 생명주기 직접 다룰 일은 줄었지만 상태
    저장은 중요합니다.
✓ View 방식
override fun onSaveInstanceState(outState: Bundle) {
  outState.putInt("count", count)
✓ Compose 방식
var count by rememberSaveable { mutableStateOf(0) }
📌 [13-3] ANR 방지와 코루틴
    Compose에서는 LaunchedEffect, rememberCoroutineScope를 통해 자연스럽게 코루틴을
    사용합니다.
☑ 잘못된 방식 (UI Thread에서 딜레이)
Thread.sleep(5000) // ANR 위험

▼ Compose + 코루틴 방식

val scope = rememberCoroutineScope()
Button(onClick = {
  scope.launch {
   delay(5000)
   // 작업 완료 후 UI 업데이트
}){
  Text("5초 대기")
또는 생명주기에 안전하게 작동하는 LaunchedEffect 사용:
LaunchedEffect(Unit) {
  val result = fetchData()
📌 [13-4] 할 일 목록 앱: Compose 재구성 예시
```

기존의 MainActivity.kt, AddActivity.kt, layout.xml 파일들을 모두 Compose UI로 통합

☑ 구조 제안

- 메인화면: 등록된 할 일 목록 LazyColumn
- 할 일 추가 화면: TextField, Button
- Navigation으로 이동
- 상태 저장은 ViewModel 또는 rememberSaveable

```
☑ 간단 예시
@Composable
fun TodoListApp() {
  val navController = rememberNavController()
  AppNavHost(navController)
}
@Composable
fun TodoListScreen(todoltems: List<String>, navController: NavController) {
  LazyColumn {
    items(todoltems) { item ->
       Text(text = item)
  }
  FloatingActionButton(onClick = { navController.navigate("add") }) {
    lcon(lcons.Default.Add, contentDescription = null)
}
@Composable
fun AddTodoScreen(navController: NavController) {
  var text by rememberSaveable { mutableStateOf("") }
  Column {
    TextField(value = text, onValueChange = { text = it })
    Button(onClick = {
       // 상태에 저장하고 뒤로가기
       navController.popBackStack()
    }) {
       Text("저장")
    }
  }
}
```

### 🔽 결론 및 PDF 수정 방향 제안

- 1. 다음 항목 제거 또는 요약 처리:
  - ViewBinding, layout.xml
  - Intent 중심 액티비티 이동 (Compose Navigation으로 대체)
  - o onSaveInstanceState() 설명 (→ rememberSaveable)
  - ANR 해결을 위한 Thread, Handler → CoroutineScope, LaunchedEffect

- 2. Compose에서 동일 기능을 구현하는 방식으로 재작성:
  - Activity 간 전환 → NavHost, NavController
  - 상태 저장 → rememberSaveable, ViewModel
  - UI 구성 → XML 제거, Composable 함수 기반 구성
- 3. 실습 중심 재구성 제안:
  - 기존 할 일 앱을 **Compose** 기반으로 재작성 실습
  - 단계를 다음과 같이 정리:
    - 1. Compose Navigation 설정
    - 2. TodoListScreen, AddTodoScreen 작성
    - 3. 상태 저장과 목록 관리
    - 4. remember, LaunchedEffect, CoroutineScope 실습

#### 원하신다면 위 내용을 기반으로:

- PDF 수정본 (Compose 중심)
- 제 Marp 마크다운 슬라이드
- 👰 수업 교안 문서 (.docx/.md)
- ■ 실습 프로젝트 템플릿

형태로 제작해드릴 수 있습니다. 어떤 형식으로 드릴까요?

첨부하신 PDF 「14 15 16 브로드캐스트리시버 컴포넌트」 문서는 View 기반 Android 컴포넌트 교육용이지만, Android Studio 2025.1.1 + targetSDK 35 기준 Jetpack Compose UI 수업에 맞게 정리 및 변환할 수 있습니다.

### 🔽 1. Compose UI와 무관하거나 변경이 필요한 내용 요약

항목 처리 방안

BroadcastReceiver, Service, Compose UI 수업에서는 컴포넌트의 기본 구조는 설명하되 UI는 Compose로 대체 ContentProvider의 선언 방식 layout.xml, ViewBinding 기반 UI 모두 제거. @Composable 함수로 대체 구성 NavController 사용, Lifecycle-aware startActivity, Composition 구조와 통합 필요 registerReceiver, bindService 등 직접 호출 코드 파일 공유 및 권한 처리, FileProvider Compose에서도 동일하게 필요하므로 구조 설명 설정 유지 (단, UI는 Compose로) 실습 앱 구성 (e.g., MP3 앱, 갤러리 UI는 Compose, 백엔드는 기존 로직 유지 (필요 시 연동 앱) ViewModel 도입)

# ☑ 2. Compose UI 기준으로 재구성한 각 장 핵심 내용

- ▋ 14장: BroadcastReceiver와 Compose
- 💡 Compose 기준 개념
  - BroadcastReceiver는 UI와 직접 연결되지 않음
  - Compose에서는 LaunchedEffect, rememberUpdatedState를 통해 상태 변화 감지 처리

```
Compose 예시: 배터리 상태 감지 UI
@Composable
fun BatteryStatusReceiver() {
  val context = LocalContext.current
  val batteryStatus = remember { mutableStateOf("Unknown") }

DisposableEffect(Unit) {
  val receiver = object : BroadcastReceiver() {
    override fun onReceive(ctx: Context, intent: Intent) {
     val status = intent.getIntExtra(BatteryManager.EXTRA_STATUS, -1)
     batteryStatus.value = when (status) {
        BatteryManager.BATTERY_STATUS_CHARGING -> "충전 중"
        BatteryManager.BATTERY_STATUS_FULL -> "충전 완료"
        else -> "대기 중"
    }
    }
    }
}
```

```
val filter = IntentFilter(Intent.ACTION_BATTERY_CHANGED)
context.registerReceiver(receiver, filter)

onDispose {
    context.unregisterReceiver(receiver)
    }
}

Text("배터리 상태: ${batteryStatus.value}")
}
```

#### 15장: Service + JobScheduler와 Compose

- ♀ Compose 기준 개념
  - Service와 JobScheduler는 Compose UI 외부에서 동작
  - UI는 Compose로, 작업은 백그라운드로 분리 필요
  - 상태 연결은 ViewModel + StateFlow 또는 rememberCoroutineScope로 구현

```
● Composable
fun Mp3PlayerUI() {
    val context = LocalContext.current
    val serviceIntent = remember { Intent(context, MusicService::class.java) }

Row {
    Button(onClick = { context.startService(serviceIntent) }) {
        Text("재생")
    }
    Spacer(modifier = Modifier.width(8.dp))
    Button(onClick = { context.stopService(serviceIntent) }) {
        Text("정지")
    }
}
```

### 📘 16장: ContentProvider와 Compose

- ♀ Compose 기준 개념
  - 콘텐츠 프로바이더는 여전히 ContentResolver로 사용
  - 사진/연락처 선택은 ActivityResultLauncher 또는 rememberLauncherForActivityResult 사용

```
☑ 예시: 갤러리에서 이미지 선택
@Composable
fun ImagePicker(
   onImageSelected: (Uri?) -> Unit
) {
   val launcher = rememberLauncherForActivityResult(
   contract = ActivityResultContracts.GetContent()
```

```
) { uri: Uri? ->
    onImageSelected(uri)
  Button(onClick = { launcher.launch("image/*") }) {
    Text("갤러리에서 이미지 선택")
}
☑ 예시:이미지 표시
@Composable
fun ShowSelectedImage(uri: Uri?) {
  if (uri != null) {
    Image(
      painter = rememberAsyncImagePainter(uri),
      contentDescription = "선택된 이미지",
      modifier = Modifier.size(200.dp)
  } else {
    Text("이미지가 선택되지 않았습니다")
}
```

# ☑ 3. 정리된 Compose UI 수업용 슬라이드/자료 구조 예시

개편 목차 제안

장

Compose 중심 주제

- 14장 Compose에서 시스템 이벤트 수신 (BroadcastReceiver 대체 패턴)
- 15장 Compose UI + 백그라운드 서비스 연동 (MP3 재생, JobScheduler)
- 16장 콘텐츠 프로바이더 연동 (갤러리/카메라/주소록) UI 구성

### ☑ 다음 단계 제안

원하시면 다음 중 하나를 제작해 드릴 수 있습니다:

- 📄 수정된 PDF: Compose UI 코드와 설명으로 재작성
- 📝 Marp 슬라이드: 수업용 마크다운 형식 슬라이드 자료
- ■ 실습 템플릿: Compose 기반 모듈 코드 제공 (Ch14~Ch16)

어떤 형식으로 도와드릴까요?

첨부하신 PDF 「17 저장소에 데이터 보관하기」 문서는 Android 앱의 데이터 저장 방식을 설명하고 있으며, 전통적인 View + XML 기반 설명이 주를 이룹니다. 이를 Android Studio 2025.1.1 + targetSDK=35 + Jetpack Compose UI 수업용으로 바꾸기 위해 다음과 같이 정리해드릴 수 있습니다.

### 🔽 1. Compose UI 수업 기준으로 제거 또는 대체해야 할 항목

제거/축소 대상

설명

ViewBinding, layout.xml,
activity\_setting.xml

Compose는 XML 없이 @Composable 함수 기반

PreferenceScreen XML 기반 설정 Compose에서는 AndroidX Preference는 지원하되 UI는 Compose로 따로 구현 가능

Activity.kt 중심의 findViewById,Intent **UI** 연동 Compose Navigation과 상태 기반 컴포저블 UI로 대체

SQLiteOpenHelper 중심 예제

Room 사용 권장, Compose에서는 ViewModel과 함께 상태 관리 필요

# 🔽 2. Compose 기준으로 재구성한 핵심 설명

[17-1] 데이터베이스에 보관하기 (SQLite → Room + ViewModel)

#### ☑ 기존 방식:

- SQLiteDatabase, SQLiteOpenHelper 从용
- Cursor 수동 이동, 쿼리 직접 작성

#### ✓ Compose 수업 기준:

- Room + ViewModel + LiveData/Flow 사용
- UI는 상태 기반으로 자동 업데이트
- ♀ 예시 구성
  - TodoEntity.kt

```
@Entity(tableName = "todo")
data class TodoEntity(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val content: String,
    val timestamp: Long
)
```

```
2. TodoDao.kt
```

```
@Dao
interface TodoDao {
  @Insert fun insert(todo: TodoEntity)
  @Query("SELECT * FROM todo ORDER BY timestamp DESC")
  fun getAll(): Flow<List<TodoEntity>>
}
   TodoDatabase.kt
@Database(entities = [TodoEntity::class], version = 1)
abstract class TodoDatabase : RoomDatabase() {
  abstract fun todoDao(): TodoDao
}
   4. TodoViewModel.kt
class TodoViewModel(application: Application): AndroidViewModel(application) {
  private val db = Room.databaseBuilder(
     application, TodoDatabase::class.java, "todo.db"
  ).build()
  val todos = db.todoDao().getAll().stateIn(viewModelScope, SharingStarted.Eagerly, emptyList())
  fun addTodo(content: String) {
    viewModelScope.launch {
       db.todoDao().insert(TodoEntity(content = content, timestamp = System.currentTimeMillis()))
    }
  }
}
   @Composable UI
@Composable
fun TodoListScreen(viewModel: TodoViewModel = viewModel()) {
  val todos by viewModel.todos.collectAsState()
  Column {
    LazyColumn {
       items(todos) { todo ->
         Text(todo.content)
       }
    }
    var newTodo by remember { mutableStateOf("") }
    TextField(value = newTodo, onValueChange = { newTodo = it })
    Button(onClick = { viewModel.addTodo(newTodo); newTodo = "" }) {
       Text("추가")
  }
}
```

```
▼ Compose 수업 기준 핵심만 유지:

     내부 저장소: context.openFileOutput() 활용
     외부 저장소: getExternalFilesDir() + 권한 처리
💡 Compose에서 내부 파일 저장 예시
@Composable
fun FileWriteScreen() {
  val context = LocalContext.current
  var text by remember { mutableStateOf("") }
  Column {
    TextField(value = text, onValueChange = { text = it }, label = { Text("입력") })
    Button(onClick = {
      context.openFileOutput("memo.txt", Context.MODE_PRIVATE).use {
        it.write(text.toByteArray())
    }){
      Text("파일 저장")
    }
  }
📌 [17-3] SharedPreferences → DataStore 권장
     SharedPreferences는 Compose와 잘 통합되지 않음. Jetpack DataStore(Kotlin 기반)를 추천.
💡 예시: Preferences DataStore 사용

    SettingsDataStore.kt

val Context.settingsDataStore: DataStore<Preferences> by preferencesDataStore(name = "settings")
val DARK_MODE = booleanPreferencesKey("dark_mode")
   2. 사용 예시 (Compose 내부)
val context = LocalContext.current
val dataStore = context.settingsDataStore
val darkMode by dataStore.data.map { it[DARK_MODE] ?: false }.collectAsState(initial = false)
Switch(
  checked = darkMode,
  onCheckedChange = {
    coroutineScope.launch {
      dataStore.edit { it[DARK_MODE] = it }
```

)

구성 요소 Compose 수업 기준 구성 방식

AddActivity.kt Composable AddTodoScreen()

SettingActivity.kt, Composable SettingScreen() +

settings.xml DataStore

MyAdapter.kt LazyColumn with items()

메뉴/프래그먼트/프리퍼런스 XML 제거하고 상태 기반 UI로 전환

# ✓ 4. 요약 정리

영역	View 기반 설명	Compose 수업 기준 변환	

데이터베이 SQLiteOpenHelper + Cursor Room + ViewModel + Flow

파일 저장 FileWriter, FileReader 내부/외부 저장소 접근 유지

설정 저장 SharedPreferences + Preference XML DataStore + Composable UI

UI 구성 XML, findViewById @Composable, 상태 기반 구성

앱 구조 Activity + layout Compose Navigation + 상태

### ☑ 다음 단계 안내

원하신다면 아래 포맷으로 결과물을 만들어 드릴 수 있습니다:

- 📄 수정된 PDF (Compose 방식으로 재작성된 전체 교재)
- **I** Compose UI 수업용 슬라이드 (Marp 마크다운)
- ■ 실습 프로젝트 템플릿 (Ch17\_ComposeTodo)

필요한 자료 형식을 알려주세요!

첨부하신 PDF \*\*「18·19 네트워크 프로그래밍」\*\*은 View 기반 중심의 Android 교육 콘텐츠로 구성되어 있습니다. 이를 Android Studio 2025.1.1 + targetSDK=35 기준 Jetpack Compose UI 수업용으로 사용하기 위해 다음과 같이 \*\*정리(불필요한 부분 제거)\*\*하고, Compose 방식으로 재해석한 내용을 제시합니다.

### 🔽 1. Compose UI 수업에 맞게 제거/수정할 내용 요약

기존 설명 요소

Compose 수업에서의 처리 방식

Volley, NetworkImageView,
RecyclerView + Adapter,
layout.xml

Compose에서는 사용하지 않음. Retrofit + Coil/Glide + LazyColumn으로 대체

ViewBinding, findViewById,
Fragment

Compose에서는 모두 제거. @Composable 함수로 UI 직접 구성

GoogleMapFragment, 레이아웃 XML, onMapReady() 중심 지도 처리

Compose에서는 <u>Maps Compose API</u> 사용

LocationManager, TelephonyManager 등의 콜백 리스너 기반 상태 감지 Compose에서는 LaunchedEffect나 Flow를 활용한 방식으로 처리

# 🛾 2. Compose 수업 기준 재구성 설명 (PDF 각 장별 정리)

- 18장. 네트워크 프로그래밍 (Retrofit + Compose + Coil)
- 🔄 Retrofit 기반 뉴스 앱 예시 (Compose 방식)
  - 1. 모델 클래스

```
data class NewsItem(
val title: String,
val link: String,
val image: String)
```

2. Retrofit 인터페이스

```
interface NewsApi {
    @GET("news")
    suspend fun getNews(): List<NewsItem>
}
```

3. Retrofit 클라이언트

```
val retrofit = Retrofit.Builder()
   .baseUrl("https://example.com/api/")
   .addConverterFactory(GsonConverterFactory.create())
```

```
.build()
val newsApi = retrofit.create(NewsApi::class.java)

4. ViewModel 구성

class NewsViewModel : ViewModel() {
  private val _news = mutableStateOf<List<NewsItem>>(emptyList())
  val news: State<List<NewsItem>> = _news
```

\_news.value = newsApi.getNews()

#### 5. Compose UI

viewModelScope.launch {

init {

}

```
@Composable
fun NewsList(viewModel: NewsViewModel = viewModel()) {
  val news by viewModel.news
  LazyColumn {
    items(news) { item ->
       NewsCard(item)
    }
  }
}
@Composable
fun NewsCard(news: NewsItem) {
  Card(modifier = Modifier.padding(8.dp)) {
     Column {
       Asynclmage(
         model = news.image,
         contentDescription = null,
         modifier = Modifier
            .fillMaxWidth()
            .height(180.dp)
       Text(news.title, style = MaterialTheme.typography.titleMedium)
    }
  }
}
```

■ 19장. 위치 정보 + 구글 지도 (Compose + Maps Compose)

Glide도 사용 가능하지만, Coil이 Compose와 통합성이 좋음 (coil-compose).

```
☑ 위치 권한 요청 + 현재 위치 가져오기 (Compose 방식)
@Composable
fun LocationRequester() {
val context = LocalContext.current
```

```
val locationPermission = Manifest.permission.ACCESS_FINE_LOCATION
  val launcher = rememberLauncherForActivityResult(
    contract = ActivityResultContracts.RequestPermission(),
    onResult = { granted -> if (granted) { /* 위치 얻기 */ } }
  )
  Button(onClick = { launcher.launch(locationPermission) }) {
    Text("위치 권한 요청")
  }
}
🔽 현재 위치 가져오기 (FusedLocationProviderClient)
@Composable
fun CurrentLocationScreen() {
  val context = LocalContext.current
  val fusedClient = remember { LocationServices.getFusedLocationProviderClient(context) }
  var locationText by remember { mutableStateOf("위치 미확인") }
  LaunchedEffect(Unit) {
    val location = fusedClient.lastLocation.await()
    location?.let {
       locationText = "위도: ${it.latitude}, 경도: ${it.longitude}"
  }
  Text(locationText)
}
     lastLocation.await()는 kotlinx-coroutines-play-services 사용 필요

▼ Google Maps Compose 활용

   1. 의존성 추가
implementation("com.google.maps.android:maps-compose:2.11.4")
   2. 지도 UI 표시
@Composable
fun SimpleMap() {
  val singapore = LatLng(1.35, 103.87)
  GoogleMap(
    modifier = Modifier.fillMaxSize(),
    cameraPositionState = rememberCameraPositionState {
       position = CameraPosition.fromLatLngZoom(singapore, 10f)
  ) {
    Marker(position = singapore, title = "Singapore")
  }
}
```

앱 기능 Compose 구성 방식

뉴스 리스트 Retrofit + ViewModel + LazyColumn

이미지 Coil (AsyncImage)

위치 권한 처리 rememberLauncherForActivityResu

1t

위치 가져오기 FusedLocationProviderClient +

LaunchedEffect

지도 GoogleMap 컴포저블 (Maps Compose)

마커, 중심 이동 Marker(), CameraPositionState 사용

# ✓ 4. 요약: Compose UI 수업 기준으로 변환

주제 View 기반 설명 Compose 기준 대체

네트워크 Volley, JsonObjectRequest Retrofit + ViewModel +

요청 State

이미지 표시 NetworkImageView, Coil AsyncImage()

Glide.into(view)

리스트 UI RecyclerView + Adapter LazyColumn + items()

지도 MapFragment, XML 레이아웃 GoogleMap() 컴포저블

위치 권한 ActivityCompat.requestPermissio rememberLauncherForActivi

tyResult()

위치 LocationManager, LaunchedEffect + await()

FusedLocationProviderClient

### 🔽 다음 작업 제안

필요하신 자료를 아래 중 어떤 형식으로 원하시나요?

- 1. Compose 기준으로 수정된 PDF 문서 (원본 PDF 구조 반영)
- 2. | Compose 수업용 슬라이드(Marp 마크다운)
- 3. 💻 뉴스 앱 & 지도 앱의 Compose 실습 프로젝트 템플릿

필요한 항목과 원하는 스타일을 알려주시면 제작해드리겠습니다.

첨부하신 PDF \*\*「20·21 파이어베이스 연동과 인증 기능」\*\*은 View 기반 안드로이드 앱에 Firebase를 연동하는 내용을 담고 있습니다. 이를 Android Studio 2025.1.1 + targetSdk=35 기준 Jetpack Compose UI 수업에 맞게 활용하기 위해, 다음과 같이 불필요한 View 기반 요소를 정리하고, Compose UI 중심으로 재구성된 설명을 제시합니다.

### 🔽 1. Compose UI 수업 기준으로 정리해야 할 항목

기존 설명 요소

Compose 수업에서의 처리 방식

layout.xml, findViewById, Activity, 전면 제거. Compose의 @Composable UI로 대체

AuthActivity.kt, AddActivity.kt, Composable 함수 + NavHost 기반 화면 구성으로 대체

RecyclerView + Adapter LazyColumn + items()로 대체

Glide + XML ImageView Coil AsyncImage()로 대체

FirebaseUI, firebase-ui-storage 직접 사용보다 Compose-friendly 방식 권장 (downloadUrl → Coil.load)

# 🔽 2. Compose 기준 주요 기능 설명

- 20장: Firebase 인증 기능 (Email/Password + Google Sign-In)
- ₹ 설정 요약
  - google-services.json 앱 루트에 추가
  - build.gradle 구성:

implementation("com.google.firebase:firebase-auth-ktx") implementation("com.google.android.gms:play-services-auth")

```
✔ Firebase Email/Password 로그인: Compose UI 예시
@Composable
fun AuthScreen(viewModel: AuthViewModel = viewModel()) {
var email by remember { mutableStateOf("") }
var password by remember { mutableStateOf("") }
Column(modifier = Modifier.padding(16.dp)) {
TextField(value = email, onValueChange = { email = it }, label = { Text("Email") })
TextField(value = password, onValueChange = { password = it }, label = { Text("Password") })
Button(onClick = { viewModel.signIn(email, password) }) {
Text("로그인")
}
Button(onClick = { viewModel.signUp(email, password) }) {
Text("회원가입")
```

```
}

▼ Firebase 인증 ViewModel

class AuthViewModel : ViewModel() {
  private val auth = Firebase.auth
  fun signIn(email: String, password: String) {
    auth.signInWithEmailAndPassword(email, password)
       .addOnCompleteListener { task ->
         if (task.isSuccessful) {
           Log.d("Auth", "로그인 성공: ${auth.currentUser?.email}")
       }
  }
  fun signUp(email: String, password: String) {
    auth.createUserWithEmailAndPassword(email, password)
       .addOnCompleteListener { task ->
         if (task.isSuccessful) {
           auth.currentUser?.sendEmailVerification()
       }
  }
}
 ▋ 21-1: Firebase Firestore 데이터 저장 및 조회 (Compose)
₹ 설정 요약
implementation("com.google.firebase:firebase-firestore-ktx")
✓ Firestore 사용 예시
data class Message(val text: String = "", val timestamp: Long = 0)
val db = Firebase.firestore
fun addMessage(msg: String) {
  val message = Message(msg, System.currentTimeMillis())
  db.collection("messages").add(message)
}

▼ Compose UI + LazyColumn으로 출력
@Composable
fun MessageListScreen() {
  val messages = remember { mutableStateListOf<Message>() }
  LaunchedEffect(Unit) {
    db.collection("messages")
       .orderBy("timestamp", Query.Direction.DESCENDING)
       .addSnapshotListener { snapshot, ->
         messages.clear()
         snapshot?.toObjects(Message::class.java)?.let { messages.addAll(it) }
       }
  }
```

}

```
LazyColumn {
    items(messages) { msg ->
       Text(msg.text)
  }
}
  ▋21-2: Firebase Storage + 이미지 공유 앱
implementation("com.google.firebase:firebase-storage-ktx")
implementation("io.coil-kt:coil-compose:2.4.0")
☑ 이미지 업로드
fun uploadImage(uri: Uri, onComplete: (String) -> Unit) {
  val storageRef = Firebase.storage.reference.child("images/${UUID.randomUUID()}.jpg")
  storageRef.putFile(uri)
    .continueWithTask { task -> storageRef.downloadUrl }
    .addOnSuccessListener { onComplete(it.toString()) }
}
🔽 이미지 표시 (Coil 사용)
@Composable
fun ImageItem(url: String) {
  Asynclmage(
    model = url,
    contentDescription = null,
    modifier = Modifier
       .fillMaxWidth()
       .height(200.dp),
    contentScale = ContentScale.Crop
  )
}
```

### 21-4: FCM (Firebase Cloud Messaging)

**¾** 설정

- FirebaseMessagingService는 여전히 백그라운드에서 동작 → Compose와는 직접 연관 없음
- 다만 FCM 토큰을 UI에서 확인하는 경우 Compose로 가능

✓ 3. "이미지 공유 앱", "회원가입 앱" 실습 구조 (Compose 기준)

구성 요소 Compose 방식

로그인 UI @Composable

AuthScreen()

인증처리 ViewModel +

FirebaseAuth

데이터 저장 FirebaseFirestore

이미지 FirebaseStorage.pu

업로드 tFile()

이미지 보기 LazyColumn +

AsyncImage

네비게이션 NavController +

NavHost

### 🥻 4. Compose UI 수업용 구성 제안

주제 Compose 기준 핵심 구현

Firebase 연동 google-services.json, build.gradle

구성

인증 기능 Firebase.auth, ViewModel, @Composable

AuthScreen

Firestore 사용 실시간 리스너 + mutableStateListOf()

Storage 사용 URI → putFile() → URL → Coil 이미지

표시

클라우드 FirebaseMessagingService → 로그로

메시징 토큰 확인

앱 구조 MainNavHost, ViewModel, Composable

화면으로 분리

# ☑ 다음 단계 제안

이 자료를 기반으로 다음 중 어떤 자료로 정리해 드릴까요?

- 1. Compose 기반 PDF 개정판 (기존 슬라이드 내용 정리 포함)
- 2. 🤵 Compose UI 수업 슬라이드 (Marp 마크다운)
- 3. 실습 템플릿 프로젝트 (AuthApp, ImageShareApp 등 Compose 기반)

필요한 자료와 선호 형식을 알려주시면 제작해 드리겠습니다!

좋은 질문입니다. navigation-3은 Android Jetpack Compose의 최신 네비게이션 API로, 기존 androidx.navigation:navigation-compose보다 더 명확하고 구조적이며 안전한 화면 전환 구성이가능합니다. 아래에 Compose UI 수업에 적용 시 좋아지는 점과 영향을 받는 챕터들을 정리해드립니다.

# 🔽 navigation-3 적용 시 좋아지는 점

개선점 설명

 ◆ 경로 문자열 대신 함수 기반 "detail/{id}" 같은 경로 문자열 대신 함수 호출로 명시적 선언
 선언 → IDE 자동완성, 경로 오류 방지

◆ 타입 안전한 매개변수 전달 경로 내 파라미터를 함수 인자로 명시함으로써 컴파일

타임 체크 가능

● 명확한 구조의
 NavGraphBuilder.composable()을 중첩 가능 →
 NavGraphBuilder
 하위 화면을 구조적으로 구성 가능

◆ Single Activity App 구조에 Compose 구조와 더 잘 맞고, Fragment 의존 없이 화면 최적화 관리 가능

◆ 경로 분기와 UI 스택 흐름이 복잡한 네비게이션 흐름을 화면 구조 그대로 코드로 선언적으로 표현됨 표현 가능

# ☑ Compose 수업에서 영향을 받는 챕터 정리

영향 내용 Chapter navigation-3 적용 시 이점 14~16 컴포넌트 기존 화면 분기 **UI**를 Receiver 설정 화면, 권한 설정 UI 등 Compose로 대체 시 분리된 화면 전환 개선 (Receiver, Service 등) Navigation 필요 17 저장소에 데이터 예: 설정 화면, 할 일 목록 navGraph { screen("add") { 보관하기 추가 화면 AddScreen() } 형식으로 AddTodoScreen 명확하게 전환 가능

**18~19** 뉴스 상세 화면, 지도 화면 뉴스 리스트 → 뉴스 상세 등 경로 네트워크**/**위치**/**지도 앱 등으로 화면 분기 오류 방지. 인자 전달 안전성 향상

20 Firebase 인증 로그인, 회원가입, screen(Auth) → screen(Home)

메인으로의 전환 흐름 선언으로 NavHost 구조

간결화

21 Firebase이미지 업로드, 이미지이미지 URI 전달 시 타입 안정성Firestore/Storage 앱보기 화면확보. screen("detail",

imageUrl: String) 식 호출 가능

```
기존 navigation-compose 방식
NavHost(navController, startDestination = "home") {
  composable("home") { HomeScreen(navController) }
  composable("detail/{id}") {
    val id = it.arguments?.getString("id") ?: ""
    DetailScreen(id)
}
navigation-3 방식 (권장 방식)
val navController = rememberNavController()
NavHost(navController, startDestination = Destination.Home) {
  screen(Destination.Home) {
    HomeScreen(onItemClick = { id -> navController.navigate(Destination.Detail(id)) })
  screen(Destination.Detail) { backStackEntry ->
    val id = backStackEntry.arguments["id"] as String
    DetailScreen(id)
  }
}
```

# ☑ 수업 적용 요약

적용 영역 기존 방식 navigation-3 적용 시

경로 선언 문자열 직접 작성 타입 안전한 함수 호출 기반 ("screen/{arg}") screen(Destination.Detail(id))

인자 전달 Bundle 기반 수동 처리 매개변수 자동 매핑 가능 (Detail(id: String))

NavGraph 구성 한 곳에 모아서 구성 중첩 및 모듈화된 NavGraph 구성 쉬움

IDE 지원 문자열 오류 없음 IDE 자동완성, 타입 체크 가능



Compose UI 수업에서 navigation-3을 적용하면:

- 학생들에게 명확하고 오류 없는 네비게이션 구성 방식을 가르칠 수 있으며,
- 모든 프로젝트 실습(할 일 목록, 뉴스 앱, Firebase 인증, 지도 앱)에서 화면 간 전환이 구조화됩니다.

원하신다면 각 챕터에서 navigation-3을 적용한 코드 템플릿으로도 바꿔드릴 수 있습니다. 예: Ch17\_AddTodoScreen, Ch20\_AuthFlow, Ch21\_ImageDetailScreen 등.

필요하신 챕터/코드 영역을 알려주시면 구체적으로 변환해드릴게요.