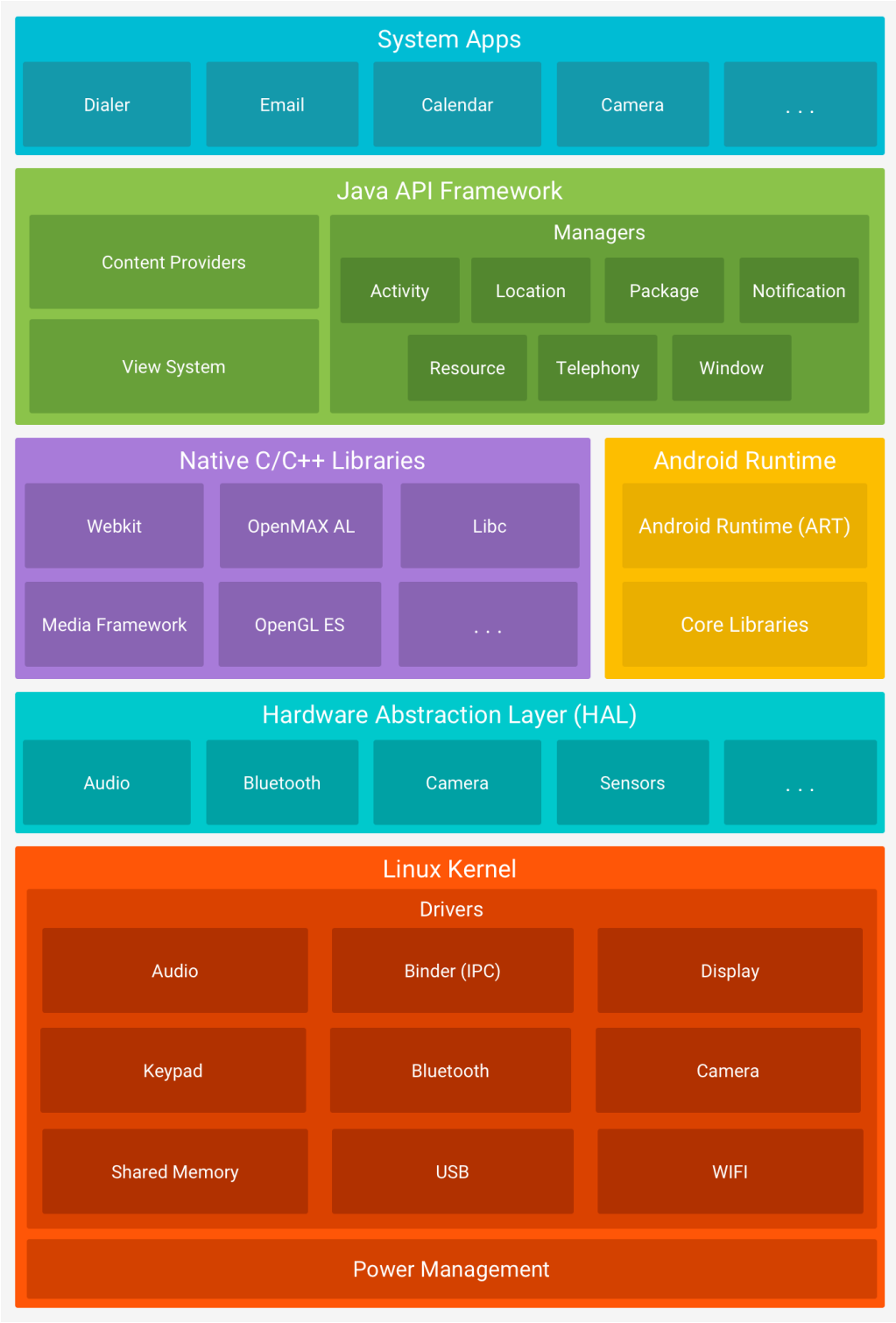
● 안도르이드 특징

장점	단점
라이브러리 코드, 구글의 기본 앱 코드가 오픈돼 있어 앱을 개발할 때 참조할 수 있다.	
	기기의 파편화(fragmentation): 스마트폰 제조사들이 스마트폰을 제작할 때, 구글에서 만든 안드로이드 플랫폼을 그대로 탑제하는 것이 아니라 추가/삭제할 수 있어서, 표준 API로 개발된 코드가 특정 스마트폰에서 수행되지 않거나 에러가 발생할 수 있다. → 테스트의 중요성: 유통되고 있는 대부분 스마트폰에서 테스트하고 차이점을 발견하고 해결해야 한다. 또한 다양한 업체에서 만들어서 크기가 다양할 수 있는데, 하나의 UI 구성 프로그램으로 다양한 스마트폰 크기에 정상적으로 출력되게 신경써야 한다.
멀티태스킹(multitasking): 프로세스가 동시에 동작할 수 있다. 어떤 프로세스가 화면을 점유하여 사용자가 이용하고 있어도, 앱이 백그라운드에서 동작하여 계속 앱을 수행할 수 있다. 또한 기술적으로 iOS에서는 서드 파티 (third party)앱에서 구현할 수 없는 여러가지 기능들을 안드로이드 앱에서는 구현할 수 있다(ex. 걸려오는 전화의 전화번호를 추출할 수 있는 기능 등)	민감한 사용자 정보를 이용하여 악성 앱이 될 수 있다. 그래서 앱을 개발할 땐 많은 주의를 기울여야 한다.

● 안드로이드 플랫폼 아키텍쳐



→ 안드로이드 플랫폼은 리눅수 커널(Linux Kernel)기반이다.

→ HAL(Hardware Abstraction Layer)은 자바 API프레임워크에 하드웨어 기능을 이용하는 표준 인터페이스를 제공한다. 자바 API프레임워크에서 하드웨어 기기(카메라, 블루투스 등)을 이용하기 위한 코드가 실행되면, 내부적으로 HAL의 라이브러리 모듈이 로딩되어 처리한다.

→ 안드로이드 런타임(Android Runtime)은 ART가상 머신을 이용하여 그 위에 일반 애플리케이션 개발 시 이용할 수 있는 자바 API프래임워크를 제공한다.

→ Java API framework: 대부분 앱 개발자들은 이 자바 API프레임워크에서 제공하는 다양한 클래스를 이용해 앱을 개발한다.

▶ 안드로이드 런타임(ART): 자바로 개발된 다든 애플리케이션은 런타임 때 JVM이 수행하지만, 안드로이드의 VM은 ART(Android Runtime)을 이용한다. 자바로 개발된 코드는 컴파일러가 자동으로 DEX파일로 변경하며, 런타임 때 ART가 이 DEX파일을 해석하여 수행하는 구조이다.

Java Source(.java) --Java Complier--> Java Byte Code (.class) --Dex Complier--> Dalvik Byte Code(.dex) --> Dex파일 => ART

▶ 애플리케이션 프레임워크(Java API Framework): 표준 라이브러리로 UI를 구성할 수 있는 View클래스부터 리소스 관리, 데이터 영속화 등의 기능을 제공한다. 이 책의 주 목적인 자바 API Framework를 이해하고 그것이 제공하는 다양한 자바 API를 이용하여 앱을 작성하는 데 있다. 앱 개발자들은 하위의 커널이나 시스템 라이브러리를 직접 이용할 필요 없이, 자바 API 프레임워크에서 제공하는 클래스들을 이용하여 앱의 모든 기능을 구현할 수 있다.

● 컴포넌트 기반 개발

컴포넌트(Component): 안드로이드 앱의 아키텍처에서 가장 큰 특징은 컴포넌트 기반이라는 것이다. 컴포넌트는 앱의 구성 단위이며, 컴포넌트 여러 개를 조합하여 하나의 앱을 만든다. 안드로이드 앱에서 컴포넌트의 물리적인 모습은 클래스이다. 즉, 모든 컴포넌트는 클래스이다. 하지만 모든 클래스가 다 컴포넌트는 아니다. 따라서 안드로이드에서 클래스는 일반 클래스와 컴포넌트로 나뉜다. 둘의 차이점은 클래스의 생명 주기를 누가 관리하는지에 있다. 일반 클래스의 생명주기는 개발자 코드로 관리한다. 필요한 순간 new연산자로 생성해서 이용하고, 필요 없는 순간 null을 대입해서 소멸한다. 개발자 코드에서 직접 생명주기를 관리하는 클래스는 컴포넌트가 아니다. 반면 컴포넌트는 생명주기를 안드로이드 시스템이 생성하여 관리하다가 소멸한다.

컴포넌트 클래스는 독립적인 수행 단위로 동작한다. 이는 '느슨한 결합도'와 관련이 있다. 예를 들어, 일반 클래스에서 A 클래스에서 B클래스를 실행하려면 B b1 = new B();구문으로 객체를 생성해서 실행한다. 이렇게 하면 A클래스가 B클래스에 직접 결합되었다고 표현한다. 하지만 실행하고자 하는 B클래스가 안드로이드 컴포넌트 클래스라면, 직접 개발자 코드로 생성해서 실행할 수 없다. 대신 인텐트(Intent)라는 것을 매개로 하여 결합하지 않은 상태에서 독립적으로 실행한다. A 컴포넌트는 B컴포넌트 실행을 시스템에 의뢰하고 시스템에서 B컴포넌트를 실행하는 식으로, 서로 직접 결합이 발생하지 않는 구조이다.

앱이 컴퓨넌트 기반으로 설계되어 발생하는 부가적인 특징은 다음과 같다.

1. main 함수 같은 애플리케이션의 진입 지점이 따로 없다: 앱의 수행 시점은 다양할 수 있다. 컴포넌트 클래스들은 모두 프로세스가 구동되었을 때 최초로 실행되는 수행 시점이 될 수 있다. 예를 들어 SMS앱 같은 경우 사용자가 아이콘으로 앱을 실행하면 SMS목록 컴포넌트부터 실행되고, SMS가 수신되어 입이 실행되면 SMS수신 컴포넌트부터 실행된다. 이처럼 앱의 수행 시점은 다양할 수 있으며, 이런 이유로 main함수가 없다고 표현한다. 이 모든 것이 안드로이드 컴포넌트가 앱 내에서 각각 독립적인 단위로 수행될 수 있기 때문이다.
2. 애플리케이션 라이브러리 개념이 있다: 안드로이드가 컴포넌트 기반이기 때문에, 외부 앱과 연동할 수 있다. 만약 두 클래스가 직접 결합하여 실행된다면, 자기 앱의 클래스가 아니어서 코드로 실행할 수 없다. 하지만 안드로이드 컴포넌트는 개발자 코드의 결합이 발생하지 않으므로 외부앱의 컴포넌트도 실행할 수 있다. 또한 사용자가 아이콘을 눌러 앱을 실행하지 않아도 앱의 프로세스가 실행될 수 있어서 얼마든지 외부 앱과 연동할 수 있다. 개발자가 만들지 않았는데 그 앱의 기능처럼 느껴진다면, 그것을 활용할 수 있다면, 그것은 라이브러리다. 해서 개발자 입장에서는 스마트폰의 다든 앱들을 라이브러리로 생각할 수 있다. Ex) SMS앱에서 갤러리앱을 실행시켜 사진 목록을 띄워서 사용자가 사진을 하나 선택해 SMS에 포함시킬 수 있는 기능.

정리하자면,

- 안드로이드는 컴포넌트 기반의 개발이다.
- 각 컴포넌트는 개발자 코드 간의 결합이 발생하지 않는다.
- 컴포넌트의 생명주기는 시스템이 관리하므로, 앱 수행 시점은 다양할 수 있다.
- 에플리케이션 라이브러리 개념이 생긴다.

▶ 안드로이드 컴포넌트 종듀

1) 액티비티(Activity): UI를 구성하기 위한 컴포넌트. 사용자 화면을 제공한다. 안드로이드 앱은 클라이언트 측 애플리케이션이므로 화면 구성이 중요하다.
2) 서비스(Service): 화면과 전혀 상관없이 사용자 눈에는 보이지 않지만, 백그라운드에서 장시간 무언가를 수행할 수 있는 컴포넌트다. Ex) 사용자가 화면에서 다른 것을 하고 있더라도, 채팅 앱이 서버랑 계속 연결을 유지한 상태에서 데이터를 주고 받아야 하는데 이럴 때 이용하는 컴포넌트다.
3) 콘텐츠 프로바이더(ContentProvider): 앱 간의 데이터 공유 목적으로 사용하는 컴포넌트. Ex) 앱에서 주소록 데이터가 필요할 때 주소록앱의 데이터를 얻어야 한다. 이때 필요한 컴포넌트.
4) 브로드캐스트 리시버(BroadcastReceiver): 이벤트 모델로 수행되는 컴포넌트. 시스템에서 배터리가 부족하거나, 시스템 부팅이 완료되는 등의 이벤트가 발생했을 때, 이 이벤트를 받기 위해 작성하는 컴포넌트.

▶ 컴포넌트를 이용한 앱 구성: 앱에 어떤 컴포넌트를 몇 개씩 만들어서 작성하는가는 개발자에게 달려 있다. 액티비티 하나로 앱을 개발할 수도 있고, 여러 액티비티를 서비스 하나로 조합해서 개발할 수도 있다. 예를 들면 액티비티2개로 구성된 앱은 화면 2장 정도를 제공하는 앱이고, 액티비티 2개와 서비스 1개로 구성된 앱은 화면과 상관없이 백그라운드에서 계속 수행할 업무를 가지는 앱이고, C는 콘텐츠 프로바이더를 하나 더 가지고 있는데, 이는 자신이 가지고 있는 데이터를 다른 앱에 공개하려는 의도라고 보면 된다. 이처럼 앱의 업무와 화면 설계에 따라 적절하게 컴포넌트를 결정하여 앱을 개발해야 한다. 앱에 최적화된 컴포넌트를 도출하는 게 앱의 효율성 측면에서 중요하다.

● 리소스를 이용한 개발
안드로이드 앱 개발의 또 다른 큰 특징 중 하나가 리소스 외부화를 극대화해서 개발한다는 것이다. 리소스 외부화란, 코드 영역에 누가 언제 실행하든 항상 같은 결과가 나오는 정적인 (static)콘텐츠를 코드에서 분리하여 개발하는 것으로, 프로그램의 유지보수에 도움이 된다. 정적인 부분은 별도의 리소스 파일로 작성하고, 코드에서는 리소스를 얻어서 사용하므로써 유지보수성을 높이자는 개념이다. 이와 같이 하면 코드 영역은 알고리즘 위주의 코드만 남게 되고, 그만큼 코드가 짧아져서 프로그램의 유지보수가 좋아진다는 개념이다. 안드로이드에서 리소스 파일 대부분은 XML이다. 안드로이드에서는 조금만 정적인 부분이라고 하면 자바 코드에서 개발하지 않고, 외부 파일(XML)로 분리해서 개발하는 특징이 있다 (문자열, 크기, 색상, 레이아웃, 메뉴, 에니메이션 등). 그래서 안드로이드 개발을 하다 보면 자바 코드 못지 않기 XML도 많이 작성하게 된다.

Resources: 깡쌤의 안드로이드 프로그래밍