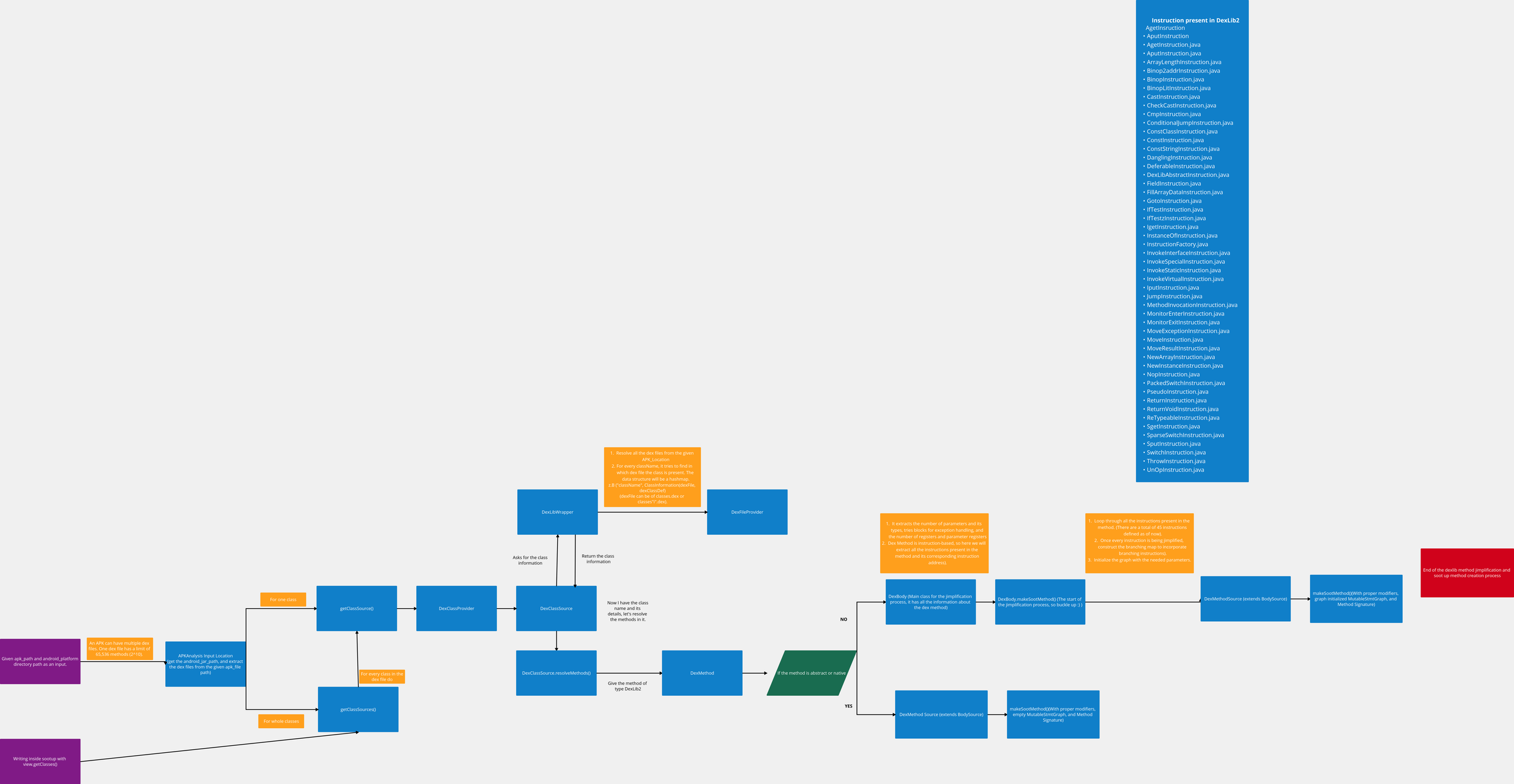
Instruction present in DexLib2
AgetInsruction
• AputInstruction
• AgetInstruction.java
• AputInstruction.java
• ArrayLengthInstruction.java
• Binop2addrInstruction.java
• BinopInstruction.java
• BinopLitInstruction.java
• CastInstruction.java
• CheckCastInstruction.java
• CmpInstruction.java
• ConditionalJumpInstruction.java
• ConstClassInstruction.java
• ConstInstruction.java
• ConstStringInstruction.java
• DanglingInstruction.java
• DeferableInstruction.java
• DexLibAbstractInstruction.java
• FieldInstruction.java
• FillArrayDataInstruction.java
• GotoInstruction.java
• IfTestInstruction.java
• IfTestzInstruction.java
• IgetInstruction.java
• InstanceOfInstruction.java
• InstructionFactory.java
• InvokeInterfaceInstruction.java
• InvokeSpecialInstruction.java
• InvokeStaticInstruction.java
• InvokeVirtualInstruction.java
• IputInstruction.java
• JumpInstruction.java
• MethodInvocationInstruction.java
• MonitorEnterInstruction.java
• MonitorExitInstruction.java
• MoveExceptionInstruction.java
• MoveInstruction.java
• MoveResultInstruction.java
• NewArrayInstruction.java
• NewInstanceInstruction.java
• NopInstruction.java
• PackedSwitchInstruction.java
• PseudoInstruction.java
• ReturnInstruction.java
• ReturnVoidInstruction.java
• ReTypeableInstruction.java
• SgetInstruction.java
• SparseSwitchInstruction.java
• SputInstruction.java
• SwitchInstruction.java
• ThrowInstruction.java
• UnOpInstruction.java
Given apk_path and android_platform directory path as an input.
An APK can have multiple dex files. One dex file has a limit of 65,536 methods (2^10).
APKAnalysis Input Location (get the android_jar_path, and extract the dex files from the given apk_file path)
For one class
For whole classes
getClassSource()
getClassSources()
For every class in the dex file do
Writing inside sootup with view.getClasses()
DexClassProvider
DexClassSource
Asks for the class information
Return the class information
DexLibWrapper
1. Resolve all the dex files from the given APK_Location
2. For every className, it tries to find in which dex file the class is present. The data structure will be a hashmap. z.B {"className", ClassInformation(dexFile, dexClassDef)
(dexFile can be of classes.dex or classes*.dex).
DexFileProvider
Now I have the class name and its details, let's resolve the methods in it.
DexClassSource.resolveMethods()
Give the method of type DexLib2
DexMethod
If the method is abstract or native
NO
YES
DexBody (Main class for the jimplification process, it has all the information about the dex method)
DexBody.makeSootMethod() (The start of the jimplification process, so buckle up :))
1. It extracts the number of parameters and its types, tries blocks for exception handling, and the number of registers and parameter registers
2. Dex Method is instruction-based, so here we will extract all the instructions present in the method and its corresponding instruction address).
1. Loop through all the instructions present in the method. (There are a total of 45 instructions defined as of now).
2. Once every instruction is being jimplified, construct the branching map to incorporate branching instructions).
3. Initialize the graph with the needed parameters.
DexMethodSource (extends BodySource)
makeSootMethod()(With proper modifiers, graph initialized MutableStmtGraph, and Method Signature)
End of the dexlib method jimplification and soot up method creation process
DexMethod Source (extends BodySource)
makeSootMethod()(With proper modifiers, empty MutableStmtGraph, and Method Signature)