안드로이드 빌드: 설탕없는 세계

김용욱 leonardo.kim
카카오뱅크

if(kakao)dev 2019

00 발표에 앞서

안드로이드 빌드: 설탕없는 세계

01 Post Compilation를 메인으로 다룹니다.

02 Desugar: 개발자를 위한 편의를 빌드 과정에서 제거하는 것.

03 안드로이드 빌드 과정 전체, 특히 후 처리는 쓰디 쓴 과정입니다.

04 빌드 과정을 모른다고 가정하고 안드로이드 빌드의 특이성과 전체 과정부터 다루며 Post Compilation으로 이어갑니다.

05 이 발표 자체도 설탕이 없습니다. T.T

후처리를 알면 무엇이 도움이 되나요?

01 보안을 강화할 수 있습니다.

02 언어의 한계를 극복할 수 있습니다.

03 APT의 한계도 극복할 수 있습니다.

04 반복적인 작업을 자동화를 할 수 있습니다.

05 ~~아는 척을 할~~ 교양을 쌓을 수 있습니다.

후처리를 알면 무엇이 도움이 되나요?

보안을 강화할 수 있습니다.

프로가드, R8등의 보안 솔루션들이 안드로이드 빌드 후처리 과정 (Transform) 동안 바이트 코드를 변조하고 난독화를 합니다.

후처리를 알면 무엇이 도움이 되나요?

언어의 한계를 극복할 수 있습니다.

1. 자바에서 프로퍼티는 없지만 대입문을 커스텀 getter / setter 호출로 전부 변환할 수 있습니다.

2. 언어차원에서 지원하기 힘든 Lazy Evaluation을 구현할 수 있습니다.
    1. POJO는 필드를 채워줘야 하고 Lazy Evaluation은 커스텀 Getter / Setter가 필요합니다.
    2. 바이트 코드를 변조하면 대입문을 Lazy Evaluation용 메서드로 변환할 수 있습니다.

3. 람다를 지원하지 못하는 JVM에서도 바이트코드 가공을 통해 람다를 쓸 수 있습니다.
    1. 일정 버전 이상의 Retro Lambda
    2. Desugar - 현재 안드로이드의 람다 구현

후처리를 알면 무엇이 도움이 되나요?

APT의 한계도 극복할 수 있습니다.

1. APT는 완전히 새로운 클래스를 만들거나 상속받은 클래스를 만듭니다.
2. 기존 클래스의 동작은 수정할 수 없습니다.
3. 상속받은 클래스는 기존 클래스의 특정 필드나 특정 메서드가 어떤 식으로 동작할지 약속하고 (프로토콜을 만들고) 그에 맞추어 사용해야 합니다.

후처리를 알면 무엇이 도움이 되나요?

반복적인 작업을 자동화할 수 있습니다.

1. 어떤 리소스가 얼마나 쓰였는지 어디에서 안 쓰고 있는지를 확인할 수 있습니다.
2. onResume, onPause 등 생명주기에 진입할 때 코드의 변경없이 로그를 넣을 수 있습니다.
3. DB, 네트워크 접속에 대한 로깅 코드를 추가할 수 있습니다.
4. 모든 메서드가 호출될때 마다 로깅하고 싶다면 그 작업 역시 가능합니다.

이 모든 일들은 자바나 코틀린 코드를 그때 그때 넣어 노가다로 해결할 수도 있습니다.
그렇게 하지 않아도 된다는 것이 후처리의 장점입니다.

후처리를 알면 무엇이 도움이 되나요?

교양을 쌓을 수 있습니다.

지식을 쌓는 일은 개인적인 흥미거리이기도 합니다.

또 예상하지 못한 방향으로 언젠가 다른 방향에서 도움이되기도 합니다.

01 안드로이드 빌드의 특수성

안드로이드 빌드의 특수성

Java VM은 스택기반의 VM

32비트 기반의 스택

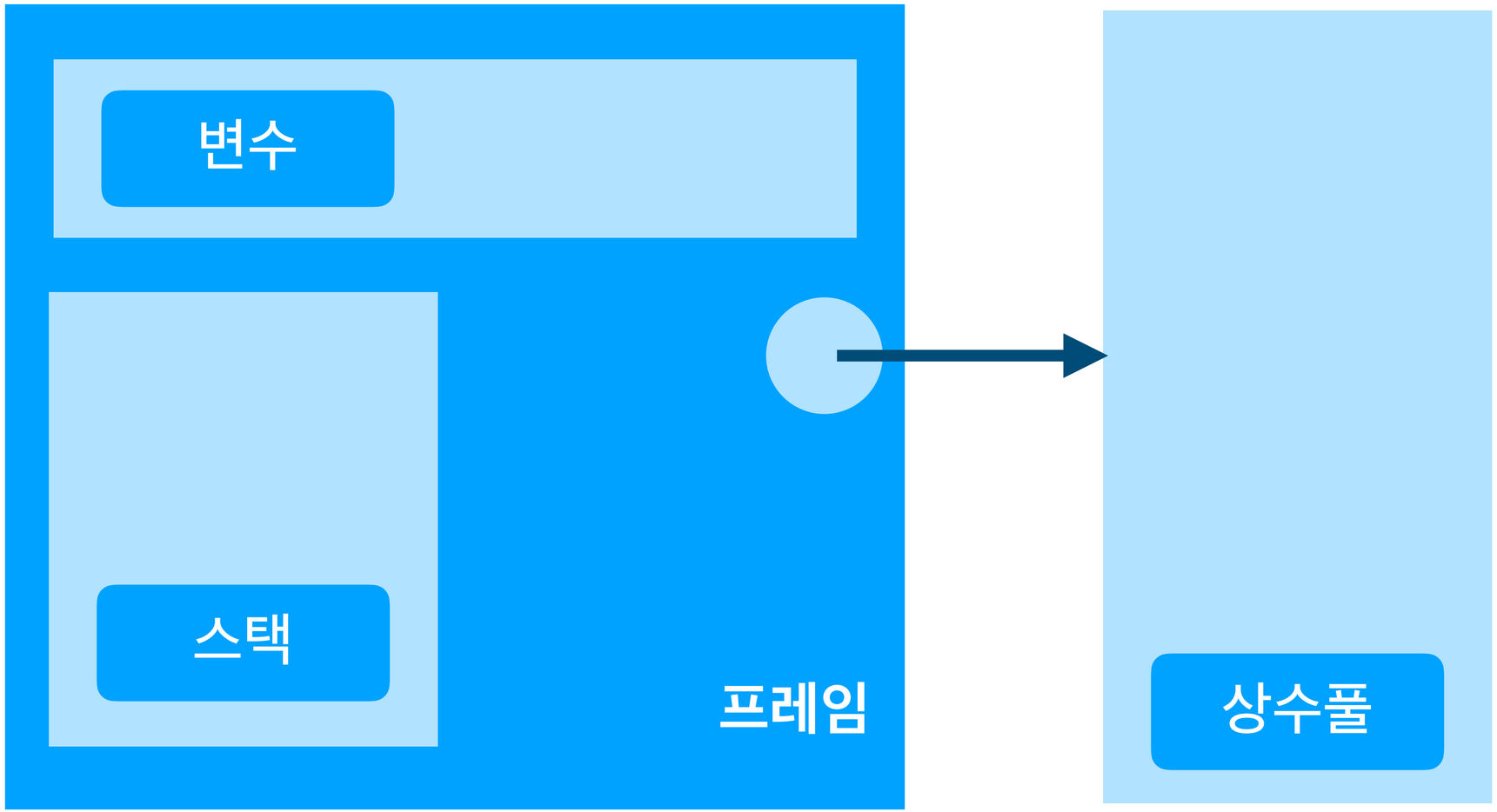
대부분의 타입 하나의 스택.

char를 사용해도 32비트 요소.

64비트 자료형은 두개.

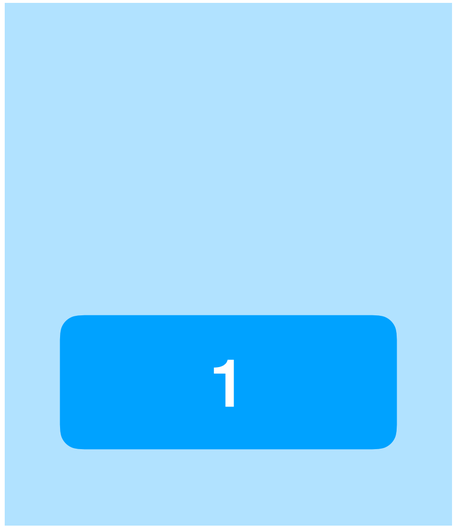
상수의 종류를 구별하지 않습니다.

모든 종류의 상수가 같은 상수 풀.



안드로이드 빌드의 특수성

스택기반의 자바 VM, 1+2



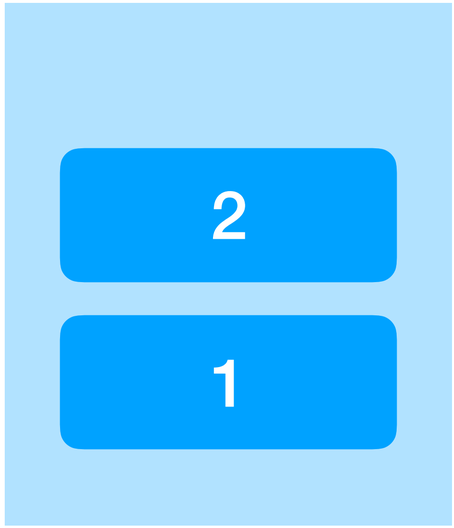
istore_1

1을 스택에 넣습니다.

상수 -1부터 5까지는 별도의 istore 명령이 있습니다.

안드로이드 빌드의 특수성

스택기반의 자바 VM, 1+2



istore_1

1을 스택에 넣습니다.

상수 -1부터 5까지는 별도의 istore 명령이 있습니다.

istore_2

2를 스택에 넣습니다.

안드로이드 빌드의 특수성

스택기반의 자바 VM, 1+2

1 + 2

istore_1

1을 스택에 넣습니다.

상수 -1부터 5까지는 별도의 istore 명령이 있습니다.

istore_2

2를 스택에 넣습니다.

iadd

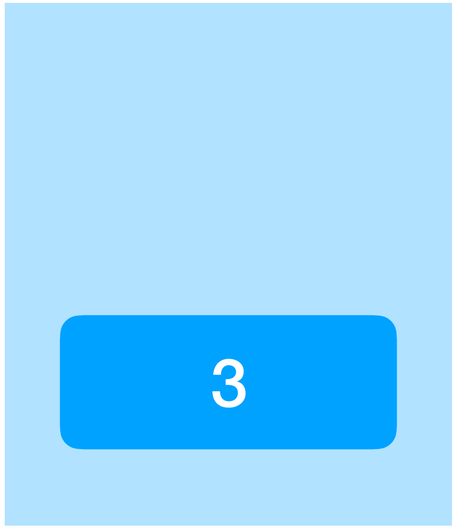
스택 상단에서 값 2개를 가져와 합산합니다.

자바의 모든 명령은 닷넷과 달리 타입 의존적입니다.

안드로이드 빌드의 특수성

스택기반의 자바 VM, 1+2

합산된 값 3을 스택에 넣습니다.



istore_1

1을 스택에 넣습니다.

상수 -1부터 5까지는 별도의 istore 명령이 있습니다.

istore_2

2를 스택에 넣습니다.

iadd

스택 상단에서 값 2개를 가져와 합산합니다.

자바의 모든 명령은 닷넷과 달리 타입 의존적입니다.

안드로이드 빌드의 특수성

스택기반의 자바 VM, 1+2

istore_1
1을 스택에.

istore_2
2를 스택에.

iadd
상단 값 2개를 합산.
닷넷과 달리 타입 의존적

낯설지만 x86이나 ARM의 기계어 코드보다는 쉬움.

https://docs.oracle.com/javase/specs/jvms/se8/html/jvms-6.html

안드로이드 빌드의 특수성

레지스터 방식의 ART, 1+2

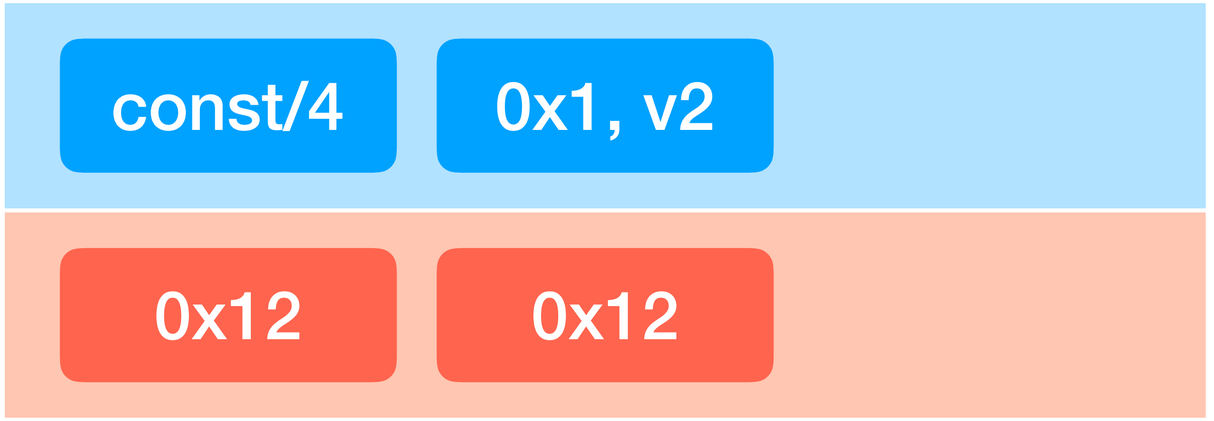
opcodes 목록

http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html

const/4 v2, 0x1

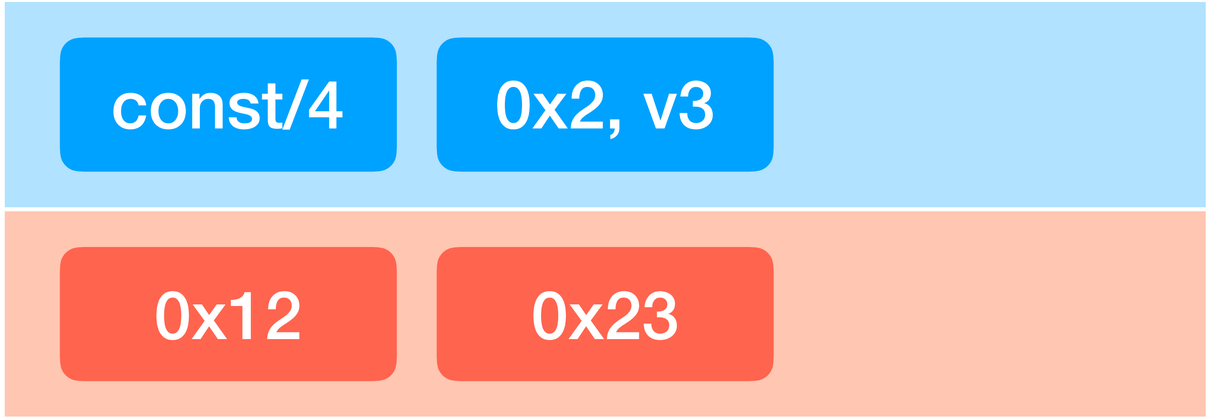
레지스터 v2에 1을 넣습니다.

기존의 PC에서 방식과 흡사합니다.



const/4 v3, 0x2

레지스터 v3에 2를 넣습니다.



왜 레지스터 기반일까? (루머)

자바 VM의 특허들을 피하기 위해?

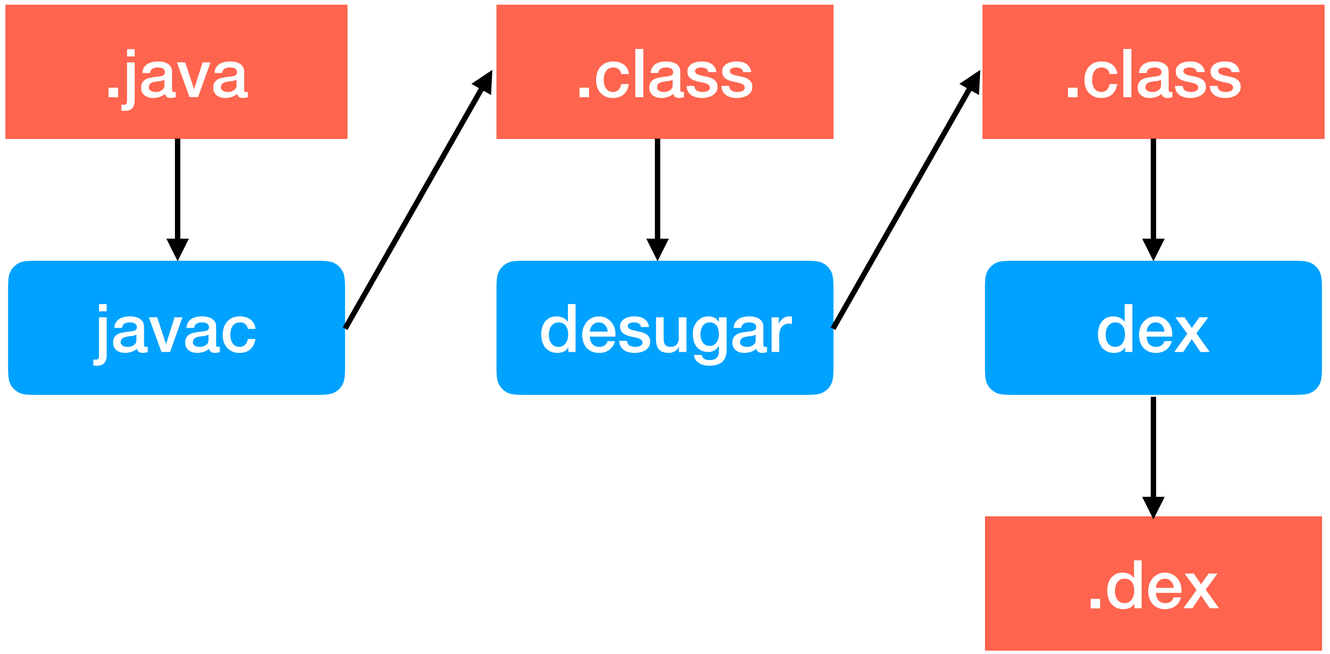
CPU와 모델이 유사해 최적화가 쉬워서?

안드로이드 빌드의 특수성

Java 바이트코드를 ART의 Dalvik 바이트코드로

단순화하여 나타낸 도식

실제 빌드 과정은 훨씬 복잡

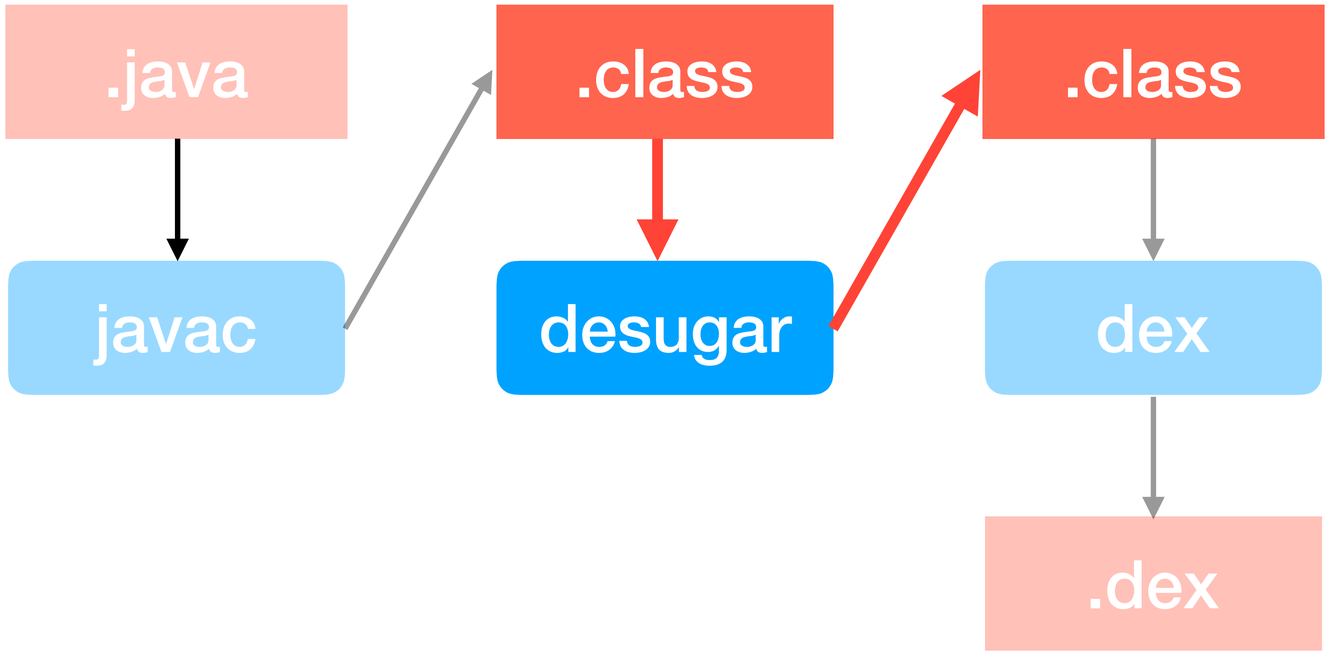


안드로이드 빌드의 특수성

Java 바이트코드를 ART의 Dalvik 바이트코드로

자바 8의 코드를 자바 7로 변환

Dalvik과 ART에서 람다 등을 해석할 수 있도록.

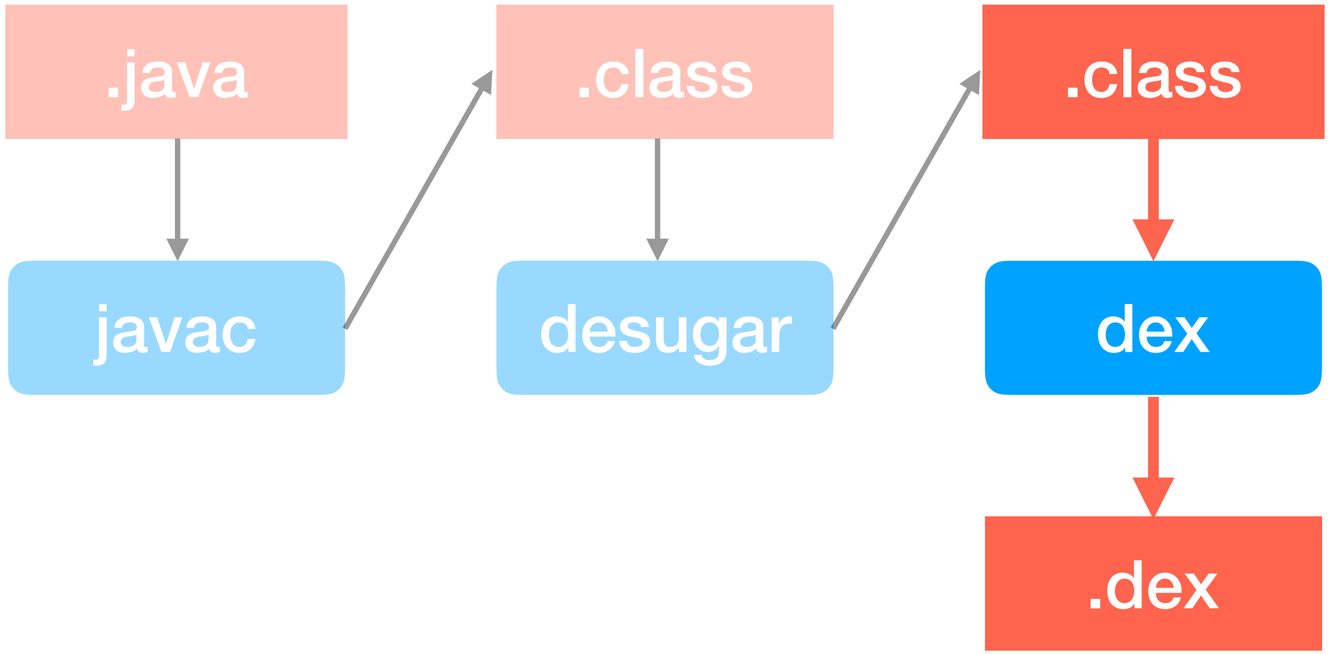


안드로이드 빌드의 특수성

Java 바이트코드를 ART의 Dalvik 바이트코드로

자바 바이트 코드를 달빅 바이트코드로 변환

이 과정은 D8(AS 3.0+)이나 DX(AS 2.X)가 수행.



안드로이드 빌드의 특수성

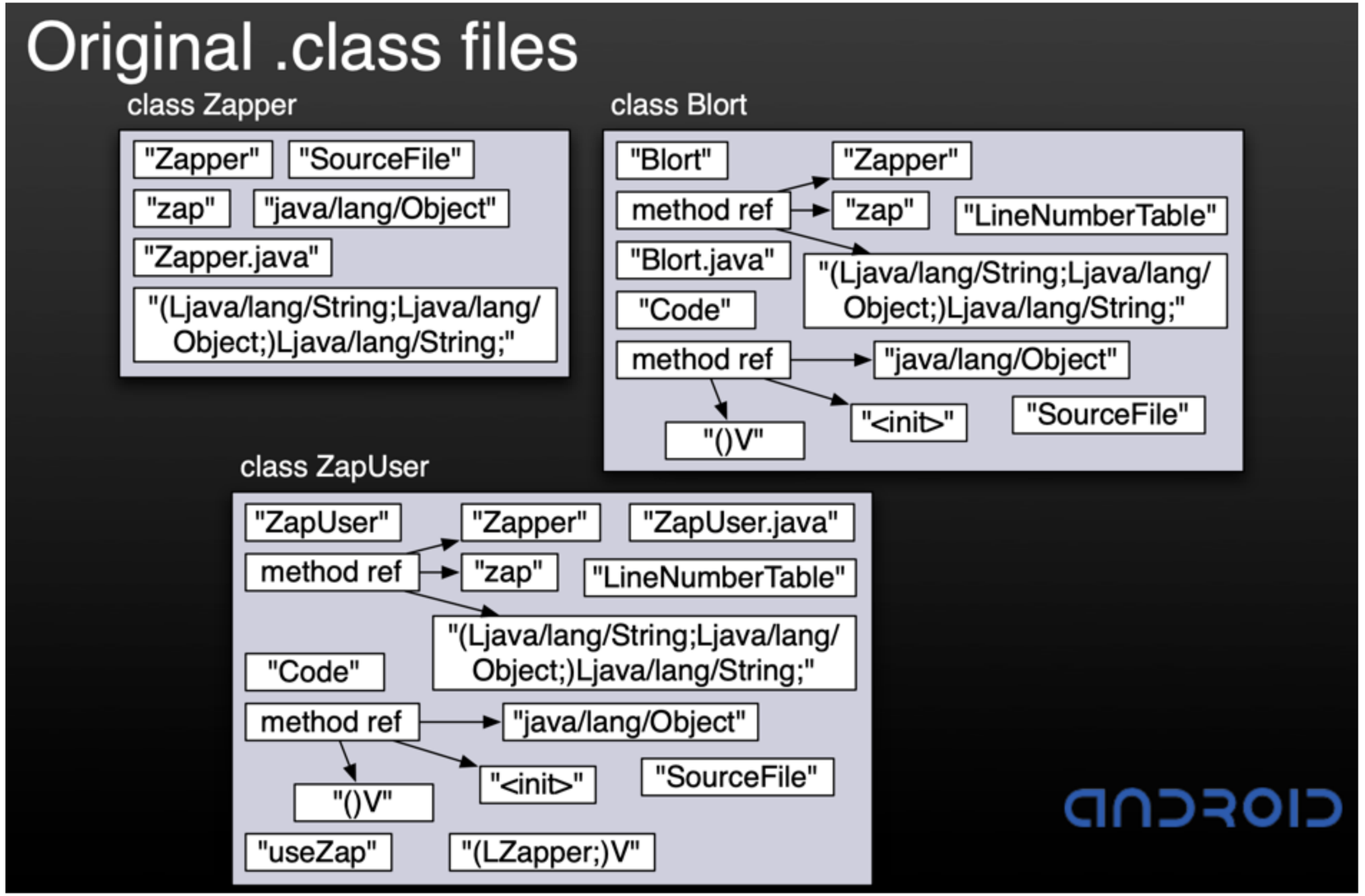
DEX가 뭐에요? (1)

자바

클래스 별로 별도의 파일로구성.

상수 역시 클래스 별로 분리되어 있음.

자바 바이트 코드로 작성됨.



안드로이드 빌드의 특수성

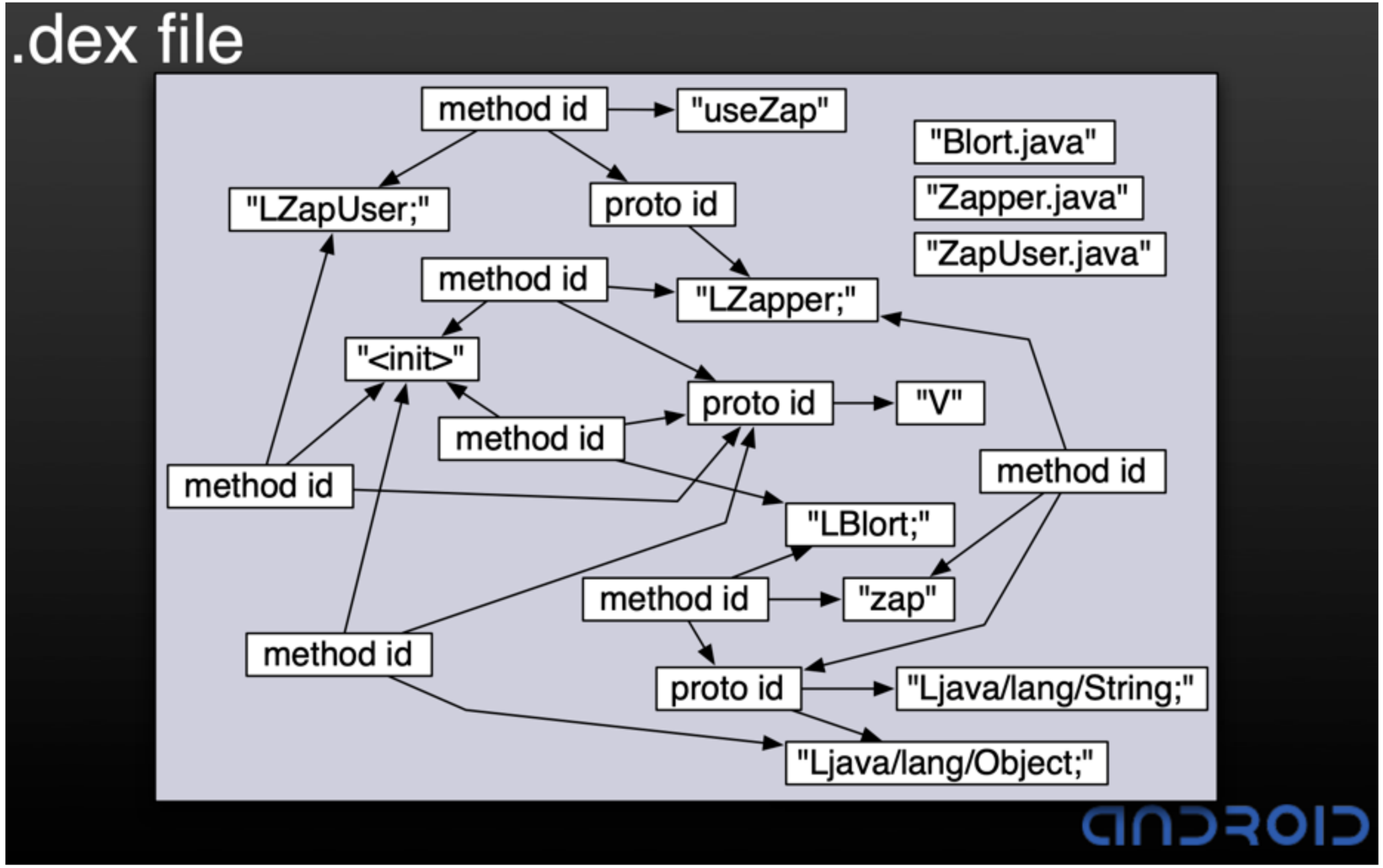
DEX가 뭐에요? (2)

DEX

64K 메서드 단위로 여러 클래스가 하나의 파일에 합쳐져 있음.

상수들은 같은 DEX내에서 공유함.
달빅 바이트 코드를 사용.

LEB-128이나 상대주소를 사용.



안드로이드 빌드의 특수성

한번에 DEX로 가면 안되요?

Jack & Jill

구글은 한번에 DEX로 가는 툴체인을 준비.

하지만 망했습니다.

생태계가 호환되지 않은 문제.

구글의 신기술은 보수적으로 도입합시다.

(렌더스크립트에 장시간을 투자한 1인. T.T)

We initially tested adding Java 8 support via the Jack toolchain. Over time, we realized the cost of switching to Jack was too high for our community when we considered the annotation processors, bytecode analyzers and rewriters impacted. Thank you for trying the Jack toolchain and giving us great feedback. You can continue using Jack to build your Java 8 code until we release the new support. Migrating from Jack should require little or no work.

02 안드로이드 빌드 과정

안드로이드 빌드 과정

그래들 플러그인

플러그인은 다음의 역할을 합니다.

1. 태스크 추가해서 빌드 과정에 추가적인 일을 할 수 있다.
2. 빌드를 위한 여러가지 설정들을 추가할 수 있다.
3. 트랜스폼을 추가해서 여러 후처리를 할 수 있다.

플러그인은 다음의 형태로 호출한다.

apply plugin: 'com.android.application'

안드로이드 빌드 과정

com.android.application 플러그인

1. 프로퍼티 파일에 지정된 플러그인을 실행

apply plugin:
'com.android.applcation'

→

implementation-class=com.android.build.gradle.AppPlugin

build.gradle 파일

com.android.application.properties 파일

2. Entry 메서드인 apply 메서드를 실행

↓

com.android.build.gradle.AppPlugin#apply 실행

com.android.build.gradle.AppPlugin 객체

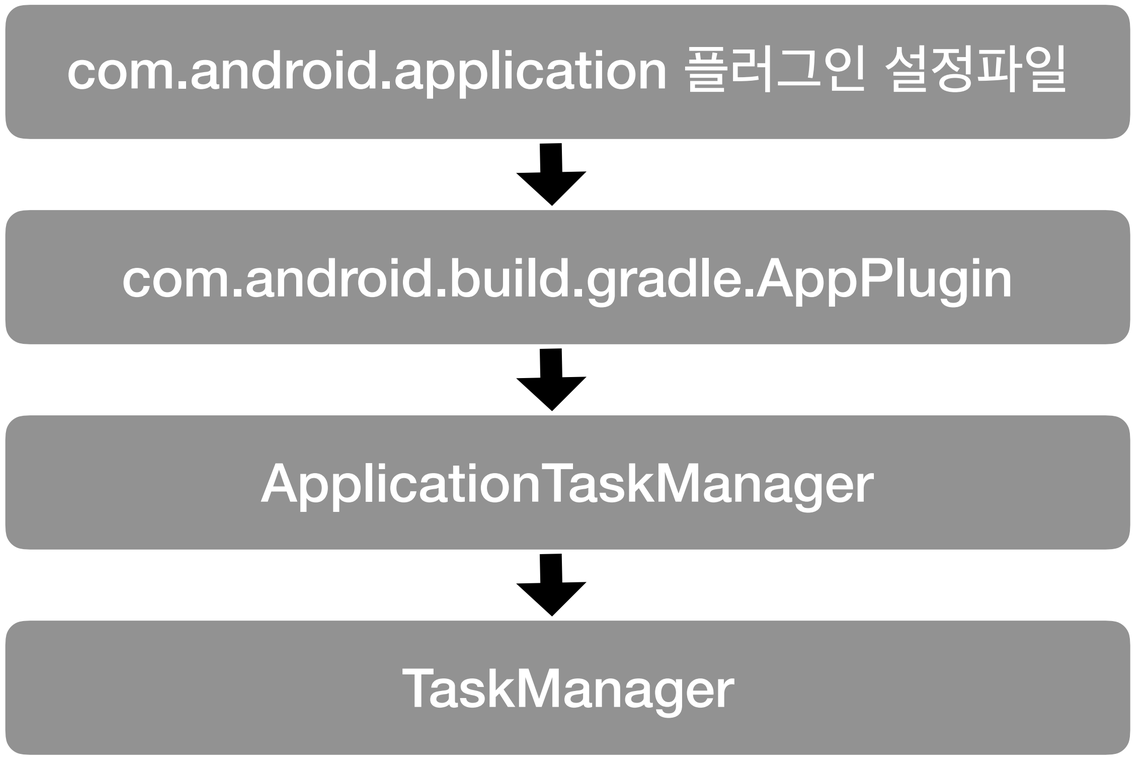
3. 트랜스폼과 환경 설정

↓

태스크 추가, 환경 설정, 트랜스폼 추가

안드로이드 빌드 과정

안드로이드 빌드 과정



플러그인 설정

com.android.application 플러그인에 대한 설정.

AppPlugin

Application 빌드를 담당하는 플러그인.

Extension을 만들며 ApplicationTaskManager로 연결.

ApplicationTaskManager

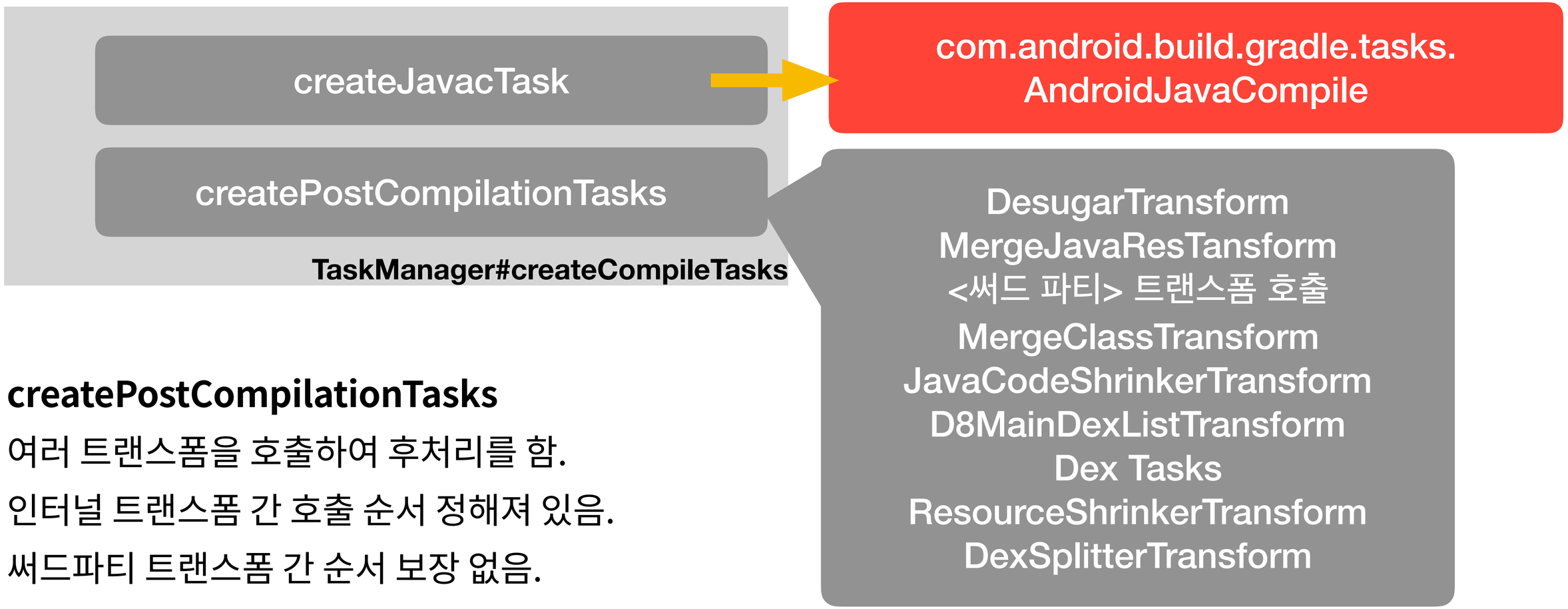
Task를 만들며 TaskManager로 연결.

TaskManager

자바 컴파일, 포스트 컴파일레이션 작업 (Transform)

안드로이드 빌드 과정

TaskManager



createPostCompilationTasks

여러 트랜스폼을 호출하여 후처리를 함.

인터널 트랜스폼 간 호출 순서 정해져 있음.

써드파티 트랜스폼 간 순서 보장 없음.

03 Transform 설정

Transform 설정

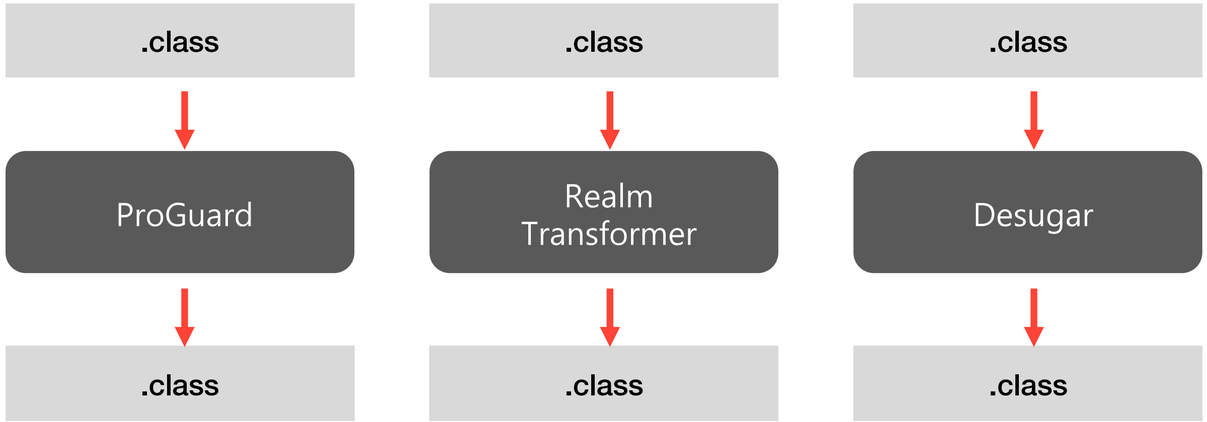
자바 VM의 생태계를 인정하고 확장성을 열자.

Transform API

다양한 후처리를 표준화된 방식으로

Proguard, Desugar, ShrinkResource 등의
공식적인 Transform이 빌드툴에 포함.

Realm과 같은 써드파티도 바이트코드 변조를
Transform API에 의존.



Transform 설정

트랜스폼 등록하기

android.registerTransform(new io.realm.transformer.RealmTransformer(project))

써드 파티는 android.registerTransform()을 통해

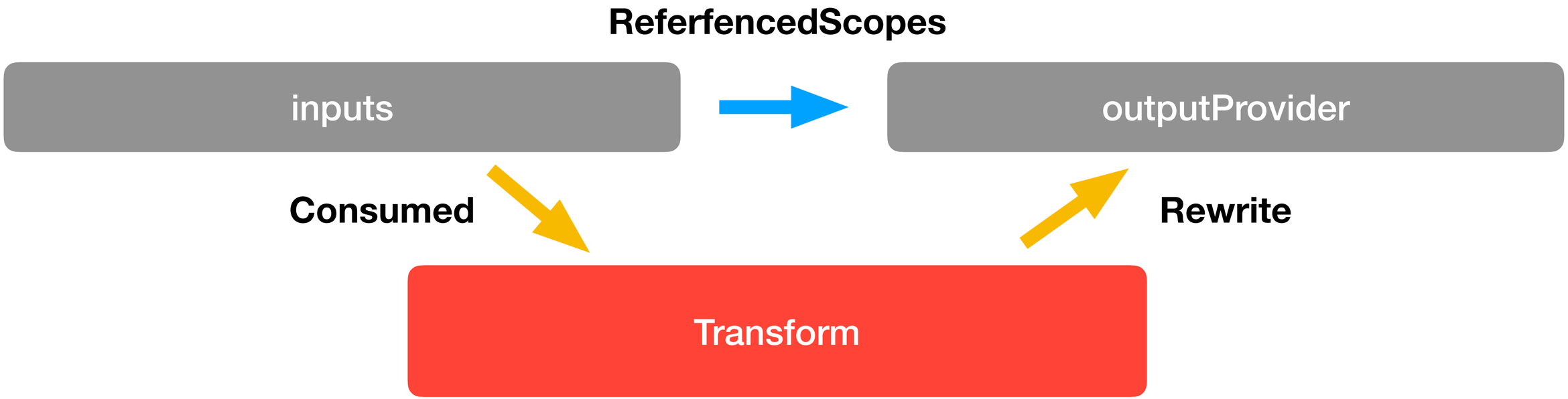
Transform 설정

소비 (consumed)

getInputTypes와 getScopes로 지정된 inputs은 Transform에 전달 후 사라짐.

Transform에서 변환된 파일을 만들어 outputProvider에 저장해야 함.

참고만 할 내용은 ReferencedScopes로 지정해야.



Transform 설정

이름, 입/출력의 유형을 정의

getName

트랜스폼의 이름

getInputTypes

트랜스 폼이 소비(consumed)하는 타입.

하나 이상 설정 가능.

getOutputTypes

트랜스폼의 출력으로 하나 이상 설정 가능.

설정값은 getInputTypes와 마찬가지로

QualifiedContent.DefaultContentType.CLASSES와 **RESOURCES**로 설정.

getScopes

이 트랜스폼에서 소비(Consumed)하는 Scope.

다른 트랜스폼에서 사용하길 원한다면

getReferencedScopes를 사용한다.

- EXTERNAL_LIBRARIES - 외부 라이브러리만
- PROJECT - 프로젝트만
- PROVIDED_ONLY - provided 만
- SUB_PROJECTS - 서브 프로젝트만
- TESTED_CODE - 테스트 된 것 만

getParameterInputs

입력 받고 싶은 파라미터로 Map을 설정.

예: TransformManager.CONTENT_CLASS

Transform 설정

이름, 입/출력의 유형을 정의한 예: DesugarTransform

람다 코드를 JVM 구버전에서 돌게 만드는 DesugarTransform

서브 프로젝트, 외부 라이브러리의 클래스 파일을 읽고

provided나 tested는 참고만 하고 클래스 파일을 출력합니다.

getName

트랜스폼의 이름:

desugar

getInputTypes

트랜스 폼이 소비(consumed)하는 타입으로 하나 이상 설정가능.

QualifiedContent.DefaultContentType.CLASSES와 RESOURCES로 설정.

TransformManager.CONTENT_CLASS (CLASSES를 담음.)

getOutputTypes

트랜스폼의 출력으로 하나 이상 설정 가능.

QualifiedContent.DefaultContentType.CLASSES와 RESOURCES로 설정.

없음. (없으면 getInputTypes를 반환)

getParameterInputs

입력 받고 싶은 파라미터로 Map을 설정.

"Min sdk" <- minSdk

getScopes

이 트랜스폼에서 소비(Consumed)하는 Scope. 다른 트랜스폼에서 사용하길 원한다면 getReferencedScopes를 사용한다.

- EXTERNAL_LIBRARIES - 외부 라이브러리만
- PROJECT - 프로젝트만
- PROVIDED_ONLY - provided 만
- SUB_PROJECTS - 서브 프로젝트만
- TESTED_CODE - 테스트 된 것 만

TransformManager.CONTENT_CLASS

(PROJECT + SUB_PROJECTS + EXTERNAL_LIBRARIES)

getReferencedScopes

PROVIDED_ONLY + TESTED_CODE

Transform 설정

이름, 입/출력의 유형을 정의한 예: ProGuardTransform

getName
proguard

안드로이드 앱에 프로가드를 적용시키는 트랜스폼.
프로젝트, 서브 프로젝트, 라이브러리의 클래스와 리소스를 받고
Provided, 테스트, 서버 프로젝트, 외부 라이브러리를 참고.

getInputTypes
QualifiedContent.DefaultContentType.CLASSES와 RESOURCES로 설정.
TransformManager.CONTENT_CLASS (CLASSES를 담음.)

getOutputTypes
QualifiedContent.DefaultContentType.CLASSES와 RESOURCES로 설정.
없음. (없으면 getInputTypes를 반환)

getParameterInputs
입력 받고 싶은 파라미터로 Map을 설정.
"shrink" <- configuration.shrink
"obfuscate" <- configuration.obfuscate
"optimize" <- configuration.optimize

getScopes
TransformManager. SCOPE_FULL_PROJECT
(PROJECT + SUB_PROJECTS + EXTERNAL_LIBRARIES)

TransformManager. SCOPE_FULL_WITH_FEATURES,
(Scope.PROJECT + InternalScope.FEATURES)

TransformManager. SCOPE_FULL_LIBRARY_WITH_LOCAL_JARS,
(Scope.PROJECT + InternalScope.LOCAL_DEPS)

getReferencedScopes
PROVIDED_ONLY + TESTED_CODE,
PROVIDED_ONLY + SUB_PROJECT + EXTERNAL_LIBRARIES

Transform 설정

이름, 입/출력의 유형을 정의한 예: RealmTransformer

오픈소스 Realm의 트랜스폼

getName

RealmTransformer

getInputTypes

QualifiedContent.DefaultContentType.CLASSES와 RESOURCES로 설정.

QualifiedContent.DefaultContentType.CLASSES

getOutputTypes

QualifiedContent.DefaultContentType.CLASSES와 RESOURCES로 설정.

없음. (없으면 getInputTypes를 반환)

getScopes

QualifiedContent.Scope.PROJECT

getReferencedScopes

EXTERNAL_LIBRARIES + PROJECT_LOCAL_DEPS + SUB_PROJECTS + SUB_PROJECTS_LOCAL_DEPS + TESTED_CODE

Transform 설정

이름, 입/출력의 유형을 정의한 예: ShrinkResourcesTransform

getName
shrinkRes

getInputTypes
ExtendedContentType.DEX, DefaultContentType.CLASSES
(DEX 읽기는 비공개 기능.)

getOutputTypes
비어있음. (getScopes도 비어있을거라 예상 할 수 있음.)

getParameterInputs
입력 받고 싶은 파라미터로 Map을 설정.
aaptOptions, variantType, isDebuggableBuildType, splitHandlingPolicy

사용하지 않는 리소스를 정리하는 안드로이드 트랜스폼.
비표준 방식으로 다룬다.

getScopes
역시 비어있음. 입력에서 어떤 것도 소비하지 않음.

입력을 소비하지 않는데 어떻게 리소스를 변조할까?
비표준적인 방식으로 그래들의 VariantScope을 직접 얻어 수정.

getReferencedScopes
TransformManager.SCOPE_FULL_PROJECT
(PROJECT + Scope.SUB_PROJECTS + EXTERNAL_LIBRARIES)

Transform 설정

Transform 객체의 Entry 메서드 transform

Transform을 상속받아야 함.

```
public class ShrinkResourcesTransform extends Transform {
    @Override
    public void transform(@NonNull TransformInvocation invocation) {
        ...
    }
}

public interface TransformInvocation {
    Context getContext();
    Collection<TransformInput> getInputs();
    @NonNull Collection<TransformInput> getReferencedInputs();
    @NonNull Collection<SecondaryInput> getSecondaryInputs();
    TransformOutputProvider getOutputProvider();
    boolean isIncremental();
}
```

transform 메서드가 Transform의 Entry.
원하는 작업을 모두 이 메서드에서 수행해야 함.

getInputs - getScope와 getInputType에 맞는 입력.
getReferencedInputs - getReferencedScope로 지정된 입력.
getSecondaryInputs - 이전 트랜스폼 이후 (증분 빌드등)을 위한 입력.
getOutputProvider - 출력 파일 작성을 위한 프로바이더.
isIncremental - 증분 빌드 여부.

Transform 설정

getInputs나 getReferences 다루기

```
@Override
public void transform(@NonNull TransformInvocation invocation) {
   Collection<TransformInput> referencedInputs = invocation.getReferencedInputs();
   List<File> classes = new ArrayList<>();
   for (TransformInput transformInput : referencedInputs) {
      for (DirectoryInput directoryInput : transformInput.getDirectoryInputs()) {
         classes.add(directoryInput.getFile());
      }
      for (JarInput jarInput : transformInput.getJarInputs()) {
         classes.add(jarInput.getFile());
      }
   }
   ...
}
```

getRefencedInputs를 호출해 참조용 입력만

입력은 DirectoryInput들과 JarInput로 분리

디렉토리와 Jar의 클래스를 다 모음

구글이 만든 트랜스폼

01 DesugarTransform: 문법 설탕등을 제거.

02 ProguardTransform: 난독화를 위한 트랜스폼.

03 ShrinkResourcesTransform: 사용하지 않는 리소스 제거.

01 DesugarTransform

DesugarTransform

transform 메서드

```
@Override
public void transform(@NonNull TransformInvocation transformInvocation)
      throws TransformException, InterruptedException, IOException {
   try {
      ...
      if (enableGradleWorkers) {
         processNonCachedOnesWithGradleExecutor(
               transformInvocation.getContext().getWorkerExecutor(), processArgs);
      } else {
         processNonCachedOnes(processArgs);
      }
      ...
   }
   ...
}
```

결국 executor를 통해
DesugarWorkerItem의
DesugarAction을 호출하는 구조.

DesugarTransform

DesugarAction

```
private static final String DESUGAR_MAIN = "com.google.devtools.build.android.desugar.Desugar";

public static class DesugarAction implements Runnable {
  ...
  @Override
  public void run() {
    try {
      ...
      Class<?> clazz = Class.forName(DESUGAR_MAIN);
      Method mainMethod = clazz.getMethod("main", String[].class);
      mainMethod.setAccessible(true);
      mainMethod.invoke(null, (Object) args.toArray(new String[0]));
    } catch (Exception e) {
      LOGGER.error("Error while running desugar ", e);
    }
  }
}
```

DESUGAR_MAIN을 통해
람다 등의 코드를 변형.

많은 작업들이 트랜스폼 내부가 아닌
외부 코드에서 수행 됨.

02 ProguardTransform

ProguardTransform

transform 메서드

```
@Override
public void transform(@NonNull final TransformInvocation invocation) throws TransformException {
  try {
    getWorkLimiter()
        .limit(
            () -> {
              doMinification(
                  invocation.getInputs(),
                  invocation.getReferencedInputs(),
                  invocation.getOutputProvider());
              ...
              return null;
            });
  ...
}
```

Semaphore로 구현된 자체 워커에서
doMinification을 수행.

ProguardTransform

WorkLimiter 메서드

```
class WorkLimiter @VisibleForTesting internal constructor(concurrencyLimit: Int) {
  private val semaphore: Semaphore = Semaphore(concurrencyLimit, true)

  @Throws(InterruptedException::class)
  fun limit(task: Callable<Void>) {
    semaphore.acquire()
    try {
      task.call()
    } finally {
      semaphore.release()
    }
  }
}
```

Semaphore로 구현된 자체 워커.

ProguardTransform

doMinification 메서드

```
private void doMinification(
      @NonNull Collection<TransformInput> inputs,
      @NonNull Collection<TransformInput> referencedInputs,
      @Nullable TransformOutputProvider output)
      throws IOException {
   try {
      ...
      for (File configFile : getAllConfigurationFiles()) {
         LOG.info("Applying ProGuard configuration file {}", configFile);
         applyConfigurationFile(configFile);
      }
      ...
      runProguard();
   }
   ...
}
```

프로가드를 위한 여러 설정을 한 후

runProguard 호출.

ProguardTransform

runProguard 메서드

```
public void runProguard() throws IOException {
    new ProGuard(configuration).execute();
    fileToFilter.clear();
}
```

Proguard 객체에게 난독화를 요청.

03 ShrinkResourcesTransform

ShrinkResourcesTransform

transform(main)에서 입력 다루기

```
private static class SplitterRunnable extends BuildElementsTransformRunnable {

    @Override
    public void run() {
        ...
        ResourceUsageAnalyzer analyzer =
                new ResourceUsageAnalyzer(
                        params.sourceDir,
                        params.classes,
                        params.mergedManifest.getOutputFile(),
                        params.mappingFile,
                        params.resourceDir,
                        reportFile,
                        ResourceUsageAnalyzer.ApkFormat.BINARY);

        ...
```

SplitterRunnable 의 run()에서
ResourceUsageAnalyzer 초기화

ShrinkResourcesTransform

실제 BuildOutputs 가공은 ResourceUsageAnalyzer

```
private static class SplitterRunnable extends BuildElementsTransformRunnable {

  @Override
  public void run() {

    ...
    analyzer.analyze();
    analyzer.rewriteResourceZip(
        params.uncompressedResourceFile, params.compressedResourceFile);
  }
}
```

analyze()로 분석.

rewriteResourceZip으로 압축파일을 재생성.

ShrinkResourcesTransform

ResourceUsageAnalyzer의 분석 (analyze)

```
public class ResourceUsageAnalyzer {
  ...
  public void analyze() throws IOException, ParserConfigurationException, SAXException {
    gatherResourceValues(mResourceClassDir);
    recordMapping(mProguardMapping);
    recordUsages(mClassesJar);
    recordManifestUsages(mMergedManifest);
    recordResources(mMergedResourceDir);
    keepPossiblyReferencedResources();
    dumpReferences();
    findUnused();
  }
  ...
}
```

gatherResourceValues 리소스 값 수집.

리소스 실제 사용 기록.

로그

미사용 찾음

ShrinkResourcesTransform

ResourceUsageAnalyzer의 분석 (analyze)

```
public class ResourceUsageAnalyzer {
  ...
  public void analyze() throws IOException, ParserConfigurationException, SAXException {
    gatherResourceValues(mResourceClassDir);
    recordMapping(mProguardMapping);
    recordUsages(mClassesJar);
    recordManifestUsages(mMergedManifest);
    recordResources(mMergedResourceDir);
    keepPossiblyReferencedResources();
    dumpReferences();
    findUnused();
  }
  ...
}
```

gatherResourceValues

리소스 값 수집.

ShrinkResourcesTransform

gatherResourceValues: 리소스 값 수집

```
private void gatherResourceValues(File file) throws IOException {
    if (file.isDirectory()) {
        File[] children = file.listFiles();
        if (children != null) {
            for (File child : children) {
                gatherResourceValues(child);
            }
        }
    } else if (file.isFile() && file.getName().equals(SdkConstants.FN_RESOURCE_CLASS)) {
        parseResourceClass(file);
    }
}
```

재귀적으로 자식에 대해

리소스 클래스 파일이면?

parseResourceClass

리소스 클래스를 해석

ShrinkResourcesTransform

parseResourceClass

```
private void parseResourceClass(File file) throws IOException {
    ...
        // Find next declaration
        for (; index < length - 1; index++) {
            char c = s.charAt(index);
            if (Character.isWhitespace(c)) {
                //noinspection UnnecessaryContinue
                continue;
            } else if (c == '/') {
                char next = s.charAt(index + 1);
                if (next == '*') {
                    // Scan forward to comment end
                    end = index + 2;
                    while (end < length -2) {
                        c = s.charAt(end);
                        if (c == '*' && s.charAt(end + 1) == '/') {
                            end++;
```

손으로 한땀 한땀 딴 짠 파서.

구글도 마법은 없구나.

리소스를 찾으면 **addResource**를 호출한다.

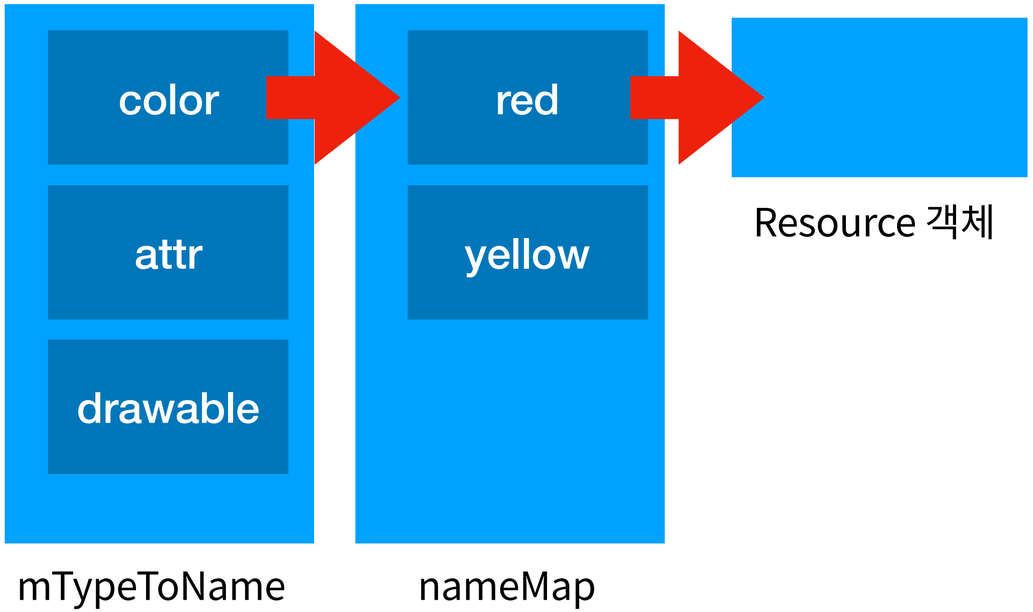
ShrinkResourcesTransform

addResource

```
private void addResource(@NonNull ResourceType type, @NonNull String name,
    @Nullable String value) {
  ...
  resource = new Resource(type, name, realValue);
  mResources.add(resource);
  if (realValue != -1) {
    mValueToResource.put(realValue, resource);
  }
  Map<String, Resource> nameMap = mTypeToName.get(type);
  if (nameMap == null) {
    nameMap = Maps.newHashMapWithExpectedSize(30);
    mTypeToName.put(type, nameMap);
  }
  nameMap.put(name, resource);
}
```

Resource 객체로 포장해

mTypeToName의 **nameMap**에 담음.



ShrinkResourcesTransform

Resource

```
public static class Resource {
    public ResourceType type;
    public String name;
    public int value;
    public boolean reachable;
    public boolean hasDefault;
    public List<Resource> references;
    public final List<File> declarations = Lists.newArrayList();
    ...
}
```

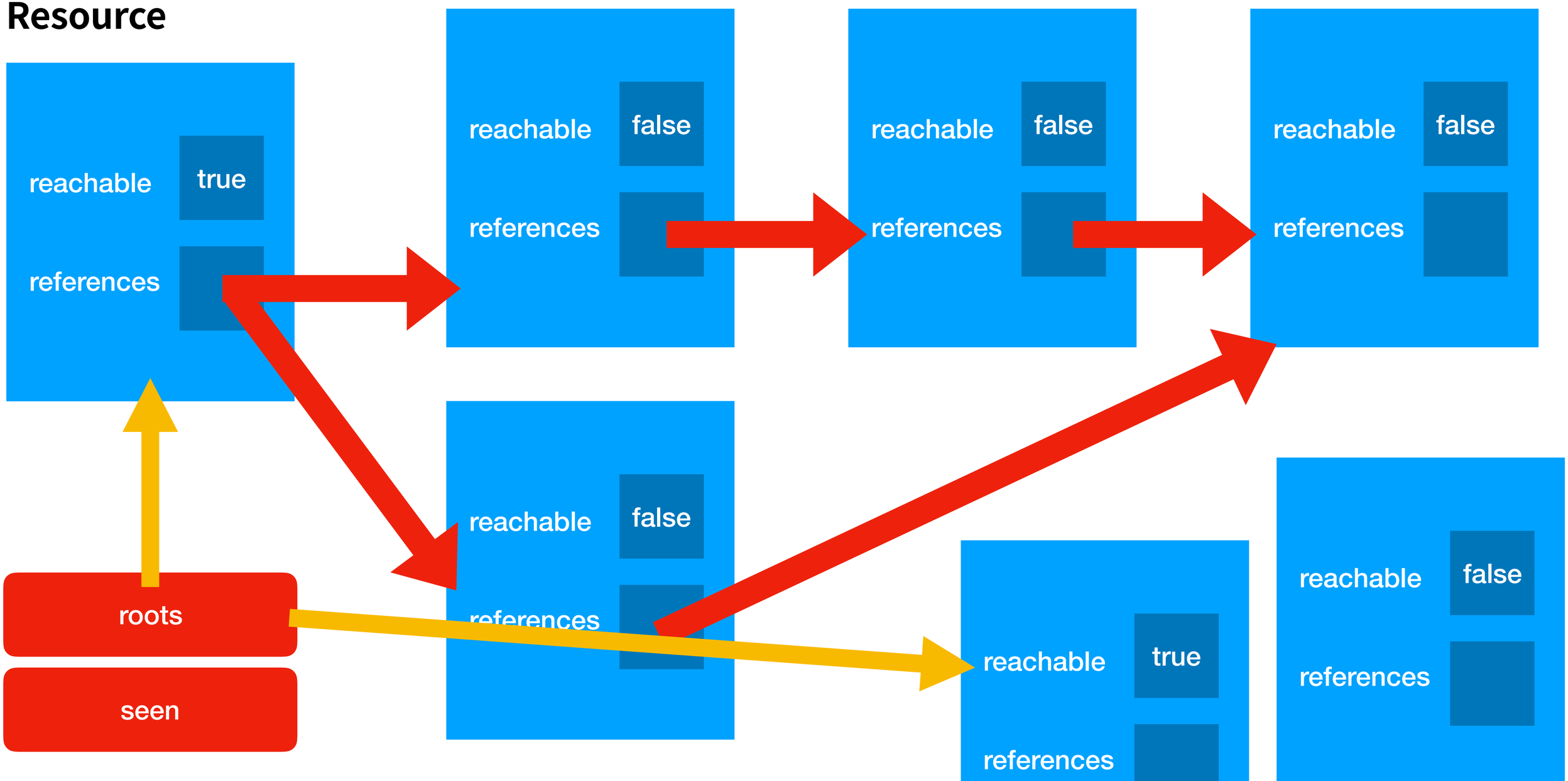
리소스 타입, 네임 등을 가지고 있음.

markReachable 메서드를 통해 클래스등에서 접근하는 경우 **reachable**을 체크.
GC의 Mark & Sweep 처럼 제거 대상을 찾음.

해당 리소스가 다른 리소스를 참조할 경우 **addReference**를 통해 **references**에 추가.
reachable한 Resource가 가진 references도 제거 대상에서 제외.

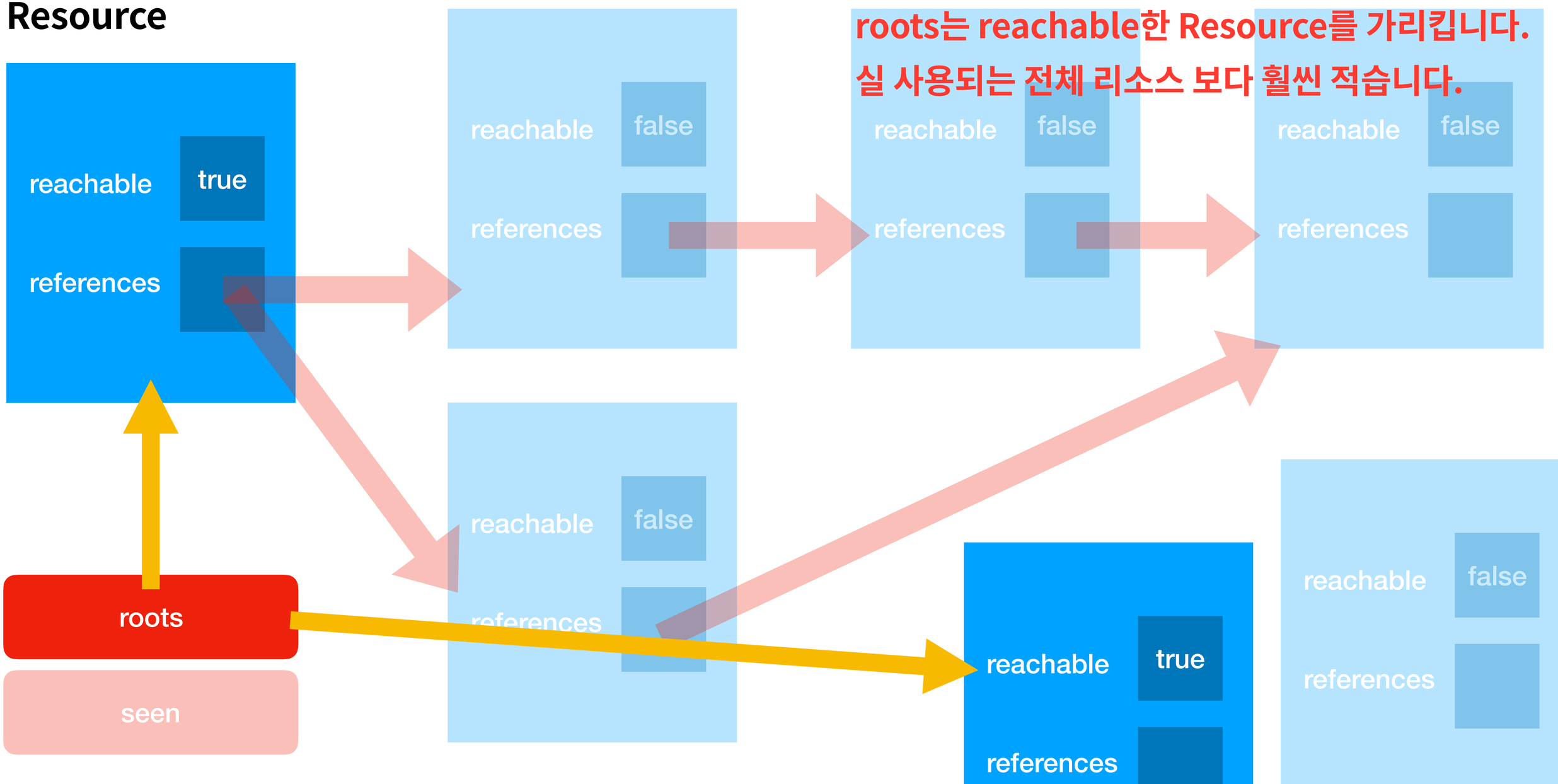
ShrinkResourcesTransform

Resource



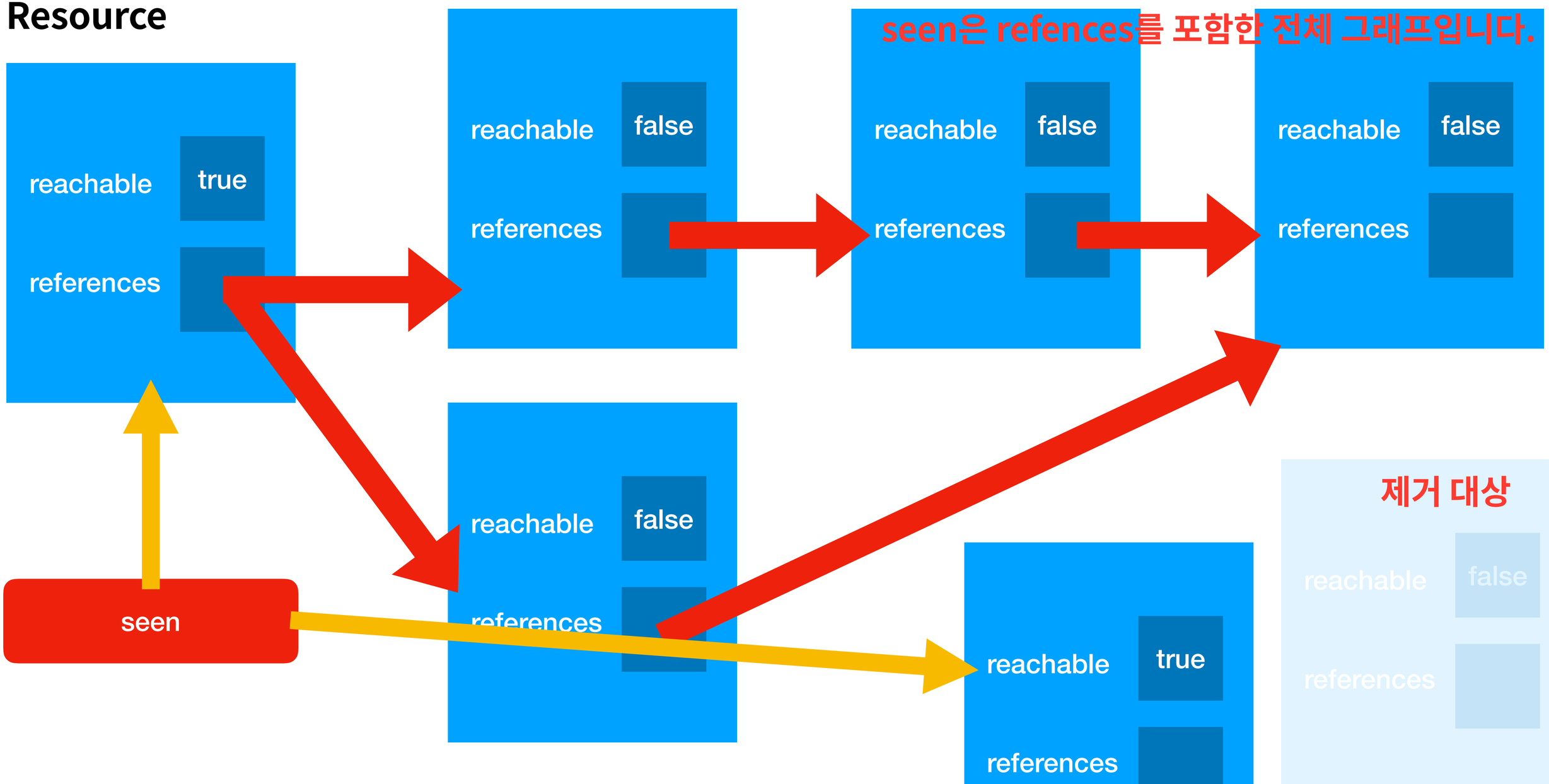
ShrinkResourcesTransform

Resource



ShrinkResourcesTransform

Resource



ShrinkResourcesTransform

ResourceUsageAnalyzer의 분석 (analyze)

```
public class ResourceUsageAnalyzer {
  ...
  public void analyze() throws IOException, ParserConfigurationException, SAXException {
    gatherResourceValues(mResourceClassDir);
    recordMapping(mProguardMapping);
    recordUsages(mClassesJar);
    recordManifestUsages(mMergedManifest);
    recordResources(mMergedResourceDir);
    keepPossiblyReferencedResources();
    dumpReferences();
    findUnused();
  }
  ...
}
```

한땀 한땀 텍스트 파싱하거나

ASM (바이트코드 조작 도구)의 **비지터로 바이트코드**로 Resource들의 **references를 갱신**.

ShrinkResourcesTransform

ResourceUsageAnalyzer의 분석 (analyze)

```
public class ResourceUsageAnalyzer {
  ...
  public void analyze() throws IOException, ParserConfigurationException, SAXException {
    gatherResourceValues(mResourceClassDir);
    recordMapping(mProguardMapping);
    recordUsages(mClassesJar);
    recordManifestUsages(mMergedManifest);
    recordResources(mMergedResourceDir);
    keepPossiblyReferencedResources();
    dumpReferences();
    findUnused();
  }
  ...
}
```

루트부터 references를 순회하며 사용하지 않는 리소스를 찾음.

ShrinkResourcesTransform

findUnused

```
private void findUnused() {
    for (Resource resource : mResources) {
        if (resource.reachable && resource.type != ResourceType.ID
                && resource.type != ResourceType.ATTR) {
            roots.add(resource);
        }
    }
    for (Resource root : roots) {
        visit(root, seen);
    }
    for (Resource resource : mResources) {
        if (!resource.reachable && resource.isRelevantType()) {
            unused.add(resource);
        }
    }
}
```

reachable한 리소스를 **roots**에 등록.

roots의 요소를 방문하며 **references**에 들어있는 Resource를 재귀적으로 **seen**에 추가.

전체 리소스를 순회하며 **unused**를 갱신.

ShrinkResourcesTransform

실제 BuildOutputs 가공은 ResourceUsageAnalyzer

```
private static class SplitterRunnable extends BuildElementsTransformRunnable {

  @Override
  public void run() {

    ...
    analyzer.analyze();
    analyzer.rewriteResourceZip(
        params.uncompressedResourceFile, params.compressedResourceFile);
  }
}
```

unused 리소스 파일을 제외한 리소스 ZIP을 생성.

04 Transform 실습

Transform 실습

로그를 남겨 봅시다.

메서드를 진입할 때 마다 로그를 남겨봅시다.

구글이 종종 사용하는 ASM대신 Javassist

- ASM은 로우레벨
- Javassist는 하이레벨

Transform 실습

outputProvider 구성

```
override fun transform(transformInvocation: TransformInvocation) {
    val outputDir = transformInvocation.outputProvider.getContentLocation(
        "classes",
        outputTypes,
        scopes,
        Format.DIRECTORY
    )
    …
}
```

outputProvider로 부터 파일의 경로를 얻어냄.

Transform 실습

outputProvider 구성

```
override fun transform(transformInvocation: TransformInvocation) {
  ...
  transformInvocation.inputs.forEach { transformInput ->
    transformInput.directoryInputs.forEach { inputDirectory ->
      inputDirectory.file.walkTopDown().forEach { originalClassFile ->
        if (originalClassFile.isClassfile()) {
          val classname = originalClassFile.relativeTo(inputDirectory.file).toClassname()
          val clazz = pool.get(classname)
          clazz.declaredMethods.forEach { method ->
            method.insertBefore(
              "{android.util.Log.d("${clazz.name}","${method.name} entered.");}"
            )
          }
          clazz.writeFile(outputDir.absolutePath)
        }
      }
    }
```

transformInvocation.inputs

입력 파일을 찾아서 순회.

transformInput.directoryInputs

디렉토리를 찾아 순회.

Transform 실습

outputProvider 구성

```
override fun transform(transformInvocation: TransformInvocation) {
    ...
            val classname = originalClassFile.relativeTo(inputDirectory.file).toClassname()
            val clazz = pool.get(classname)
            clazz.declaredMethods.forEach { method ->
              method.insertBefore(
                "{android.util.Log.d("${clazz.name}","${method.name} entered.");}"
              )
            }
            clazz.writeFile(outputDir.absolutePath)
    ...
}
```

pool.get로 Javassist 클래스 접근.

Transform 실습

outputProvider 구성

```
override fun transform(transformInvocation: TransformInvocation) {
    ...
            clazz.declaredMethods.forEach { method ->
              method.insertBefore(
                "{android.util.Log.d("${clazz.name}","${method.name} entered.");}"
              )
    ...
}
```

declaredMethods로 정의된 메서드 접근

insetBefore로 코드 삽입

Transform 실습

outputProvider 구성

```
override fun transform(transformInvocation: TransformInvocation) {
    ...
            clazz.writeFile(outputDir.absolutePath)
    ...
}
```

clazz.writeFile로 Javassist 클래스 저장.

06 정리

후처리를 알면 무엇이 도움이 되나요?

01 보안을 강화할 수 있습니다.

02 언어의 한계를 극복할 수 있습니다.

03 APT의 한계도 극복할 수 있습니다.

04 반복적인 작업을 자동화를 할 수 있습니다.

05 ~~아는 척을 할~~ 교양을 쌓을 수 있습니다.

세션에서 다루었던 내용과 향후 과제

01 안드로이드 빌드 과정의 특이성과 흐름을 배웠습니다.

02 PostCompilation 과정과 작성 방법을 배웠습니다.

03 어렵지만 자신의 분야와 접목해서 가능성을 찾아보십시요.

Fin. 감사합니다.