DEVIEW
2019

앱 패러다임의 변화 어떻게 대응할 것인가?

- 구조 설계와 모듈 관점의 접근

신동길
네이버 앱 개발
NAVER

CONTENTS

DEVIEW
2019

1. 무엇이 변했는가?
2. 안드로이드 앱의 구조와 요소
3. 어떤 구조를 적용할 것인가
4. 효율적인 앱이란?

DEVIEW
2019

1. 무엇이 변했는가?

1.1 무엇이 변했는가? (1/2)

Hardware

- MultiCore
- Large Memory
- Big Display
- Foldable

Platform

- Dalvik->JIT/ART
- Many Restrictions
- App Bundle(Dynamic Feature & Delivery)

1.1 무엇이 변했는가? (2/2)

DEVIEW
2019

Language

- Functional Programming
- Java8, Kotlin

다양한 프레임워크

- JetPack
- RxJava
- Lottie
- Retrofit
- Glide, Picasso, OkHttp

1.2 좋은 앱이란?

DEVIEW
2019

좋은 앱은

- Multi-Featured
- Multi-Media
- Multi-CORE
- Big-Sized
- Multi-Module

에서 효율적 + 빠른 개발과 유지 보수

1.3 객체형이 모바일 환경에 적합한가? (1/2)

DEVIEW 2019

단말의 앱은

- 복잡한 UI 와 잦은 변경
- 많은 객체가 정의
- 플랫폼 요소가 다양함(멀티 미디어)

=> BIG Size, 복잡한 GUI 환경에 자바가 적합하지는 않음

=> 좀 더 간결한 코딩이 필요 (많은 기능이 필요)

1.3 함수형 vs 객체형 프로그래밍(2/2)

DEVIEW 2019

Why Functional (Not Overriding)

- UI 코드를 방대함 이를 간결하게 표현할 수 있는 도구 필요
- 코어 로직에 대한 간결한 표현 방식의 필요성
- 많은 함수를 가진 큰 객체의 이벤트 처리의 효율성 제고 필요

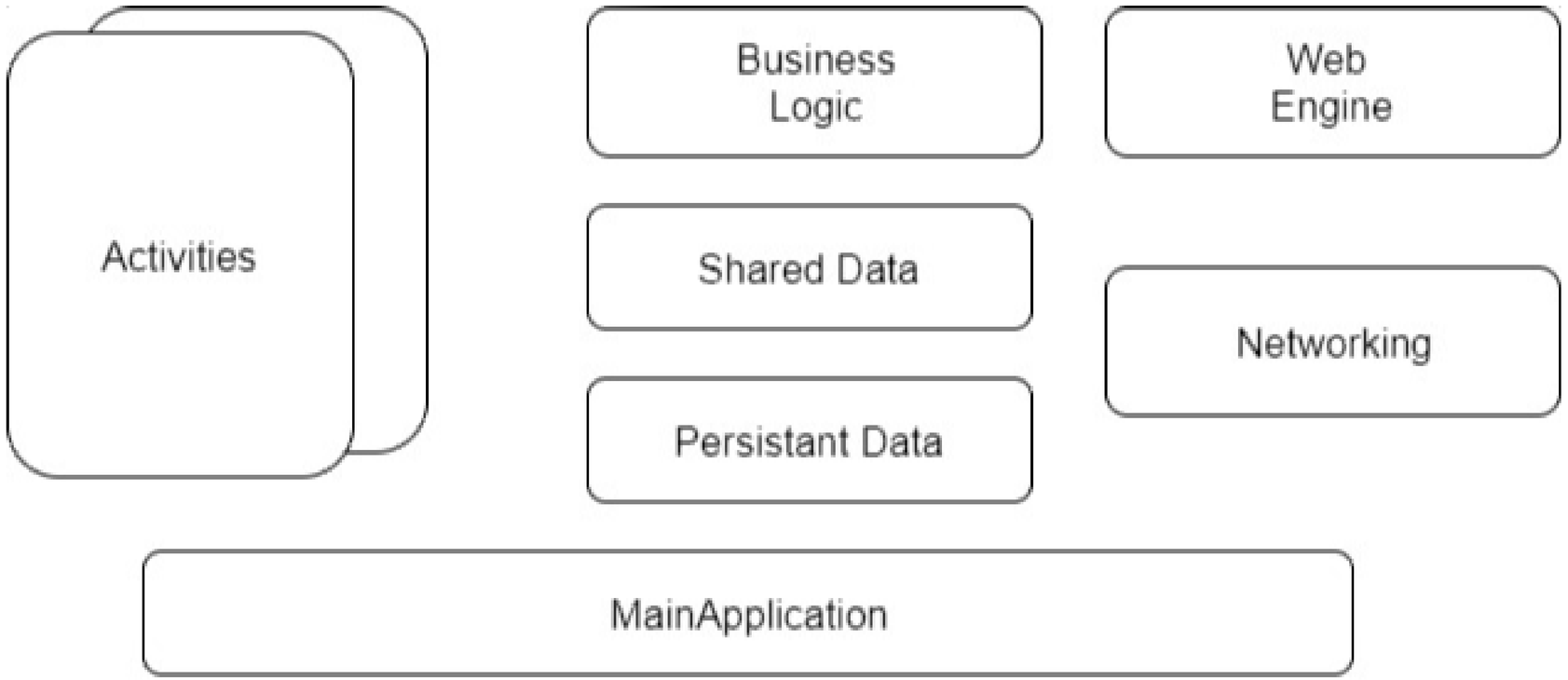
```
this.findViewById(R.id.content).setOnClickListener(
        new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                //Action
            }
        });
```

```
this.findViewById<View>(R.id.content) += {
    //Action
}


operator fun View.plusAssign(block:(v:View)->Unit)
        = setOnClickListener { block(it) }
```

DEVIEW
2019

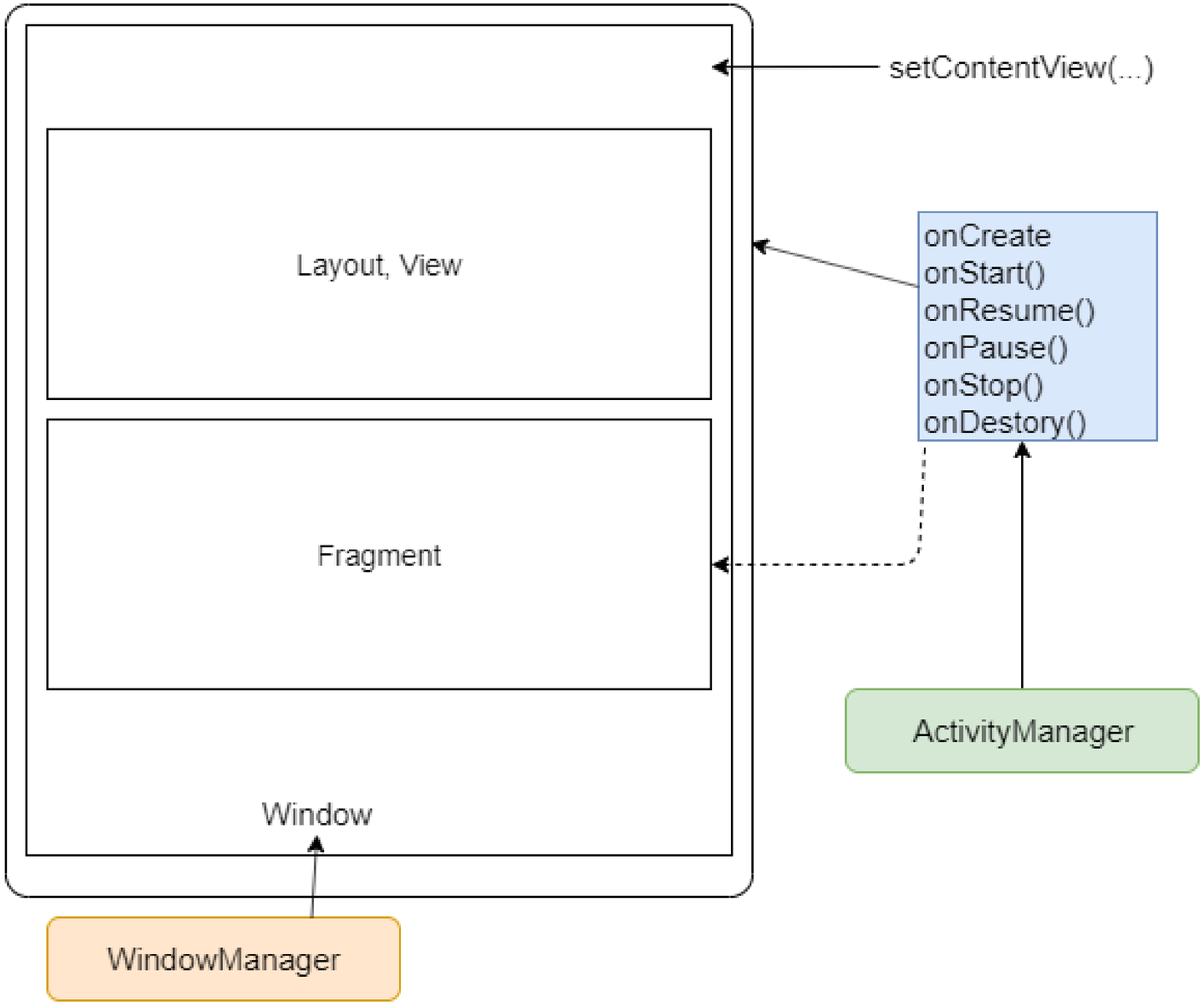
2.기본 앱의 기본 구조와 요소

2.1 안드로이드 앱의 구조



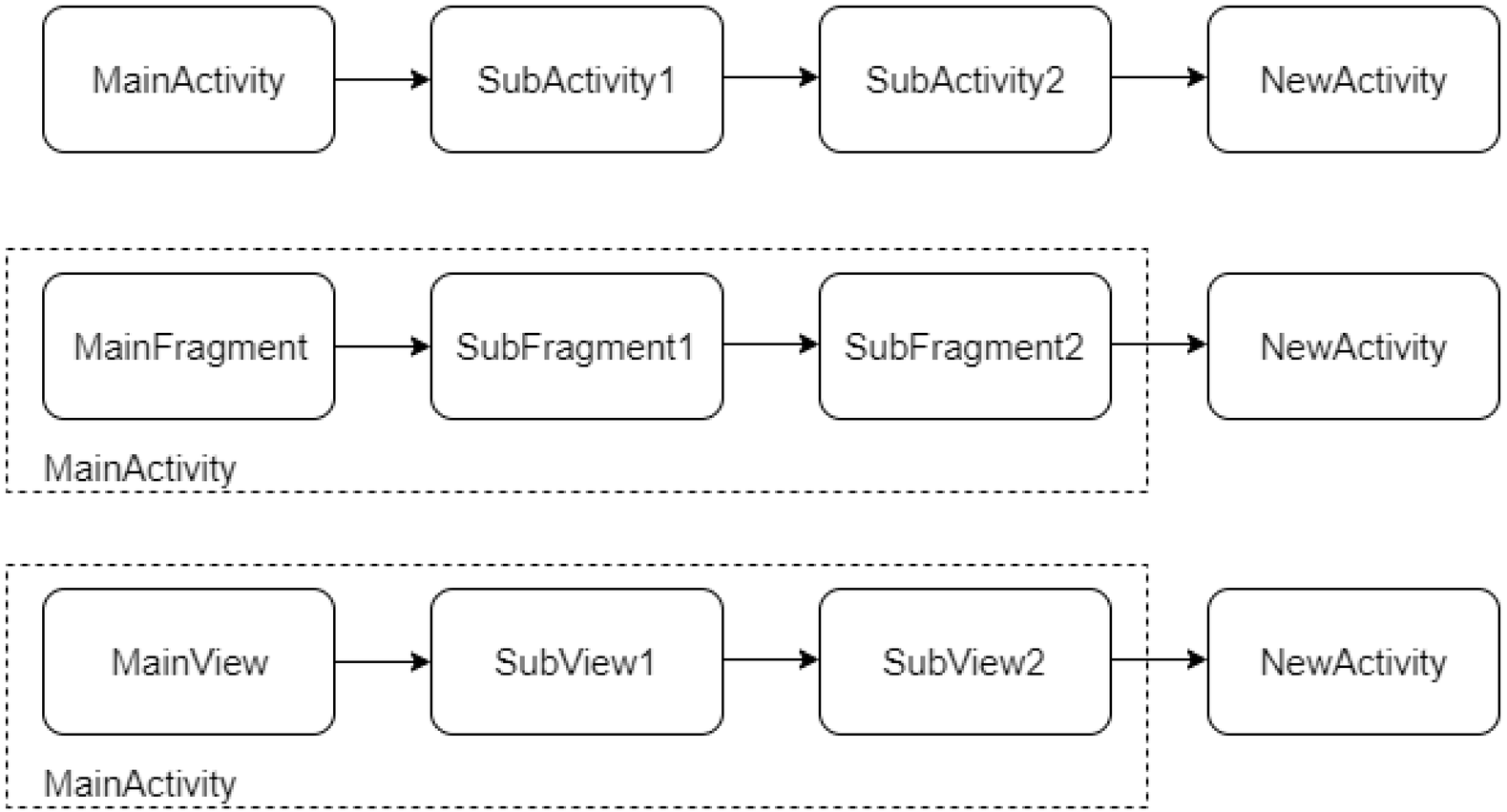
2.2 UI구조 - Activity, Window, Fragment

DEVIEW 2019



2.2 UI와 Navigation

DEVIEW 2019



2.3 Multi-Process(1/2)

DEVIEW
2019

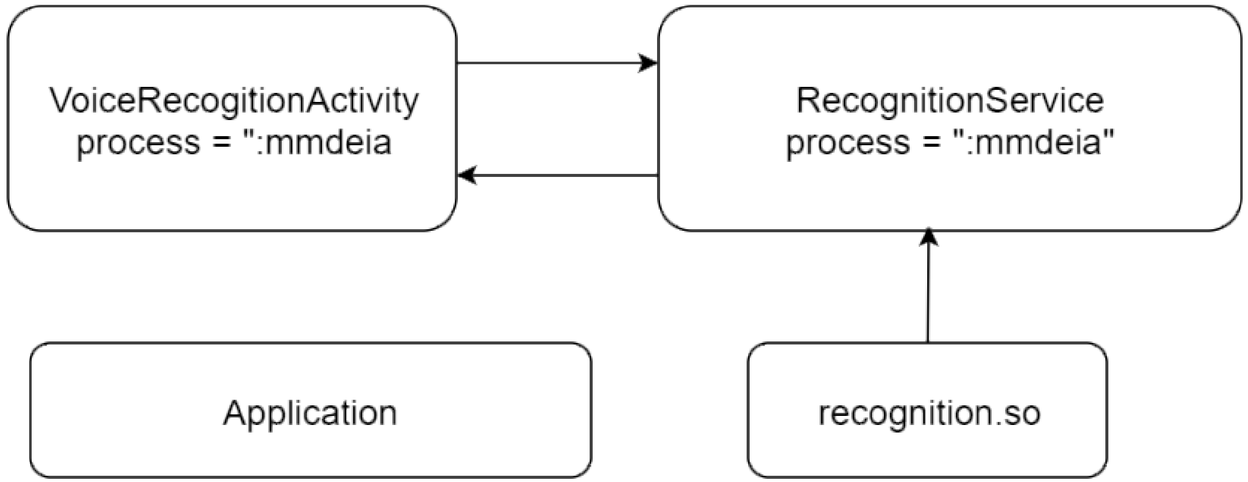
멀티 프로세스 필요성

- 메인과 구별되는 기능요소
- 별도의 모듈(Dynamic Linked Library) 로드 하는 요소
- 모듈의 크기가 크고 필요에 따라 제거할 수 있는 모듈
  - 멀티 미디어와 같이 자원을 많이 소요하는 요소
  - 다른 앱에서 해당 기능만을 사용할 경우
- 크래시 발생시 해당 모듈에서 전파되기를 원하지 않을 때

2.3 Multi Process(2/2)

멀티 프로세스 구현

- 별도의 Activity, Service에 process 속성 지정
- 프로세스마다 Application.onCreate()에서 초기화
- Content Provider, Intent Bundle, Intent Broadcast 로 데이터 공유



3.어떤 아키텍쳐를 적용할 것인가?

3. 아키텍쳐 기반의 접근

DEVIEW 2019

Language Tookit

- Kotlin Tookit

UI-Data 패턴

- MV, MVC, MVP, MVVM

Network Module, Complicated UI

- State Model – 간단한 이벤트로 상태관리 (디바이스)
- FSM – 이벤트에 따른 상태변화와 동작 구조(프로토콜, 앱상태)

Multi-Media Processing

- Event Queue Model
- Piped Stream Model

Module Projects

- Layed Module
- Dynamic Feature Module

3.1 Kotlin Toolkit (1/2)

DEVIEW 2019

간결한 하고 효율적인 표현 위한 도구

- 전역 변수
- 함수 확장
- 프로퍼티
- 연산자의 재정의 활용
- DSL 스타일의 정의

```
val cookieSet:List<Pair<String,String?>>? = cookies["http://m.naver.com"]
cookieSet?.forEach { it: Pair<String, String?>
    if (it.first == "svcList") {
        print("${it.first} = ${it?.second}")
    }
}
```

```
val cookies
 inline get() = CookieManager.getInstance()

operator fun CookieManager.get(url:String):List<Pair<String,String?>>? {
    val cookieSet = this.getCookie(url);
    return if (cookieSet.isNullOrEmpty() == false) {
       cookieSet.split( ...delimiters: ';').map { it.split( ...delimiters: "=").let{ Pair(it[0],it[1])}}
    } else null
}
```

3.1 Kotlin Toolkit(2/2)

DEVIEW 2019

함수 확장+DSL 스타일의 정의를 통한 화면 방향 판단

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

    isLandscape {
        showToast( message: "Not supporting orientation!!")
    }

    if (isLandscape.not()) {
        //...
    }
    //...
}

val Activity.isLandscape:Boolean
    inline get() = resources.configuration.orientation == Configuration.ORIENTATION_LANDSCAPE

inline fun Activity.isLandscape(action: ()-> Unit) {
    if (resources.configuration.orientation == Configuration.ORIENTATION_LANDSCAPE) {
        action()
    }
}

inline fun Activity.isPortrait(action: ()-> Unit) {
    if (resources.configuration.orientation == Configuration.ORIENTATION_PORTRAIT) {
        action()
    }
}
```

3.2 Event Re-Dispatching (1/3)

복잡한 Activity-View 관계

- 많은 이벤트 처리가 필요
- 기능이 많아서 깊은 View Depth

- e.g)인앱 브라우저 툴바

⇒ 이벤트의 라우팅 필요 (Activity, WebView, …)

3.2 Event Re-Dispatching (2/3)

필요성 (Activity 예제)

- Activity의 이벤트를 Child View나 비즈니스 로직에서 사용하는 경우가 많다
- 함수를 만들고 Activity에서 함수들을 만들어야 함
- LifeCycle, Permissions, onBackKeyPressed, onActivityResult, …

```
activity.startActivityForResult(new Intent(getContext(), MainActivity.class), requestCode: 1001,
        (requestCode,resultCode,intent)->{
            //procesing result
        });

activity.requestPermission(new String[]{Manifest.permission.ACCESS_WIFI_STATE}, requstCode: 100, (code, permission, granted)->{
    if (code == 100) {
        //Progress Netwotk Action
    }
});

activity.addBackKeyListener( ()->{
    //Process Backey
    return true;
});
```

3.3 Event Re-Dispatching (3/3)

DEVIEW 2019

<= Activity의 이벤트를 Re-dispatch의 간단한 구현 예

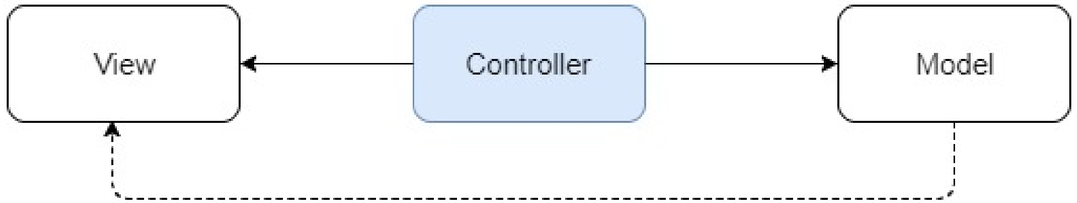
```
Map callMap = new HashMap<Integer,Object>();
List<Object> eventList = new LinkedList<>();

@Override
public void onBackPressed() {
    for(Object l : eventList) {
        if (l instanceof OnBackPressedListener) {
            if (((OnBackPressedListener)l).onBackPressed() == true) {
                return;
            }
        }
    }
    super.onBackPressed();
}
public void startActivityForResult(Intent intent, int requestCode, OnActivityResultListener resultCallback) {
    callMap.put(requestCode, resultCallback);
    startActivityForResult(intent, requestCode);
}
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    Object l =  callMap.get(requestCode);
    if (l != null && l instanceof OnActivityResultListener) {
        ((OnActivityResultListener)l).onActivityResult(requestCode,resultCode,data);
    }
    super.onActivityResult(requestCode, resultCode, data);
}
public void requestPermission(String[] permssison, int requstCode, OnPermissionResultListener callback) {
    callMap.put(requstCode, callback);
    ActivityCompat.requestPermissions( activity: this, permssison, requstCode);
}

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantRe
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    Object l =  callMap.get(requestCode);
```

3.3 View-Data Model

View-Data 패턴 – MV,MVC, MVP, MVVM

- 핵심 키워드는 View 와 Model의 의존성의 줄이고 간결하게 데이터를 반영
- 패턴은 통일된 인터페이스를 갖지만 코드의 량을 증가

- 작은 부분에 과도하게 적용하면 배보다 배꼽이 더 큼
- 협업 시에 레이어 구분을 위해서 사용은 좋은 예
- 단순 구조에 적용은 신중해야 함



3.4 Network Model(1/3)

Session-Client모델

- Transaction - Request/Response 관리
- Session-연속적인 Transaction을 통한 상태 관리(State)
- Client – Session 관리 및 전체 상태관리

State 모델

- 여러 개의 변수(플래그)를 묶어 상태로 관리
- 비동기적인 상태 관리(주로 네트워크) – Session & State
- 카메라, 오디오 장치와 같이 상태를 가지는 경우 – State
- 많은 이벤트와 복잡한 상태의 경우– Finite State Machine

3.4 Network Model (2/3)

업 로드 서비스 예 (Session - State Model)

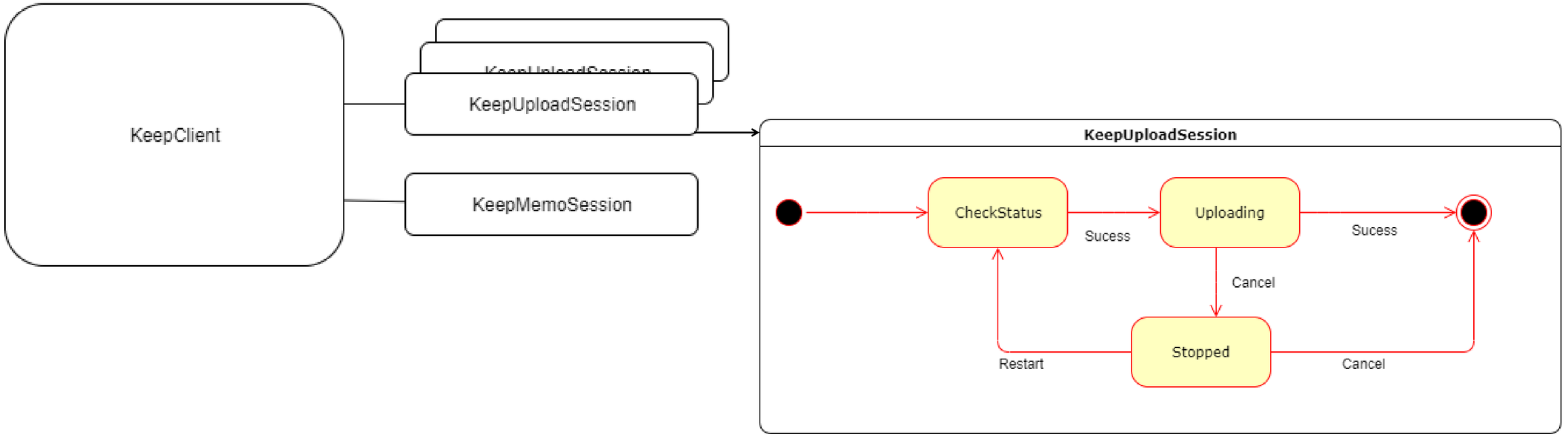
- 기존에 일부라도 업로드 되었는지 확인(Prechecking Tranasction)
- 파일을 조각 내어 연속적인 업로드(Upload Transactions)
- 3개의 파일 동시 업로드
- 업로드 중 취소 가능

3.4 Network Model(3/3)

DEVIEW 2019

Keep Client (Session - State Model)

- 기존에 일부라도 업로드 되었는지 확인(Prechecking Tranasction)
- 파일을 조각 내어 연속적인 업로드(Upload Transactions)



3.5 MultiMedia Model

Message Queue Model

- 센서와 같이 작은 크기의 이벤트가 연속적으로 발생하는 경우

Produce-Consumer Model

- 미디어를 생성-소비의 단순한 구조에 적합

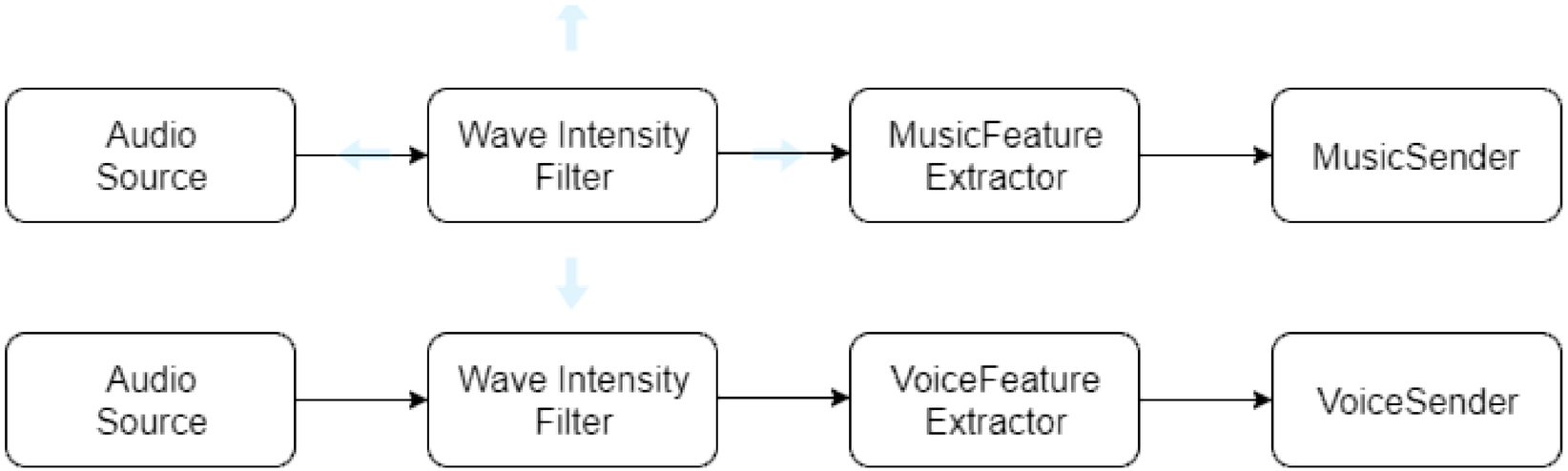
- e.g)Camera-QR Scanner

Piped Filter Model

- 미디어 생성-변환-소비를 실시간에 동시에 수행해야 하는 구조에 적용

3.5 Piped Filter Model(1/2)

- 실시간 미디어는 Source, Transform, Sink의 모델
- 실시간 미디어 처리는 공통 모듈이 많음 (Threading, Buffering, Streaming)
- 미디어의 병렬처리를 통한 실시간성이 요구되거나 변화가 잦은 경우



3.5 Piped Filter Model (2/2) - Example

DEVIEW 2019

- MediaProcessor추상 객체

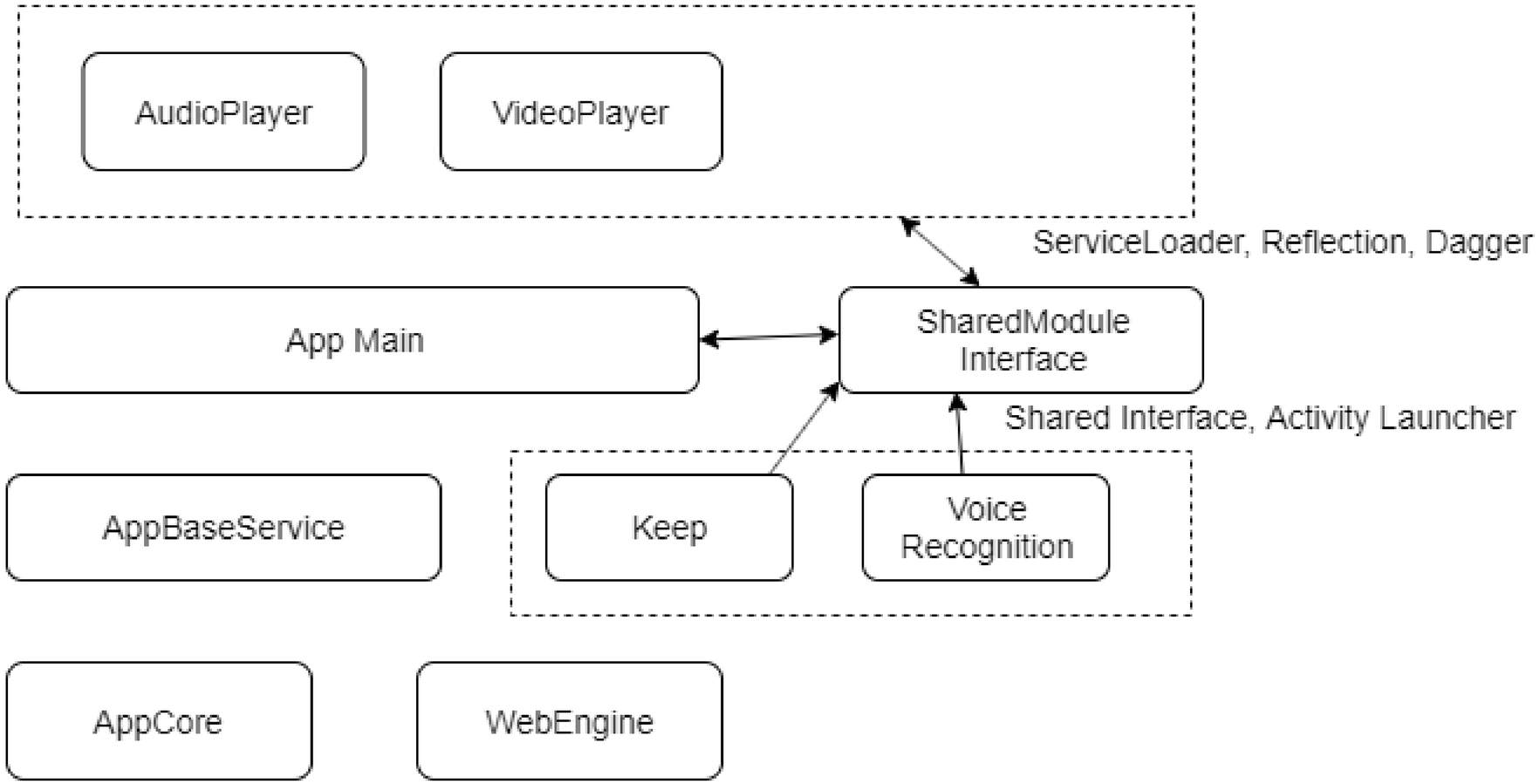
```
public class MediaProcessor implements Runnable{
    public enum State {
        Unknown, Ready, Running, Stopped, Paused
    }
    enum Type {
        Source, Transform, Sink
    }

    public PipedInputStream inputStream;
    public PipedOutputStream outputStream;
    protected  State state = State.Unknown
    Thread mediaThread;

    public void init(Type type) {
        if (type == Type.Source || type == Type.Transform) {
            outputStream = new PipedOutputStream();
        }
        if (type == Type.Sink || type == Type.Transform) {
            inputStream = new PipedInputStream();
        }

        state = State.Ready;
    }

    void start() {
        if (state == State.Ready) {
        mediaThread = new Thread( target: this);
        mediaThread.start();
        state = State.Running;
        }
    }

    void stop() throws Exception {
        if (state == State.Running && mediaThread != null) {
            state = State.Stopped;
            mediaThread.join();
        }
    }

    void connectTo(MediaProcessor nextProcessor) throws Exception {
        nextProcessor.inputStream.connect(outputStream);
    }

    @Override
    public void run() {
        if (state == State.Running) {
            processMedia();
        }
    }

    protected void processMedia() {
    }
}
```

3.6 대용량 앱 & 모듈 구조(1/2)

필요성

- Apk ~ 100MB 근처
  - Apk Extension(.obb)
  - Another Apk (테마 팩, 웹엔진)
  - Instant App
  - App Bundle
  - Dynamic Feature Module
- 복잡한 구조의 정리와 효율적인 컴파일
- 기능의 분리 & 리소스와 코드의 컴포넌트화
- 다이나믹 피쳐를 동적 설치
- 타 조직과 협업 개발

3.6 대용량 앱 & 모듈 구조(2/2)



4. 효율적인 앱(개발)이란?

DEVIEW
2019

- 복잡한 UX를 쉽게 또는 간결하게 구성할 수 있어야 함
- 멀티미디어의 처리는 단순하고 확장이 쉬워야 함
- 협업이 쉽고, 개발 속도가 빠른 구조
- 리소스와 코드는 기능 별로 분리 되어야 함

⇒구조적인 설계 + 탄탄한 기반 모듈(Toolkit)

⇒기능적으로 잘 분리된(모듈화된) 구조

DEVIEW
2019

Q & A

DEVIEW
2019

Thank You