IDE

- ✓ 생산성 향상을 위함
- ✓ 등장
  - CUI 방식이 생산성 높은 개발 환경에 부적합
  - IDE 등장
  - 1990년경 이후 GUI 방식
- ✓ Eclipse, visual studio, Xcode, Intelli IDEA, Android Studio …

Android Studio

- ✓ Google 에서 배포하는 intelli IDEA 에 기반한 Android APP 개발용 공식 IDE
  (Intelli IDEA는 Java 중심으로 많은 프로그래밍 언어에 대응)
- ✓ 툴
  - 에디터(문서 기술) + 컴파일러 + 디버거(오류 원인 규명)
  - 2013 이전
    - Eclipse + Eclipse Android Development Tool + Java 언어 사용
  - 2013 이후
    - Google의 독자적 Android Studio 사용
    - 내장된 Java/Kotlin 컴파일러 이용

개요

- ✓ 2007년 발표한 스마트폰 태블릿용 OS
- ✓ Apple의 iphone, ipad 대응
- ✓ 2013.05.15 → Google I/O 2013서 발표
- ✓ 2014.12.08 → 1.0버전 공개
- ✓ 2016.04.07 → 2.0(안정판) 메이저 버전 업

버전 채널

- ✓ Canary
  - 테스트 중인 최신 릴리즈
  - 최신 기능 시험 → 피드백 제공
  - 개발용 권장 X
- ✓ Dev
  - 사내 테스트에서 거의 합격하여 선택됨
  - 추천 X
- ✓ Beta
  - Stable 채널 공개 전 피드백을 목적으로 릴리즈 되는 버전
- ✓ Stable
  - 안정판
  - 공식 릴리즈 버전

Kotlin

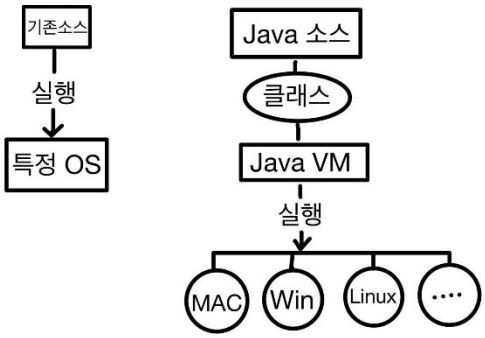
- ✓ 2011 이전
  - Java 중심 Android Studio, Eclipse 환경 주로 이용
  - 7월 등장
  - Java 와 호환성을 가지고
  - 보다 간결하게 코딩 가능
- ✓ 2017.10.
  - Android Studio 3.0 부터 Kotlin 개발용 라이브러리가 기본으로 지원 됨

- ✓ **객체 지향 언어**
  - Java와 닮아 있지만 독자성을 가진 방식
  - JetBrains 에서 배포, 관리
  - Java VM 에서 동작 → Java의 많은 자산 이용 가능

- ✓ **특징**
  - 정형화된 코드 / 보일러플레이트 코드를 경감
  - 에러가 될지도 모르는 코드를 컴파일 단계에서 검출 가능
  - 프로그래밍 실수 회피 기능
  - Java에서 Kotlin 호출 / Kotlin에서 Java 호출 가능

Java VM

- ✓ Java 프로그램을 실행하기 위한 SW
- ✓ 소스 프로그램
  - 컴파일 등의 과정을 거쳐 실행 가능한 프로그램으로 변환 필요
  - Java 등장 전 대다수의 언어
    - 특정 OS에 의존
  - Java 소스 프로그램에 의한 실행 가능 프로그램
    - 특정 OS에 의존하지 않고
    - Linux, MAC, Window 등 서로 다른 OS에서 실행 가능



- ✓ 동작언어
  - JRuby
    - 오브젝트 지향 스크립트 언어의 Ruby를 Java VM에 이식한 언어
  - Groovy
    - Java와 닮은 문법으로 기술 가능한 스크립트 언어
  - Scalar
    - 함수
    - 오브젝트 지향 프로그래밍의 양 특징을 가진 프로그래밍 언어
- ✓ Kotlin → Java 와 비교하면 이해하기 쉬움

Kotlin과 Java의 차이점

- ✓ 변수선언
  - 세미콜론 생략
  - 형 추론 가능 - Java도 10버전부터 형 추론 기능 도입 됨
- ✓ 구문형식
  - Kotlin은 switch 구문 X → when 사용
- ✓ Java의 구문 그대로 사용
  - Java에서 사용되는 package, import 구문 사용 가능
- ✓ 주석문
  - Java에서 사용하는 주석문 그대로 사용 + 중첩 표기 가능 /* ~ /* ~ */ ~ */ 가능

Kotlin 재단

- ✓ 개발처인 JetBrains와 Google은 서로 Kotlin 개발의 보호, 추진, 발전을 목표로 재단 설립
- ✓ Apache 2.0 라이선스에 기반한 오픈소스
- ✓ 공식버전 변경, 복사, 배포 가능
- ✓ 주요기능
  - 프로젝트에 관련된 상표 보유
  - 주도적 언어 디자이너의 사명
  - 호환성 없는 변경의 관리

Android Studio

- ✓ 다양한 OS에서 이용 가능
- ✓ 주요기능
  - 인텔리전트 코드 에디터
    - 고도의 코드 보완, 리팩토링, 코드 해석 기능 이용 가능 → 생산성 향상
  - Gradle 기반 유연한 빌드 시스템
    - 다양한 Android device 용 APK 작성 가능
  - 고속, 기능이 풍부한 에뮬레이터
    - GPS 위치 정보, 네트워크 지연, 모션 센서, 멀티 터치 등 다양한 HW 대응

- Instant Run 기능 탑재
  - 코드 변경점 등 실행 중의 앱 푸시 가능
  - 앱 재가동 및 APK ReBuild 불필요
- 샘플 예제 및 코드 템플릿 이용 가능
- 테스트 툴 및 프레임 워크 이용 가능
  - Junit 이라는 프레임워크 제공 → 앱 테스트 가능
- 소스코드 관리 용이
  - Git / Subversion 등 이용 가능

✓ 각 플랫폼에 대응하는 패키지 존재

✓ 공통 조건

- 3GB 이상 메모리 (8GB 권장) + Android 에뮬레이터에 1GB 필요
- 2GB 이상 HDD 빈공간 필요 (4GB 권장)
- 1280 * 800 이상 해상도

✓ **툴바**

- 

– Run ‘app’ (Shift + F10)
-  – Debug ‘app’ (Shift + F9)
-  – Project Structure (Ctrl + Alt + Shift + s)
-  – Sync Project with Gradle Files
-  – AVD Manager
-  – SDK Manager

✓ **가상 디바이스**

- 대표적인 Nexus 시리즈
- Intel Atom(x86) → 인텔 CPU HW 가속 기능 (HAXM) 사용 / ARM
- 해상도
  - WVGA 800 (480 * 800) / WVGA 854 (480 * 854) 중 선택

개발 흐름

- ✓ 프로젝트 작성 (구성)
  - 타깃 디바이스에 적합한 기본 환경 구성 정보 기반 프로젝트 (공간) 구축
- ✓ 애플리케이션 작성 (소스코딩)
  - 다양한 스튜디오의 기능을 사용해 효율적인 앱 구축
- ✓ 빌드
  - APK 패키지로 빌드하고 에뮬레이터 / Android 탑재 디바이스에서 실행
- ✓ 디버그 / 테스트 수행
  - 앱이 정상적으로 동작하는지 디버깅
  - 앱의 비정상 상태 검출을 위한 테스트 기능 활용
- ✓ APP 공개
  - 몇 가지 절차를 거쳐 사용자에게 공개
  - (마켓 플레이스, 사내 비공개 등)

폴더 구성

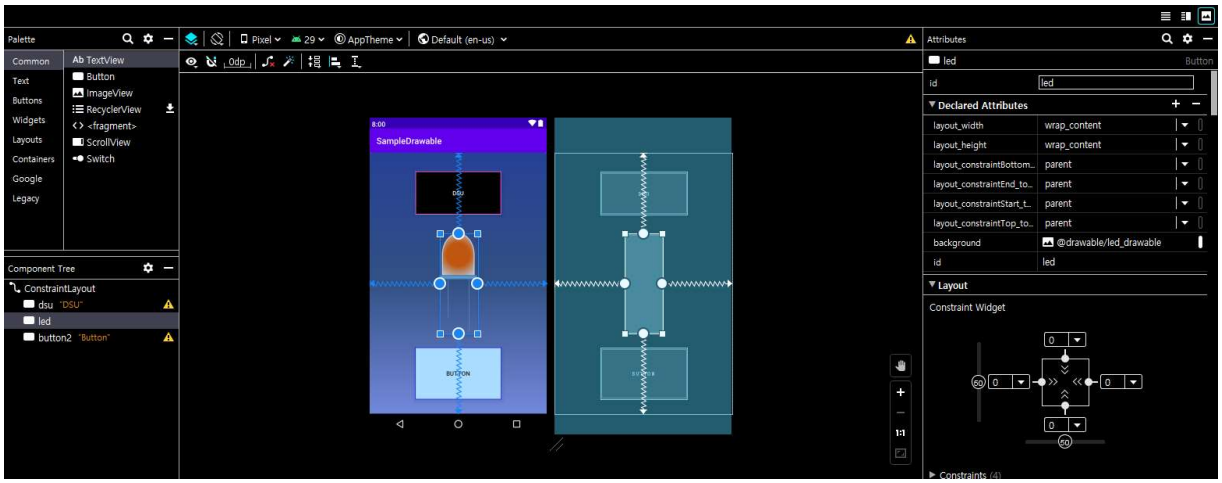
- ✓ bin - 실행파일을 포함하여 다양한 프로그램 존재
- ✓ gradle - 빌드 툴 Gradle에 관련된 프로그램 존재
- ✓ jre - Java와 관련된 다양한 프로그램 존재
- ✓ lib - 앱 개발에 도움이 되는 다양한 부품 프로그램 등 jar 형식으로 존재
- ✓ license - 연관 툴 및 프로그램 라인선스 등
- ✓ plugins - 기능 확장 프로그램. 플러그 인 등 존재

화면 구성

메뉴바 - 풀 기능 제공

네비게이션 바 - 현재 열린 파일 위치 표시 → 이동, 선택 전환　　툴 바 - 빈도 높은 기능 버튼 형식 제공

→ 툴 윈도우 바 - 클릭 → 툴 윈도우로 전개, 전환 가능



에디터 윈도우 - 소스코드 작성, 편집

툴 윈도우 - 프로젝트, 버전 관리, 로그 감시 등

스테이터스 바 - 경고 메시지, 전반적인 상태 등 표시

Project 구조

- ✓ Manifest 폴더
  - 프로젝트 신규 작성, 파일 편집, 리팩토링, 앱 실행, 디버그 등
  - 스튜디오 풀 기능 제공

- ✓ Java 폴더
  - 사용 빈도가 높은 기능을 버튼 형식으로 제공
- ✓ Res 폴더
  - 현재 열린 파일 위치 표시
  - 프로젝트 내 이동, 파일 선택 전환 등
- ✓ Build 스크립트 관리 폴더
  - 빌드 툴인 Gradle 기반 관리 폴더
  - build, gradle 파일은 프로젝트 관리에 중요 역할

빌드 툴

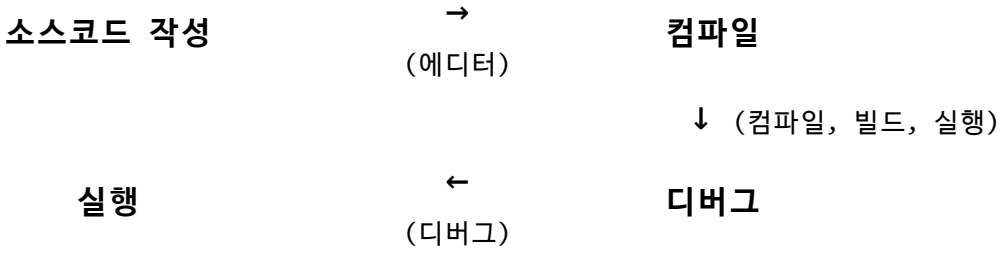
- ✓ Make
  - 빌드 툴의 시초
  - 주로 C언어 개발 등에 사용
- ✓ Ant
  - Java 베이스 빌드 툴
  - xml로 기술
- ✓ Maven
  - Ant의 자손 격
  - Maven부터 빌드에 필요한 라이브러리 자동 취득 가능
- ✓ Gradle
  - Groovy 스크립트 언어로 기술
  - Java, Android 개발에 보급
- ✓ Bazel
  - Google 사내 빌드툴을 오픈 소스로 제공
  - 다양한 언어에 대응, 병렬 처리 가능

Gradle 특징

- ✓ groovy 이용
- ✓ 태스크에 의한 태스크 처리로 불리는 작업 단위로 빌드 처리를 기술
- ✓ 이용되는 라이브러리가 업로드 가능한 repository jCenter 이외에 Maven Central Repository 에도 대응
- ✓ 빌드를 Gradle에 위임하여 처리
  독립해서 실행 가능한 구조로 필요한 경우 커맨드 라인에서 이용 가능
- ✓ 빌드 공정을 자동화하여 관리
  개발자는 거의 인식하지 못하고 사용

프로그래밍 기본 흐름 (시스템 개발 작업 공정)

- ✓ 기본 계획 (요건 정리)
  - 조사, 분석하고 대상이 되는 시스템 기능 정의 (요구사항 분석)
- ✓ 외부 설계 (기능 설계)
  - 사용자 관점에서 시스템의 기능 설계
- ✓ 내부 설계
  - 개발자 입장에서 시스템에 필요한 기능 및 처리, 정의
- ✓ 프로그램 설계
  - 프로그램을 기능 별로 분할하고, 각 프로그램의 구조 설계 수행
- ✓ 프로그래밍
  - 프로그램 설계의 정의를 토대로 프로그램 작성 (코딩)
- ✓ 테스트
  - 시스템이 정상적으로 동작하는지 테스트
- ✓ 운용, 유지 보수
  - 시스템이 안정한 동작인지 모니터링하고 필요에 따라 기능 추가, 변경 수행



컴파일

- ✓ 코딩 처리한 소스 프로그램을 컴퓨터가 실행 가능한 기계어로 번역하는 것
- ✓ Android Studio에서 하나의 컴파일 작업으로 진행되지 않고
  (컴파일 → 빌드 → 실행) 단위로 처리 된다.

디버그

- ✓ Bug
  - 프로그램의 오류 (error, 미스타이핑) / 결함 (defect. fault(결점)와 구분)
- ✓ Debug
  - 버그 제거 작업
- ✓ Android Studio에서 디버그에 도움이 되는 편리한 툴 탑재

디버그 / 테스트

- ✓ 테스팅
  - 프로그램의 결함을 발견하는 작업
    테스터라는 담당자가 수행
  - 원치 않는 동작 / 작동은 됨
- ✓ 디버깅
  - 장애의 원인을 찾고 분석하여 제거하는 것
    프로그래머가 수행
  - 동작하지 않음 / 컴파일 . 빌드 . 실행 안되는 것

SW 개발 방법론

- ✓ 폭포수 모델 (프로그래머) → V자 모델 (프로그래머 + 테스터)
- ✓ 제 3자를 포함한 개발자 단독으로 개발하는 것이 아니라
  테스팅 전문가도 함께 참여하여 개발

빌드

- ✓ 컴파일 + 라이브러리와 링크 + 실행 가능한 파일 작성 등 모든 공정 과정
  *라이브러리 - 프로그램에 필요한 부품군 Github, jCenter 등 다양한 repository 이용, 참조 가능

테스팅

- ✓ 프로그램이 정상적으로 동작하는지 여부를 테스트하는 것
- ✓ 단위 테스트
  - 한 기능에 해당하는 모듈 단위로 하는 테스트
  - Kotlin → 메소드 단위 테스트
  - 클래스 단위 테스트
  - 코딩 직후의 소스코드 테스트
- ✓ 통합 테스트
  - 단위 테스트를 종료한
    복수의 클래스, 프로그램을
    클래스, 기능 구간 단위의 인터페이스를 중심으로 실시하는 테스트
- ✓ 시스템 테스트
  - 인수 전 단계의 환경에서 실시하는 테스트
  - 주로 개발된 시스템이 스펙에 맞춰 동작하는지 여부 확인하는 공정

- ✓ 목적
  - 결함 발견 (동적 테스팅과 정적 테스팅)
  - 결함 예방
  - 품질 수준에 대한 자신감 획득
  - 의사 결정을 위한 정보 제공 (개발 프로세스 등 개선)

리팩토링

- ✓ 현재 동작하고 있는 프로그램의 동작, 기능, 스펙을 유지하거나 내부 구조를 되살펴보는 것
- ✓ 여러 변수를 선언하고 변수명 등을 변경시 프로그램 전체에 영향을 주지 않도록 하기 위함

프로젝트 생성

- ✓ File > New > New Project > 템플릿 선택 > Name, Package name, Save location, Language, Minimum SDK > finish

레이아웃 추가

- ✓ app >> New > Android Resource File > File name, Resource type, Root element, Source set, Directory name > OK

리소스 파일

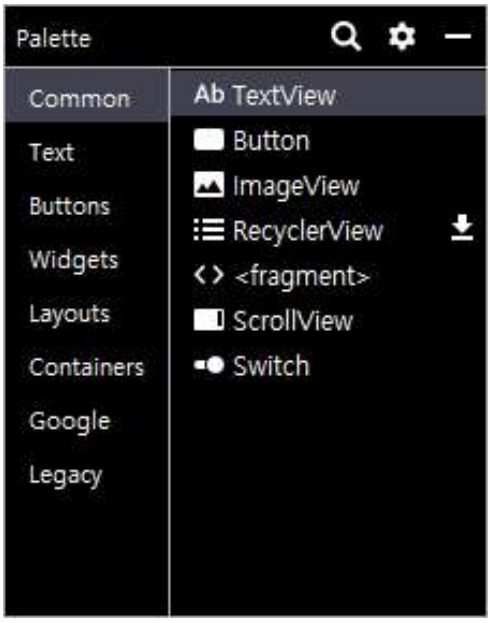
- ✓ xml로 작성된 레이아웃, 이미지, 문자열, 색상 등 설정 파일
- ✓ 관련 코드 별도 저장 → 프로그램에 영향을 주지 않고 설정 변경 가능
- ✓ 리소스는 빌드 과정에서 R 클래스로 생성
- ✓ 생성된 R 클래스의 리소스 id를 사용해 각 리소스에 접근 가능
- ✓ Animator - 프로퍼티 애니메이션
- ✓ Anim - 트윈 애니메이션
- ✓ Color - 색 상태 리스트
- ✓ Drawable - 비트맵 파일
- ✓ Mipmap - 다양한 런처 아이콘 관련 (해상도 대응)
- ✓ Layout - UI 작성을 위한 레이아웃
- ✓ Menu - 옵션, 컨텍스트, 서브 메뉴 등 애플리케이션 메뉴
- ✓ Raw - 미 가공 형식으로 저장된 임의 파일
- ✓ Values - 문자열, 정수, 색 등 단순 값 → 색(16진수)
  - string (문자열), color (색), style (스타일) 은 기본 제공

소스파일 추가

- ✓ Java 폴더 안의 세 패키지 중 첫 번째 >> New > Java Class
  > Kolin File/Class
- ✓ Manifest 파일
  - < activity android:name="가장 먼저 실행될 Java/Kotlin 클래스">
    <intent_filter>
    <action android:name="">
    <category android:name="">
    </intent_filter>
    </activity>

레이아웃 화면 구성

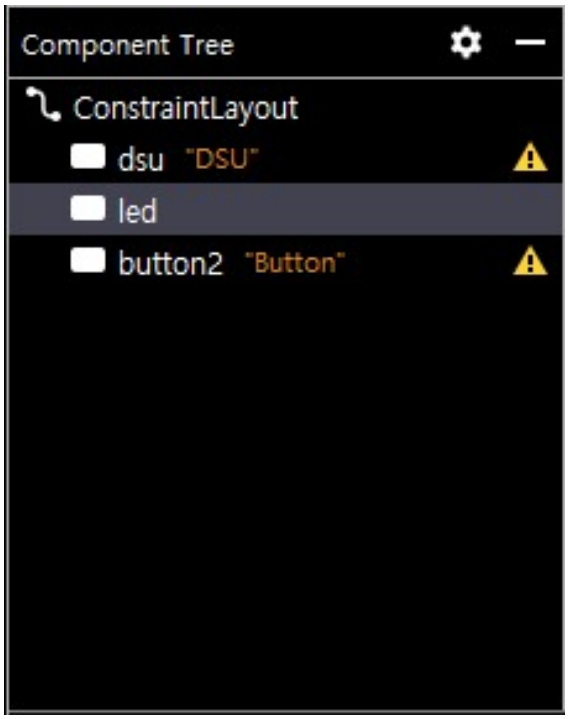
- ✓ **Palette**



  - 레이아웃에서 사용 가능한 요소 리스트
  - 레이아웃에서 사용 가능한 요소(GUI 부품)가 카테고리 별로 표시
  - 요소를 디자인 에디터 및 컴포넌트 트리 영역으로 드래그 앤 드롭 방식으로 레이아웃 작업 가능
  - Comon
    - 일반적으로 자주 사용하는 항목
  - Text
    - 문자 표시.입력, 체크 기능이 있는 문자 입력 등 문자 처리용
    - InputType 으로 결정 가능
  - Buttons
    - 버튼, 체크박스, 라디오버튼 등 선택 가능성 요소

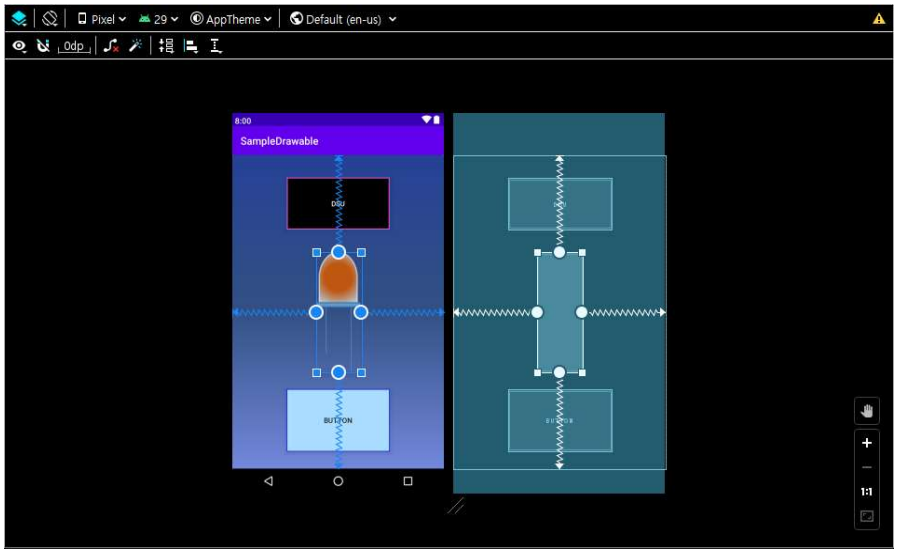
- Widget
  - 이미지, 비디오, 달력, 프로그레스바, 서치바 등
  - Seekbar - 볼륨 조절 / Ratingbar - 별점
- Layouts
  - 배치 목적의 요소 (배치 관리자)
  - Guideline[V/H] / LinearLayout[V/H] / FrameLayout - 구조..잘 사용 X
- Containers
  - 다른 요소들을 담기 위한 컨테이너, 스피너(드랍다운 리스트), 스크롤 뷰, 툴바 등
- Google
  - Google API가 제공하는 배너 광고, 지도 표시 요소
- Legacy
  - 최근에 사용되지 않는 요소
    (GridLayout / ListView) → RecyclerView / RelativeLayout

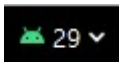
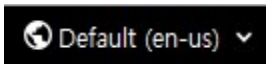
✓ **Component Tree**



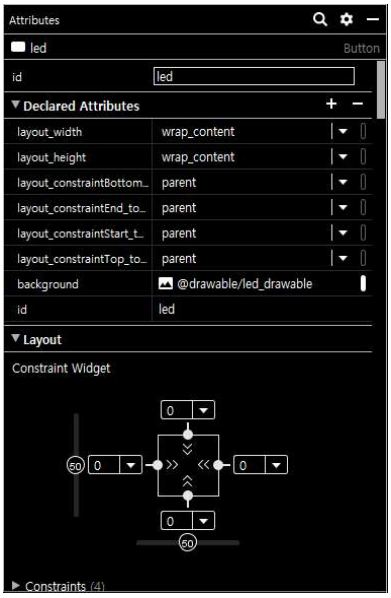
- 레이아웃 구성 요소를 속성 구조로 표시

✓ **View Control**



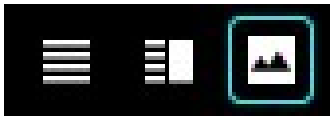
- 툴바 포함하여 레이아웃 외관 설정 및 프로퍼티 편집
- 레이아웃의 외관을 설정하거나 레이아웃의 프로퍼티를 편집하는 용도의 버튼 집합
-  - 디자인 & 블루프린트
-  - 화면 방향 전환 제어
-  - 단말의 타입과 사이즈(해상도) 제어
-  - API Version (안드로이드 버전)
-  - 앱 테마 변경
-  - 문자열에 표시하는 언어
-  - 표시 옵션
-  - LinearLayout 확장 툴 (수직/수평)

✓ **Attributes (속성)**



- 현재 선택된 View의 속성 표시
- 뷰의 종류에 따라 표시되는 속성도 다르므로 주의
- 상황에 맞게 설정
- 필요한 것만 정의하여 설정

✓ **창 전환 버튼**



- 레이아웃 에디터와 텍스트 에디터 창 전환

레이아웃

✓ LinearLayout

- 수직/수평 방향으로 뷰를 선상 배치하는 레이아웃

✓ TableLayout

- 표 형식의 레이아웃
- 각 행을 TableRow 요소로 저장하고 그 안에 1개 이상의 뷰를 포함시키는 것 가능

✓ ConstraintLayout

- 화면 구성을 빠르게 처리하기 위한 레이아웃 (성능↑)
- 뷰를 다른 뷰와의 위치 관계를 정의(제약)하여 배치하는 것이 특징 (비율)
- 퍼포먼스를 높이기 위해 고안된 레이아웃
- 이전 중첩 구조로 표현되던 위치 관계(복잡)
  각각의 뷰 사이에 제약을 정의하여 중첩관계를 사용하지 않고 평이한 배치를 구현

크기 지정 단위

- ✓ dp (density-independent pixel)
  - 해상도 비의존 픽셀
    해상도가 다른 디바이스에서 동일한 비율로 표시되도록 고안한 단위
  - 화면의 물리적인 해상도에 기반을 둔 추상적인 단위
  - 통상의 160dpi 화면 기준 → 높은 해상도
    1dp 그리는데 필요한 픽셀 수 증가
- ✓ sp (scale-independent pixel)
  - 스케일 비의존 픽셀
    화면의 물리적인 해상도와 디바이스의 폰트 크기 지정 시 주로 사용
  - 문자 크기 지정 가능
- ✓ 그 외
  - pt(point) - 72dpi 화면 밀도의 스크린을 가정 → 화면의 물리적인 크기에 기반하여 1/72 인치
  - px(pixel) - 화면 그대로의 픽셀에 대응 → 표시는 디바이스에 의존
  - mm(millimeter) - 화면의 물리 크기에 대응
  - in(inch) - 화면의 물리 크기에 대응

성능을 고려한 레이아웃

- ✓ Measure
  - Component Tree를 근간으로 부모 요소로부터 순서대로, 각 뷰 요소의 크기 결정
  - ViewGroup의 경우, 중첩되어 있는 자식요소의 크기를 계측하고 자신의 크기 결정
- ✓ Layout
  - 다른 부모 요소들로부터 순서대로 컴포넌트 트리로 되돌아가 추정으로 구해진 크기로부터
    각 요소의 위치를 결정
- ✓ Draw
  - 계속하여 부모 요소로부터 순서대로 컴포넌트 트리로 되돌아가
    각 뷰의 그리기 오브젝트를 생성하고 GPU에서 계측된 크기와 그리기 위치로 송신

- ✓ App 디자인 시 물리적 크기 / 해상도 차이 / 사용자 환경 고려

Hardcoded

- ✓ 코드에 포함 시켜 사용자가 직접 변경 불가능한 상태를 의미 → values > string.xml 파일 수정

해상도

- ✓ 화면, 인쇄 등에서 하나의 이미지를 표현하는 척도
- ✓ 주로 몇 개의 픽셀, 도트로 표현되는 것인지 나타내는 정밀도의 지표로 사용
- ✓ ppi (pixels per inch)
  - 1인치당 몇 개의 픽셀로 이루어졌는지 나타냄
- ✓ dpi (dots per inch)
  - 1인치당 몇 개의 점(도트)로 이루어졌는지 나타냄
  - 주로 인쇄물의 점을 해상도로 표현할 때 사용
  - 최근 스마트폰 디자인 시 중요하게 사용
- ✓ dp/dpi
  - 어떤 화면에서도 동일한 크기 표시
  - 화면 밀도가 변해도 비율에 따라 이미지의 픽셀을 조정해
    큰 이질감 없이 동일한 크기의 이미지를 보여줌
- ✓ 비트 해상도 = 픽셀 해상도
- ✓ 이미지 해상도 - 이미지의 픽셀 수 (dpi, ppi)
- ✓ 화면 해상도 - 정밀도 (ppi)
- ✓ 모니터 해상도 - 화면의 픽셀 → 대개 가로*세로 픽셀 수 형태로 나타냄

규격

- ✓ **VGA**
  - IBM 사가 제정한 아날로그 방식의 컴퓨터 디스플레이 규격의 표준
  - 640*480
  - 디스플레이 규격의 기준
- ✓ **HVGA**
  - 표준 VGA를 반으로 접은 해상도
  - 480*320
  - 3:2 비율
  - 초기 스마트폰 해상도
  - iPhone 3GS, 안드로이드 원, 옵티머스 원, 갤럭시 에이스, 옵티머스 시크, 미라크 A 등

- ✓ **WVGA**
  - VGA를 넓게 늘린 해상도
  - 800*480
  - 16:9.6 비율
  - 갤럭시 S를 포함 안드로이드 2.3까지 안드로이드의 표준 규격 해상도
  - 갤럭시 S, 넥서스 원, 넥서스 S, 옵티머스 Q, 옵티머스 2X, 디자이어, 베가레이서, 베가 X 등
- ✓ **FWVGA**
  - 일본에서 쓰던 와이드 해상도 규격
  - 854*480
  - 16:9 비율
  - 동영상 감상에 최적화된 해상도
  - 엑스페리아 X10, 모토로이, 모토쿼티 등
- ✓ **QHD**
  - WVGA 보다 높은 해상도
  - 960*540
  - 16:9 비율
  - 동영상 감상에 최적화된 높은 해상도
  - 갤럭시 R, 테이크 핏, 테이크 야누스, 센세이션, 이보 4G, 레이더 4G, 아트릭스 등
- ✓ **Retina Display**
  - '레티나(Retina)'와 '디스플레이(Display)'라는 단어를 결합하여 만든 용어
  - HVGA에서 가로와 세로를 두 배씩 늘린 해상도
  - 960*640 1136*640
  - 3:2 비율 16:9 비율
  - iPhone 4, iPhone 4S, iPhone 5, iPhone 5S 등
- ✓ **HD**
  - LTE 스마트폰에 출시와 함께 탑재된 높은 해상도
  - 1280*720
  - 16:9 비율
  - 갤럭시 S2, 갤럭시 S3 LTE HD, 베가 레이서 2, 옵티머스 LTE 2 등

- ✓ **WXGA**
  - HD 해상도에서 가로 폭이 넓어진 해상도
  - 1280*800
  - 16:10 비율
  - 아랫부분에 검은 여백의 레터박스가 생기는 해상도
  - 갤럭시 노트, 베가 LTE 등
- ✓ **Full HD**
  - HD 해상도의 하나
  - 1920*1080
  - 16:9 비율
  - WVGA 해상도 이후로 가장 많이 사용되고 있는 해상도
  - 갤럭시 S4, 옵티머스 G 프로, 엑스페리아 Z, 베가 NO.6 등
- ✓ **Retina HD**
  - 1334*750 1920*1080
  - 16:9 비율 16:9 비율
  - iPhone 6, iPhone 6 Plus
- ✓ **QHD**
  - 2560*1440
  - 16:9 비율
  - HD 크기 4배, Full HD 크기 2배
  - 갤럭시 S6, 갤럭시 S6 엣지, 옵티머스 G3, 갤럭시 노트 4, 갤럭시 노트 엣지, 구글 넥서스 6 등

기기특성

- ✓ 화면크기 - 실제 물리적 크기
- ✓ 화면 해상도 - 전체 픽셀의 수
- ✓ 화면 밀도 - 얼마나 많은 픽셀이 존재하는지 나타냄
- ✓ dip는 화면의 크기에 반비례

드로어블 리소스

- ✓ 화면에 그릴 수 있다
- ✓ API를 사용하거나 / 가져오거나 / android:drawable
  android:icon 등의 특성을 사용하여 리소스에 적용
- ✓ 비트맵 그래픽파일 생성
- ✓ 나인 패치 파일 → 이미지 크기 조정 가능
- ✓ 계층 목록, 상태 목록 및 레벨 목록 → 다른 드로어블 배열 관리, 상태 참조, 드로어블의 수 관리
- ✓ 전환 드로어블 → 두 개의 드로어블 간을 크로스 페이드 처리
- ✓ 인셋 드로어블 → 지정된 거리만큼 다른 드로어블 인셋 처리
- ✓ 클립 드로어블 → 현재 레벨 값 기준 다른 드로어블을 클립
- ✓ 배율 조정 드로어블 → 현재 레벨 값 기준 다른 드로어블 크기 변경
- ✓ 세이프 드로어블 → 색상과 그라데이션 포함 기하학적 모양 정의
- ✓ **res/drawable 폴더에 등록**
- ✓ Background 속성 → “@drawable/파일명” 등의 방법으로 사용 가능

상태 드로어블

- ✓ <item
  android:state_pressed="true"
  android:drawable="@drawable/png 파일 1"/> → 위젯을 누르면 png 파일 1이 보인다

  <item
  android:drawable="@drawable/png 파일 2"/> → 위젯을 누르지 않으면
  png 파일 2가 보인다

모양 만들기

- ✓ <shape xmlns:android=”” android:shape=”모양”>

  < size android:width=”가로 크기” android:height=”세로 크기” /> - 크기

  < stroke android:width=”테두리 크기” android:color=”테두리 색” /> - 테두리

  < solid android:color=”배경 색” /> - 채우기

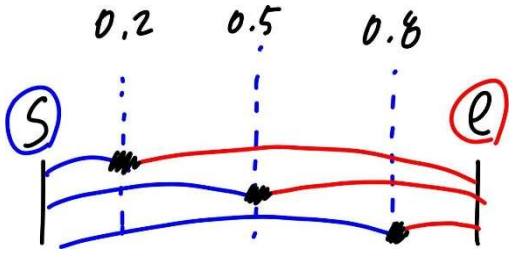
  <padding android:속성=”여백 크기” /> - 내부 여백
  속성 - top/bottom/left/right
  - 여러 개는 따로 지정

  </shape>

그라데이션 배경

✓ <shape xmlns:android="">

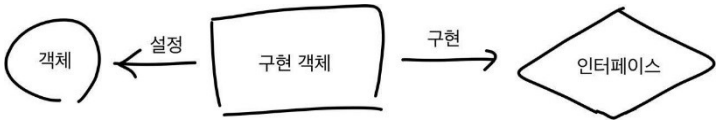
그라데이션
<gradient
android:angle="각도" → 그라데이션 각도 조절
android:startColor="시작 색상"
android:CenterColor="중간 색상"
android:endColor="끝 색상" → 시작 색상에서 중간색상을 거쳐
끝 색상으로 그라데이션
android:CenterY="비율" → Y축을 기준으로 시작 색상으로부터 비율만큼 나눈다
>



모서리 둥글게 (반지름을 반지름으로 갖는 원처럼 깎는다)
< corners android:radius="반지름" />

</shape>

이벤트

✓ 

setOnClickListener() onClick

✓  - 핀치(Zoom)

✓  - 스와이프(Swipe)

✓  - 스크롤(Scroll)

✓  - 회전(Rotate)

- ✓ 터치 이벤트 - 화면을 손가락으로 누를 때 발생하는 이벤트
  - GestureDetector 클래스 사용
    - detector = new GestureDetector(this, new GestureDetector.OnGestureListener(){
      메소드 재정의
      });
  - 오버로딩 메소드
    - onDown - 눌림
    - onShowPress() - 눌렸다 떼어짐
    - onSingleTapUp() - 한 손가락으로 눌렀다 떼어짐
    - onSingleTapConfirmed() - 한 손가락으로 눌림
    - onDoubleTap() - 두 손가락으로 눌림
    - onDoubleTapEvent() - 두 손가락으로 눌려진 상태에서 떼지거나 이동하는 등의 세부 액션
    - onScroll() - 스크롤 - 거리 값이 중요
    - onFling() - 가속도 (갑자기 빠르게 스와이프) - 속도 값이 중요
    - onLongPress() - 오래 누르는
- ✓ 키 이벤트 - 가상 키보드나 하드웨어를 누를 때
  - onKeyDown() / onKey() 메소드 재정의 하여 이용 가능
  - [BACK] - 이전 화면으로 되돌아가기 또는 작업 취소
  - [HOME] - 홈 화면으로 이동 또는 최근 실행된 앱 보기
  - onKeyListener() 인터페이스 구현 대표키
    - KEYCODE_DPAD_LEFT - 왼쪽 화살표
    - KEYCODE_DPAD_RIGHT - 오른쪽 화살표
    - KEYCODE_DPAD_UP - 위쪽 화살표
    - KEYCODE_DPAD_DOWN - 아래쪽 화살표
    - KEYCODE_DPAD_CENTER - [중앙] 버튼
    - KEYCODE_CALL - [통화] 버튼
    - KEYCODE_ENDCALL - [통화 종료] 버튼
    - KEYCODE_BACK - [뒤로가기] 버튼
      void onBackPressed() 메소드를 재정의 하는 것이 일반적
    - KEYCODE_VOLUME_UP - [소리크기 증가] 버튼
    - KEYCODE_VOLUME_DOWN - [소리크기 감소] 버튼
    - KEYCODE_0 ~ KEYCODE_9 - 숫자 0~9까지의 키 값
    - KEYCODE_A ~ KEYCODE_Z - 알파벳 A~Z까지의 키 값

- ✓ 제스처 이벤트 - 스크롤처럼 일정 패턴으로 구분
- ✓ 포커스 이벤트 - 뷰마다 순서대로 주어지는 포커스
- ✓ 화면 방향 변경 - 방향이 가로/세로로 바뀜에 따라 발생
  - layout 파일 2개 준비
    - 가로 화면용 파일은 layout-land 폴더에 저장
    - 화면 방향이 바뀔 때마다 클래스가 사라지고 다시 생긴다
      - onCreate 메소드가 다시 생긴다 → 이전 메소드에서 사용하던 변수가 값이 사라질 수 있다
      - 메소드 밖에서 변수를 선언하면 변수 값이 유지된다
  - Manifest 파일 수정
    - android:configChanges="속성설정"
    - 속성
      - orientation - 방향 전환
        - Configuration.속성 → ORIENTATION_LANDSCAPE – 가로모드
→ ORIENTATION_PORTRAIT - 세로모드
      - screenSize - 화면 크기
      - keyboardHidden - 가상 키보드 유무
      - ……

토스트

- ✓ 간단한 메시지를 잠깐 보여주고 없어지는 뷰
- ✓ 앱 위에 떠 있는 뷰 → 대표적인 위젯
- ✓ 대화상자와 함께 사용자에게 필요 정보 제공
- ✓ 포커스를 받지 않아 대화상자보다 간단하게 사용
- ✓ 디버깅을 목적으로 사용 됨
- ✓ 앱이 화면에서 사라지더라도 필요한 메시지가 그대로 표시 → 앱의 상태와 관계없음
- ✓ 사용법
  - 일반적인 사용법
    - Toast.makeText(
Context context, → Context 클래스를 상속한 액티비티 사용
참조 불가능 시 → getApplicationContext() 사용
String message, → 보여줄 메시지 문자열
int duration → 디스플레이 시간
).show();

- 모양 변경시 사용법
  - Toast toast=new Toast(this); → 선언
  - text.setText("모양 바꾼 토스트"); → 보여줄 메시지
  - toast.setDuration(Toast.LENGTH_SHORT); → 디스플레이 시간
  - toast.setView(뷰 변수); → 보여줄 뷰 지정
  - toast.show(); → 출력

✓ 위치변경

- public void setGravity(
  int gravity, → Gravity.CENTER 등으로 정렬위치 지정
  int xOffset, int yOffset → x, y축 이동 거리
)

✓ 여백

- public void setMargin(float horizontalMargin, float verticalMargin)

✓ 모양 변경

- xml 파일에서의 onClick 속성을 이용하여 함수이름 지정
  Java 파일에서 지정한 함수를 재정의하여 특정 위젯에서만 사용 가능
- 방법
  - drawable 파일 준비 토스트의 모양을 결정
  - layout 파일 준비 토스트 파일을 실행 시 필요 → Background를 drawable 파일로 지정
  - Java 파일에서 인플레이션 구현
    - LayoutInflater inflater=getLayoutInflater();
    - View layout=inflater.inflate(
      R.layout.준비한 layout 파일,
      (ViewGroup)findViewById(R.id.준비한 layout 파일의 TextView의 id 값)
      );
    - TextView text=layout.findViewById(R.id.text);

스낵바

- ✓ 토스트 대신 간단한 메시지를 보여주고자 할 때 사용
- ✓ 외부 라이브러리 사용
  - implementation 'com.android.support:design:28.0.0'
- ✓ 사용법
  - Snackbar.make(v, "스낵바입니다", Snackbar.LENGTH_LONG).show(); → v는 View를 의미

알림 대화창

- ✓ 사용법
  - 선언

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
```

  - 제목

```
builder.setTitle("안내");
```

  - 내용

```
builder.setMessage("종료하시겠습니까?");
```

  - 제목 왼쪽 아이콘 지정

```
builder.setIcon(android.R.drawable.ic_dialog_alert);
```

  - 가장 오른쪽 버튼 작동 지정

```
builder.setPositiveButton("예", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialogInterface, int i) { 작동 정의; }
});
```

  - 가장 왼쪽 버튼 작동 지정

```
builder.setNeutralButton("취소", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialogInterface, int i) { 작동 정의; }
});
```

  - Positive 버튼의 왼쪽 버튼 / 가운데 버튼 작동 지정

```
builder.setNegativeButton("아니오", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialogInterface, int i) { 작동 정의; }
});
```

  - 정의한 대화상자 객체화

```
AlertDialog dialog = builder.create();
```

- 대화상자 출력
  ```
  dialog.show();
  ```

✓ 주의사항

- AlertDialog는 기본 API 이외에 appcompat 패키지의 것도 사용 가능
- Android.support.v7.app 패키지 내 AlertDialog 사용 권장
- 대화창의 아이콘 설정 → 객체 메소드로서 setIcon() 메소드 사용 가능

프로그레스바

✓ 작업의 진행 상태를 중간중간 보여주는 좋은 방법

✓ 형태

- 막대
  - 작업의 진행 정도를 알려줄 수 있도록 막대 모양으로 표시
  - Style 속성 값을 ?android:attr/progressBarStyleHorizontal으로 설정
- 원모양
  - 작업이 진행 중임을 알려줌
  - 원 모양으로 된 프로그레스바가 반복적으로 표시

✓ 사용법

- xml
  - <ProgressBar … />
- Java
  - 선언
    ```
    ProgressBar progressBar = findViewById(R.id.progressBar);
    ```
  - 수치값 표현
    ```
    progressBar.setIndeterminate(false);
    ```
  - 0~100 사이에서 값 설정
    ```
    progressBar.setProgress(80);
    ```

시크바

- ✓ 프로그레스바와 유사
- ✓ 시크바의 값이 바뀔 때 그 값을 다시 프로그레스바와 입력 상자에 표시
- ✓ 시크바의 값이 바뀔 때 그 값을 알려주는 콜백 메소드 사용 필요
- ✓ 사용법
  - xml
    - <SeekBar … />
  - Java
    - 프로그레바 선언

      ```
      progressBar=findViewById(R.id.progressBar);
      ```
    - 텍스트 편집기 선언

      ```
      editText=findViewById(R.id.editText);
      ```
    - 시크바 선언

      ```
      SeekBar seekBar =findViewById(R.id.seekBar);
      ```
    - 시크바 작동 재정의

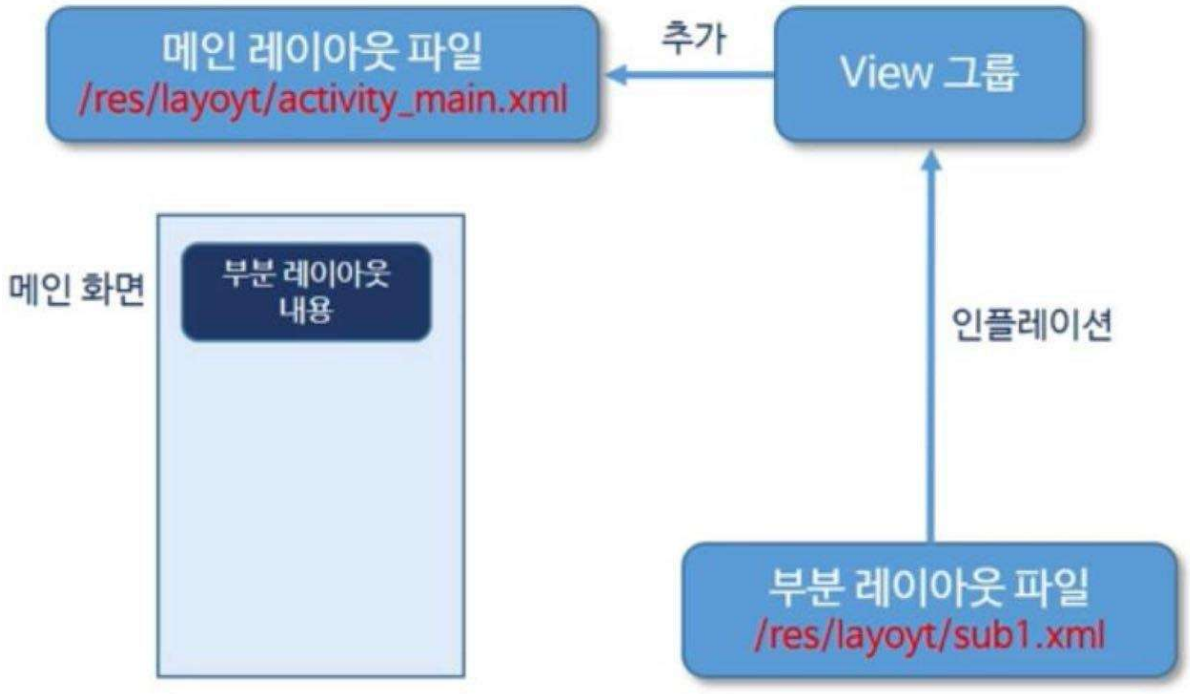
      ```
      seekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {

              변화에 따라 바뀌도록 하는 메소드
              @Override
              public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
                      프로그레스바와 텍스트 편집기의 값을 시크바의 값과 일치시킴
                      progressBar.setProgress(progress);
                      editText.setText(String.valueOf(progress));
              }

              사용하지 않더라도 비워서 재정의 해야함
              @Override
              public void onStartTrackingTouch(SeekBar seekBar) { … }
              @Override
              public void onStopTrackingTouch(SeekBar seekBar) { … }
      });
      ```

인플레이션

- ✓ 레이아웃 내용이 메모리에 객체화 되는 과정
  레이아웃의 내용을 다른 레이아웃의 일부분에 띄울 수 있다
- ✓ (xml 레이아웃 내용 → 메모리에 객체화 → 소스 파일이 객체화 된 내용 참조)
  - *** 레이아웃을 객체화시키지 않으면 그 레이아웃의 위젯 사용 불가능
  - 레이아웃을 객체화해주는 setContentView() 가 우선되어야 한다
- ✓ setContentView()
  - 화면에 나타낼 View를 지정하거나
    레이아웃을 메모리에 객체화하는 역할
  - setContentView(int layoutResID)
  - setContentView(View view, [ViewGroup.LayoutParams params])
  - *** 부분화면을 메모리에 객체화할 수 없는 경우 → 인플레이션 필요
  - 시스템 서비스 제공
  - LayoutInflater 클래스 → getSystemService(Context.LAYOUT_INFLATOR_SERVICE) 이용
  - *시스템 서비스 - 단말이 시작되면서 항상 실행되는 서비스



- ✓
  일부 화면을 xml 레이아웃으로 정의 → 인플레이터를 이용하여 메인 레이아웃에 추가
  - 메인 레이아웃에 부분 레이아웃이 포함된 경우
    - 일부 화면은 LayoutInflater 객체를 사용해 View 그룹 객체로 객체화 (인플레이션)
    - 이후 메인 레이아웃에 추가

- ✓ 사용법
  - Java
    - 레이아웃 파일을 집어 넣을 LinearLayout 선언

```
container=findViewById(R.id.container);
작동시킬 버튼 선언
Button button = findViewById(R.id.button);
버튼 클릭 재정의
button.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
                레이아웃 객체화를 위한 인플레이터 선언 - 시스템 서비스 이용
                LayoutInflater inflater
                        =(LayoutInflater)getSystemService(
                        Context.LAYOUT_INFLATER_SERVICE);
                객체화 - 앞의 레이아웃 파일을 뒤의 레이아웃에 보이게 한다
                inflater.inflate(R.layout.sub1,container,true);
                집어 넣은 레이아웃의 위젯을 선언
                CheckBox checkBox=container.findViewById(R.id.checkbox);
                생성한 위젯 사용(변경)
                checkBox.setText("로딩되었습니다");
        }
});
```

앱 내 화면 전환

- ✓ 대부분의 앱 → 여러 화면으로 구성되고 화면을 전환하면서 실행
  - 독립적인 화면 하나 → 액티비티로 구현
  - 결과적으로 액티비티를 서로 전환하는 관계
- ✓ 안드로이드 앱 구성요소
  - 액티비티 / 서비스 / 브로드캐스트 수신자 / 내용 제공자
- ✓ startActivity()
  - 단순히 액티비티를 띄워 화면에 보이게 함
  - 실제 상황
    - 메인 액티비티에 뜨워야 할 화면이 많아짐
    - 화면을 닫고 원래의 메인 화면으로 되돌아가기 → 데이터를 새로 적용하는 경우도 있음
    - 띄웠던 액티비티로부터 다시 원래의 액티비티로 복귀하면서
      응답을 받아 처리해야 할 수도 있음
    - startActivityForResult(Intent intent, int requestCode) 메소드 사용 필요
    - 메소드 인자들→ 각각의 액티비티를 구별하기 위함

✓ 앱 내 화면 전환시 여러 개의 Java 파일이 생성됨 → 반드시 Manifest 파일에 등록 해야함

✓ 사용법

- Java – 메인 화면 동작
  - 요청 코드

```
public static final int REQUEST_CODE_MENU = 101;
```

  - 메인 화면에서의 동작
    - 버튼 선언

```
Button button = findViewById(R.id.button);
버튼 클릭 재정의
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
            MenuActivity에서 다루는 레이아웃으로 이동
            Intent intent =
                    new Intent(getApplicationContext(), MenuActivity.class);
            startActivityForResult(intent, REQUEST_CODE_MENU);
    }
});
```

  - 값 전달 확인용 메소드 재정의

```
응답 시
if (requestCode == REQUEST_CODE_MENU) {
        응답 메시지 출력
        Toast.makeText(getApplicationContext(),
                "onActivityResult 메소드 호출됨. 요청코드 : " + requestCode
                + ", 결과 코드 : " + resultCode,
                Toast.LENGTH_LONG).show();
        RESULT_OK       → 정상 종료 의미 = 버튼을 눌러 종료한 경우
        → 배경을 눌러 종료된 경우 작동 X
        if (resultCode == RESULT_OK) {
                String name = data.getStringExtra("name");
                전달된 값을 메시지로 출력
                Toast.makeText(getApplicationContext(),
                        "응답으로 전달된 name : " + name,
                        Toast.LENGTH_SHORT).show();
        }
}
```

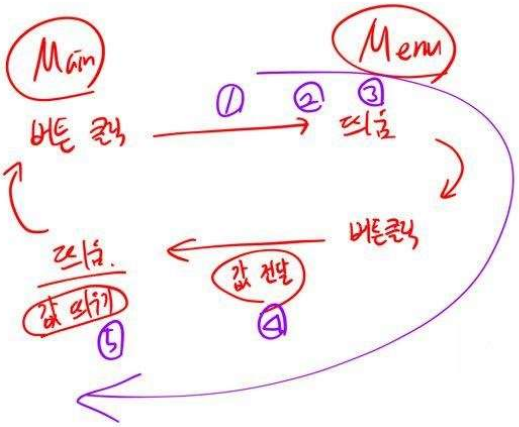
- Java – 이동한 화면 동작

  - 버튼 선언

```
Button button =findViewById(R.id.button);
버튼 클릭 재정의 → 버튼이 클릭되지 않으면 값이 전달되지 않는다
button.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
                인텐트 객체 생성
                Intent intent = new Intent();
                값 삽입
                intent.putExtra("name","mike");

                응답 보내기
                setResult(RESULT_OK,intent);

                현재 액티비티 없애기
                finish();
        }
});
```



인텐트

✓ 사용

- 다른 액티비티를 띄우거나 기능을 동작시키기 위한 수단
- 특정 작업을 수행하기 위해 사용되는 일종의 명령 또는 데이터 전달
- 시스템은 인텐트 안에 들어 있는 명령을 확인
  작성된 액티비티 또는 단말에 설치된 다른 앱의 액티비티 띄우기

✓ 역할

- android.content 패키지 → 앱 구성 요소간에 작업 수행을 위한 정보 전달
- 다른 앱 구성 요소에 인텐트를 전달할 수 있는 대표적인 메소드
  - startActivity() / startActivityForResult()
  - startSevice() / bindService()
  - broadcastIntent()
- 인텐트의 기본 구성 요소 → 액션(Action) + 데이터(Data)
  - ACTION_DIAL tel:01099990000
    - 지정된 전화번호로 전화걸기 화면 표시
  - ACTION_VIEW tel:01099990000
    - 지정된 전화번호로 전화걸기 화면 표시
    - URI 값 유형에 따라 VIEW 액션 기능 달라짐
  - ACTION_EDIT content://contacts/people/2
    - 전화번호부 DB 정보 중에서 ID=2인 정보를 편집하기 위한 화면 표시
  - ACTION_VIEW content://contacts/people
    - 전화번호부 DB 내용 표시

✓ 생성자

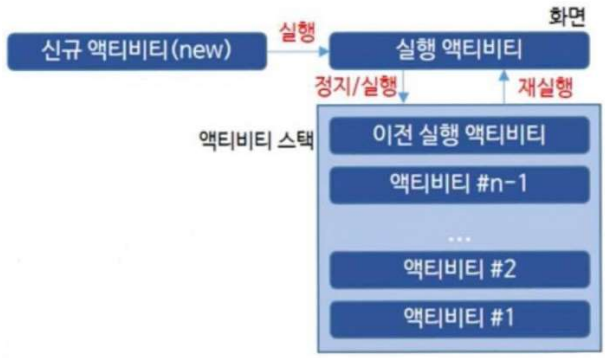
- 다른 인텐트나 클래스 객체를 인수로 지정하여 생성가능

```
Intent()
Intent(Intent o)
Intent(String action, [Uri uri])
Intent(Context packageContext, Class<?> cls)
Intent(String action, Uri uri, Context packageContext, Class<?> cls)
```

- 명시적 인텐트
  - 인텐트에 클래스 객체나 컴포넌트 이름을 지정하여 호출할 대상이 명확한 경우
- 암시적 인텐트
  - 액션과 데이터를 지정했으나 호출 대상이 달라질 경우
  - MIME 타입에 따라 시스템에서 적절한 다른 액티비티 찾아서 띄움

- 속성
  - 범주 (Category)
    - 액션 실행에 필요한 추가 정보 제공
    - CATEGORY_LAUNCHER → 최상위 앱으로 설치된 앱들의 목록을 보여주는 애플리케이션 런처(Launcher) 화면에 이 앱을 보여주는 것 의미
  - 타입 (Type)
    - 인텐트에 들어가는 데이터의 MIME 타입을 명시적으로 지정
  - 컴포넌트 (Component)
    - 인텐트에 사용될 컴포넌트 클래스 이름을 명시적으로 지정
  - 부가 데이터 (Extra Data)
    - 추가 정보를 넣을 수 있는 Bundle 객체를 통해 인텐트 안에 넣어 전달
    - 전자메일의 경우 → 제목+내용 등

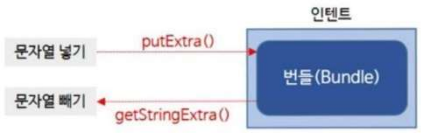
플래그와 부가 데이터



- ✓ 계속적인 startActivityXXX() 메서드 호출 → 메모리에 여러개 생성
- ✓ 같은 액티비티에 대해 인텐트를 두변 보내면 중복 액티비티가 뜸 → 플래그로 조정 가능
- ✓ 플래그
  - 액티비티는 ActivityMannager에 의해 스택에서 관리
  - 대표적인 플래그
    - FLAG_ACTIVITY_SINGLE_TOP
      - 액티비티 생성 시 이미 생성된 액티비티가 있으면 그대로 사용 추가 생성 X
    - FLAG_ACTIVITY_NO_HISTORY
      - 처음 이후에 실행된 액티비티는 스택에 추가되지 않음
    - FLAG_ACTIVITY_CLEAR_TOP
      - 해당 액티비티 위에 있는 다른 액티비티를 모두 종료 처리

✓ 부가 데이터

- 로그인 화면(ID+PW) → 메인화면으로 전달
- 데이터를 다른 앱에서 공유 가능? → 인텐트 안에 부가 데이터를 넣어 전달하는 방법 권장
- 인텐트 안에는 버들 객체 제공 : 해시 테이블(key, value) 과 유사
- 기본 자료형 또는 바이트 배열, Serializable 객체 등 가능



✓ 대표 메서드

- Intent putExtra(String name, String value)
- Intent putExtra(String name, int value)
- Intent putExtra(String name, boolean value)
- String getStringExtra(String name)
- int getStringExtra(String name, int defaultValue)
  - 데이터 없으면 디폴트 값으로 반환 됨
- boolean getStringExtra(String name boolean defaultValue)
  - 데이터 없으면 디폴트 값으로 반환 됨

✓ 전달할 데이터가 객체 자료형인 경우

- 객체 자체를 전달 불가능
  - 객체 데이터는 바이트 배열로 변환
    또는 Serializable 인터페이스를 구현하는 객체로 직렬화
  - Serializable 인터페이스와 유사한 Parcelable 인터페이스 사용 권장
- 인터페이스
  - 직렬화 했을 때 크기가 작아 안드로이 내부의 데이터 전달에 자주 사용
  - 기본적으로 두 가지 메소드 모두 구현하여 사용
    - public abstract int describeContents()
      - 직렬화하려는 객체의 유형 구분
    - public abstract void writeToParcel(Parcel dest, int flags)
      - 객체가 가지고 있는 데이터를 Parcel 객체로 작성
      - Parcel 객체는 readXXX(), writeXXX() 제공

태스크 관리

- ✓ 앱의 실행
  - 해당 앱은 하나의 프로세스 위에서 동작
  - 하나의 앱은 하나의 독립 프로세스로 동작
  - 각 프로세스는 정보 공유가 어렵다 → 시스템의 도움 필요
  - 시스템에서 이런 액티비티의 각종 정보를 저장해 두기 위해 태스크를 생성
  - 앱에서 다른 앱의 화면을 띄우지 않고 따로 실행 → 다른 앱 태스크 별도 생성
  - 두 가지 입장
    - 시스템이 알아서 태스크 처리
    - 직접 태스크 제어 필요 → Manifest 파일에서 액티비티 등록 시 태스크 설정 가능
- ✓ 자신의 앱 띄우기
  - 자기 자신 화면에 새로 생성
    ```
    Intent intent = new Intent(getApplicationContext(), MainActivity.class);
    startActivity(intent);
    ```
  - [BACK] 버튼으로 다시 돌아올 수 있다
  - 돌아가거나
    계속 누르게 되면 홈화면으로 이동하는데
    이는 앱 종료가 아니라 프로세스에서 계속 동작하여 메모리를 차지하게 된다
  - *** Manifest 파일의 android:launchMode=”옵션” 속성을 수정해야함
    - SingleTop / SingleTask / SingleInstance 를 이용하여 독립 프로세스 실행하지 않도록 함

액티비티 생명주기

- ✓ 시스템
  - 실행되는 앱의 상태를 직접 관리 (대부분의 모바일 OS)
  - 독립적인 앱이 시스템에 의해 관리 X → 실행 앱이 과도한 메모리 점유 또는 화면 표시 점유
  - 앱 사용 중에 전화가 오면 통화 앱 화면 표시 → 자신의 앱은 다른 화면 뒤로 들어가 중지
  - 액티비티 처음 실행시
    - 메모리에 만들어지는 과정부터 시작해서 실행, 중지, 메모리 해제 과정의 정보를 지님
    - 상태 정보는 시스템이 관리 → 각 상태에 해당하는 메소드를 자동으로 호출
    - 실행(Running)
      - 화면 상에 액티비티가 보이면서 실되는 상태
      - 액티비티 스택의 최상위에 포커싱

- 일시중지(Paused)
  - 사용자에게 보이지만 다른 액티비티가 위에 있어 포커스를 못 받는 상태
- 중지(Stopped)
  - 다른 액티비티에 의해 완전히 가려진 상태

- 액티비티의 상태정보가 변함 → 생명 주기(Life Cycle)를 가짐

✓ 생명 주기 (Life Cycle)

- 액티비티가 처음 만들어진 후 없어질 때까지 상태가 변하면서 각 해당하는 메소드가 자동 호출 됨
- 메소드
  - onCreate()
    - 액티비티가 처음에 만들어졌을 때 호출
    - 화면에 보이는 View들의 일반적인 상태를 설정하는 부분
  - onStart()
    - 액티비티가 화면에 보이기 바로 전에 호출됨
    - 액티비티가 화면상에 보이면 다음에 onResume()
  - onResume()
    - 액티비티가 사용자와 상호작용하기 바로 직전에 호출함
  - onRestart()
    - 액티비티가 중지된 이후에 호출되는 메소드로 다시 시작되기 직전에 호출
    - 이 메소드 다음은 onStart()
  - onPause()
    - 또 다른 액티비티를 시작하려고 할 때 호출
    - 이 메소드가 리턴되기 전에는 다음 액티비티 시작 불가
  - onStop()
    - 액티비티가 사용자에게 더 이상 보이지 않을 때 호출됨
    - 액티비티가 소멸 또는 다른 액티비티가 화면을 가릴 때
  - onDestroy()
    - 액티비티가 소멸되어 없어지기 전에 호출
    - 액티비티가 마지막으로 받는 호출

✓ *** 액티비티를 중지시키기 전에 호출되는 onSavedInstanceState() 메소드를 이용해 데이터 임시저장도 가능

- ✓ 실행 결과
  - 화면이 보이면
    - onCreate, onStart, onResume 순서로 호출
  - [BACK] 버튼을 누르면
    - onPause, onStop, onDestroy 순서로 호출
  - Logcat 창에서 상태 모니터링 가능
    앱을 반복 실행하면서 어떤 일이 일어나는지 확인 가능

//

Xml

- ✓ layout_width="가로 폭"
- ✓ layout_height="세로 높이" → wrap_content - 내용만큼 크기 설정
  → match_parent - 부모 레이아웃의 크기만큼 설정
- ✓ layout_weight="비율" → 1:1:1 등의 비율로 설정
- ✓ orientation="정렬 방식" → vertical - 세로로 나열
  → horizontal - 가로로 나열
- ✓ layout_column="배열의 위치(정수)" → TableRow를 배열 취급하여
- ✓ layout_span="차지할 크기(자연수)" → TableRow의 한 블록을 가로로 늘림
- ✓ gravity="속성" → top - 가장 위쪽
  → bottom - 가장 아래쪽
  → left - 가장 왼쪽
  → right - 가장 오른쪽
  → fill_vertical - 높이 최대
  → fill_horizontal - 폭 최대
  → center - 가장 중앙
  → fill - 높이, 폭 최대
  → clip_vertical - top, bottom의 옵션으로 상, 하부를 컨테이너 경계로 둠
  → clip_horizontal - left, right의 옵션으로 좌,우를 컨테이너 경계로 둠
- ✓ layout_gravity="속성" → gravity의 속성과 동일
- ➢ gravity는 뷰 안의 텍스트 배치 지정 (내용)
- ➢ layout_gravity는 뷰, 레이아웃 배치 지정(자기 자신의 위치 지정)
- ✓ layout_constraint현재 뷰 속성_to부모 또는 그 외의 뷰 속성Of="parent 또는 id"
  → 속성은 Start / End / Top / Bottom / Left / Right 등 사용

//

Java / Kotlin

- ✓ 레이아웃 사용
  - setContentView(R.layout.레이아웃 이름);
  - R.layout.레이아웃 이름
    - R - res 폴더
    - .layout - layout 폴더
    - .레이아웃 이름 - 레이아웃 이름.xml 을 클래스로 받아옴
- ✓ 위젯 사용
  - 위젯 사용할 변수 = findViewById(R.id.위젯 id);
    사용할 변수 . 사용할 인터페이스…{…} - 오버로딩하여 작동 시킴

//