

●J2KConverter の目的・使い方

Last Modified: 2023/02/15, Written by Kazuma Kohno

ここから先に記すのは J2KConverter の目的と、開発時の使い方に関するお話です。各種アルゴリズムなどプログラムについては修士論文 4 章・5 章を読んでください

修士論文タイトル

『Kotlin 特有の機能を選択的に導入できる Java-to-Kotlin コンバータの開発』

○J2KConverter の目的

変換の際に、ユーザが持っている様々な目的を考慮した変換結果を出力できるようにすること。コードの良し悪しの基準はユーザの目的に左右されるため、画一的な変換では対応しきれないと考えたことが理由

機能ごとの目的は以下の通り

- ・基本機能

Java で書かれたプログラムを Kotlin へと変換すること

- ・拡張機能

Kotlin 特有の機能や記述方法を活かした変換の提供

○J2KConverter を構成するプログラム、関連するプログラム

- ・ J2KConverter.java

変換用のプログラムのファイル。

- ・ J2KConverterSupporter.java

情報解析用のプログラムのファイル。

- ・ J2KConverterFull.java

上記 2 つのプログラムを接続するためのプログラム。

- ・ DataStore.java

J2KConverterSupporter にて解析された情報を保存しておくクラス。

- ・ TwinKeyDataList.java、Triplets.java

どちらも情報を保存するために開発されたプログラム。詳しくは 4 - 3 節情報保存参照

- ・ 末尾に Information とついた java ファイル

解析された情報を保存するための形式。詳しくは論文の 5 - 4 節参照

- ・ ClassTemporaryStore.java

情報解析の際に一時保存のために使われるクラス。

これを編集することはアルゴリズムの大きな変更を伴わない限りなさそう

- ConvertOutputer.java

変換出力を担うプログラムの入っているファイル。

- 各種.sh ファイル(シェルスクリプト)

J2KConverter を実行するための文言を記している。中身については使い方でも触れる
たくさんあるのはテスト用に使い分けているから
converter.sh が本当の実行用だと思ってくれば、他は適当に使ってもよし

○J2KConverter のテスト用のプログラム、ディレクトリなど

- AstViewer.java

JavaParser による解析で構文木の構造が見たい時に使うテスト用クラス。
ダミーの Visitor を入れています

- ディレクトリ pac3

各種記述のテスト用プログラムが入っている。詳しくは中身を見るとよし
(そこまで複雑でもないの)

- ディレクトリ pac4

pac3 のテストプログラム内で別ディレクトリが必要な場合にこちらに書く

- TrialClass.java

主に配列と数値、static クラスの変換のテスト用プログラム。いくつかあるが中身は同じ

- ディレクトリ pac1, pac2, testpac

これらもテスト用。けどもほぼ使用していないので消してもあまり問題はない
(少なくとも J2KConverter には繋がっていないのでご安心を)

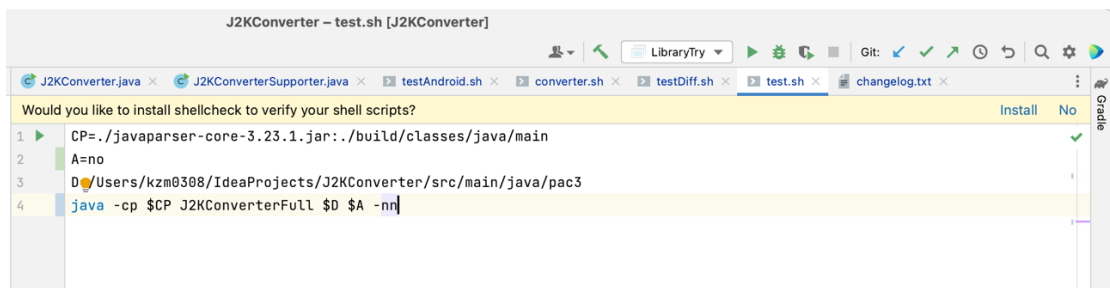
- JavaParserNodeMemo.txt

JavaParser で解析した際のノードの名前の一覧。
横にかっこ書きがあるものはどの文言に対応しているか把握済みのもの
参考資料として使ってください

○導入・使用方法

- IntelliJ IDEA 経由

1 : IntelliJ IDEA を起動し、J2KConverter のプロジェクトを開き、その中の test.sh を開く。



2 : D に変換したいプログラム群が入っているディレクトリの絶対パスを記入する

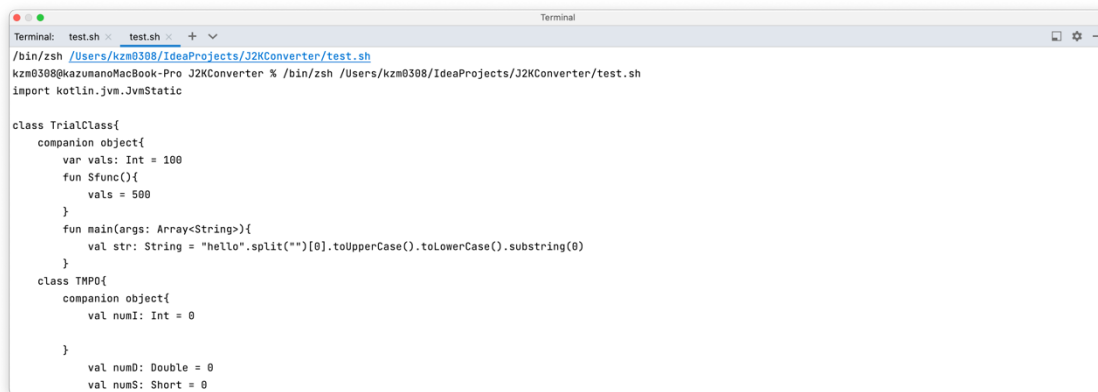
A には Android のベースライブラリのパスを記入する

(Android ではない場合は no と記入する)

D の後ろに拡張機能を使用するためのオプションを明記(全オフも可能)

CP は JavaParser ライブラリを明記

3 : 緑の矢印(Run test.sh)を押す

A screenshot of a macOS Terminal window. The title bar says "Terminal". The window has two tabs, both labeled "test.sh". The terminal content shows the execution of a script that prints the current directory, runs a Kotlin command to import a static function, and then displays the source code of two Kotlin classes: TrialClass and TMP0. TrialClass has a companion object with a val vals, a fun Sfunc, and a main function. TMP0 has a companion object with a val numI and several other val declarations.

```
Terminal: test.sh x test.sh x + v
/bin/zsh /Users/kzm0308/IdeaProjects/J2KConverter/test.sh
kzm0308@kazumanoMacBook-Pro: J2KConverter % /bin/zsh /Users/kzm0308/IdeaProjects/J2KConverter/test.sh
import kotlin.jvm.JvmStatic

class TrialClass{
    companion object{
        var vals: Int = 100
        fun Sfunc(){
            vals = 500
        }
        fun main(args: Array<String>){
            val str: String = "hello".split(" ")[0].toUpperCase().toLowerCase().substring(0)
        }
    }
}

class TMP0{
    companion object{
        val numI: Int = 0
    }

    val numD: Double = 0
    val numS: Short = 0
}
```

すると上記のように変換結果が出力される(今回は開発の都合上コマンドライン上に表示しているが、ファイルを新規に作成し、その中に書き込むこともできる)

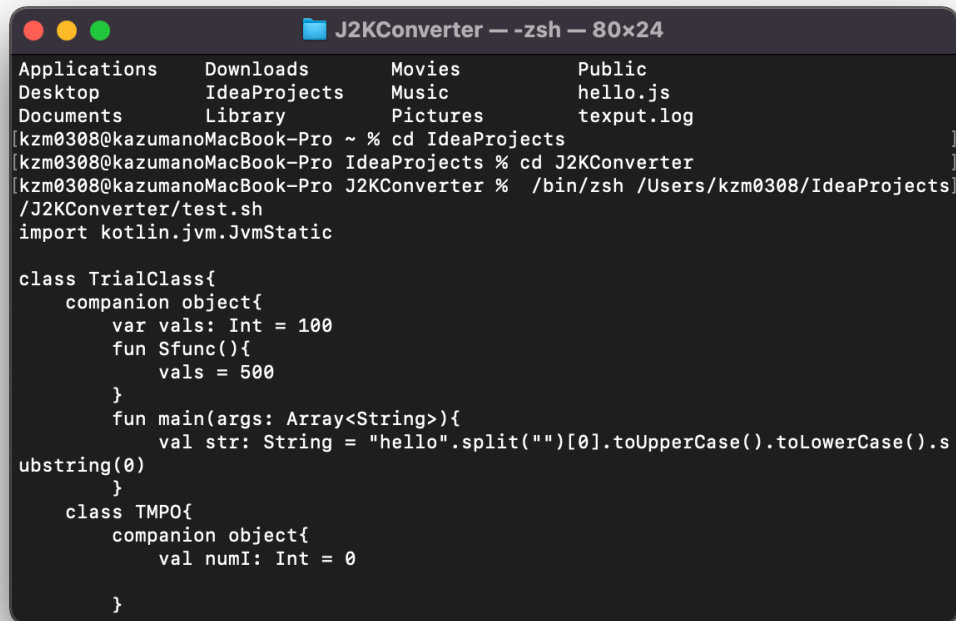
・ コマンドライン経由

1 : IntelliJ IDEA 経由の 2 番参照 (シェルスクリプトの中身の編集)

2 : J2KConverter のプロジェクトがある箇所までディレクトリを辿る

(僕の場合は cd IdeaProjects/J2KConverter でたどり着けた)

3 : /bin/zsh /Users/kzm0308/IdeaProjects/J2KConverter/test.sh と入力する

A terminal window titled "J2KConverter — zsh — 80x24" with a dark background. It shows a directory listing at the top with columns for Applications, Downloads, Movies, and Public. Below the listing, it shows the user navigating through directories and running a script. The script contains Kotlin code for a class named TrialClass, which includes a companion object with a main function and a nested class TMP0.

```
Applications  Downloads  Movies      Public
Desktop      IdeaProjects Music       hello.js
Documents    Library   Pictures    texput.log
[kzm0308@kazumanoMacBook-Pro ~ % cd IdeaProjects
[kzm0308@kazumanoMacBook-Pro IdeaProjects % cd J2KConverter
[kzm0308@kazumanoMacBook-Pro J2KConverter % /bin/zsh /Users/kzm0308/IdeaProjects
/J2KConverter/test.sh
import kotlin.jvm.JvmStatic

class TrialClass{
    companion object{
        var vals: Int = 100
        fun Sfunc(){
            vals = 500
        }
        fun main(args: Array<String>){
            val str: String = "hello".split("")[0].toUpperCase().toLowerCase().s
ubstring(0)
        }
        class TMP0{
            companion object{
                val numI: Int = 0
            }
        }
    }
}
```

実行結果は上記の通り

○プログラムの変更を行った後に実行する場合

- 1 : IntelliJ IDEA の Build から Build Project をクリック
- 2 : 完了後、変更が反映された状態で実行可能

これをしないと変更が反映されていない状態での実行になるので注意(実装ミスを疑って時間をロスすることになりかねないので徹底)

○拡張機能の実装状況(かっこの中はオプション)

- 1 : non-null 化(-nn)

実装済み。必要に応じて変更求む

- 2 : カスタムアクセサ化(-ca)

変換そのものは実装済みだが、記述量の削減には至らない。また、アクセサの中身が1行だった場合の変換に改良の余地あり(動かすことは可能)

- 3 : 後置演算子の変更

未実装。構想はしている

4 ~

構想すらまだ。必要に応じて追加求む。

○デバッグのために出力先を変えたい場合

デフォルトは全てのプログラムを入力パス直下につくった `outkt` というディレクトリの下にファイルを作って出力している

・絶対パスにディレクトリを明記した、特定のプログラムだけを出力したい

- 1 : J2KConverter.java 内の図中の『出力』と書かれた for 文内部の if 文の方を使用し、その下 2 行をコメントアウトする。
- 2 : `path.equals` の引数の絶対パスを目的のプログラムのものにする
- 3 : 実行する

・ファイルを作らずにターミナルに出力したい

- 1 : J2KConverter.java 内の図中の『出力』と書かれた for 文内部の if 文の方をコメントアウトし、その下 2 行を使用する。
- 2 : 2 行のうち、`Converter.Outputter` の文をコメントアウトする
- 3 : 実行する

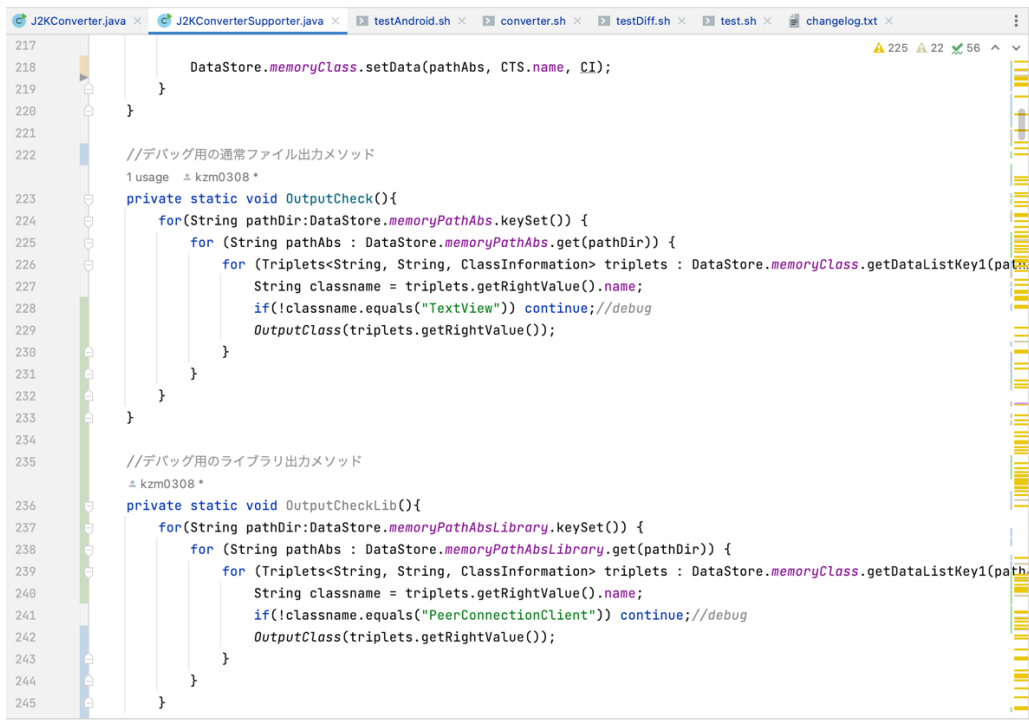
・特定のプログラムだけをターミナルに出力したい

1 : J2KConverter.java 内の図中の『出力』と書かれた for 文内部の if 文の方を使用し、その下 2 行をコメントアウトする。

2 : if 文内の Converter.Outputter の文をコメントアウトする

3 : 実行する

○各種情報が正しく取得できているかを確認したい場合



```
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245
```

OutputCheck 関数を main 関数のどこかに書く。Analysis 実行直前直後が望ましい
(実際に書いてあるのでコメントアウト解除して使って)

全てのプログラムの情報を見たい場合は、上記の図中の if(classname.equals(~~~))の行を
コメントアウトすること

どれか 1 つのクラスだけ見たい場合は、上記の図中の if(classname.equals(~~~))の行の
equals 関数の引数に適切な絶対パスを記述すること