Micro Focus Visual COBOL チュートリアル

COBOL プログラムを JVM バイトコードにコンパイルして利用する

1. 目的

Micro Focus Visual COBOL の COBOL コンパイラーは、ネイティブコード生成とは別に JVM バイトコードや CIL コードと呼ばれる.NET クラスへダイレクトに変換する機能を持っています。この機能を利用することにより複雑な計算ロジックや精度を維持するような COBOL が得意とする分野に対して既存の COBOL コードを再利用することが可能になります。生成されたクラスファイルは Java や.NET 言語を駆使する プログラマーからはあたかも既存の Java や.NET 言語で作成されたクラスファイルと同等に呼び出すことができます。

このドキュメントでは、簡単な COBOL プログラムの作成と、その後、Eclipse IDE 用の Visual COBOL を使った COBOL コードをダイレク トに JVM クラスとして生成し、Java プロジェクトからそれを利用する方法について説明します。また、作成した Java プロジェクトおよび COBOL JVM プロジェクトの内容を再構成して、JAR ファイルを作成し、Java プロジェクトにて参照、実行できるようにします。これにより COBOL コン ポーネントを利用した Java アプリケーションを任意の環境で利用できるようになります。

2. 前提条件

本チュートリアルは、下記の環境を前提に作成されています。サポートしているプラットフォームであれば Linux/UNIX でも利用可能です。

● 開発クライアント ソフトウェア

OS Windows Server 2019 Standard Edition (64bit) COBOL 開発環境製品 Micro Focus Visual COBOL 8.0J for Eclipse

チュートリアル用サンプルプログラム

下記のリンクから事前にチュートリアル用のサンプルファイルをダウンロードして、任意のフォルダに解凍しておいてください。

サンプルプログラムのダウンロード

MICRO[®] FOCUS

内容

- 1. 目的
- 2. 前提条件
- 3. チュートリアル手順の概要
 - 3.1. Windows クライアントでの開発準備作業
 - 3.2. JVM COBOL プロジェクトの作成
 - 3.3. Java プロジェクトの作成
 - 3.4. 作成した Java アプリケーションの実行
 - 3.5. COBOL パースペクティブのカスタマイズ
 - 3.6. JVM COBOL のパッケージ化
 - 3.7. プロジェクトのビルド
 - 3.8. パッケージ化されたファイルのコピー
 - 3.9. パッケージ化された Jar ファイルを使用するように Java プロジェクトを変更
 - 3.10. Java プロジェクトの実行



3. チュートリアル手順の概要

3.1. Windows クライアントでの開発準備作業

- 1) Visual COBOL for Eclipse を起動します。
 - ① [スタート] メニュー > [Micro Focus Visual COBOL] > [Visual COBOL for Eclipse] を選択します。
 - ② ワークスペースの選択画面は任意のワークスペースを指定して、[起動(L)] ボタンをクリックします。

3.2. JVM COBOL プロジェクトの作成

- 1) COBOL JVM プロジェクトを作成します。
 - ① [ファイル(F)]メニュー > [新規(N)] > [COBOLJVM プロジェクト] を選択します。

ファイ	ル(F) 編集(E) ソース リファクタリング 新規(N)	ナビゲート(N) 検索 Alt+シフト+N >	분 (같	COBOL コピーファイル プロジェクト リモート COBOL プロジェクト
2	ファイルを開く(.) ファイル・システムからプロジェクトを開く 最近のファイル	×	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	リモート COBOL コピーファイル プロジェクト COBOL ユニット テスト プロジェクト COBOL JVM プロジェクト
	閉じる(C) すべて閉じる(L)	Ctrl+W Ctrl+シフト+W	191 191 191	リモート COBOL ユニット テスト プロジェクト COBOL JVM ユニット テスト プロジェクト リモート COBOL JVM プロジェクト

② プロジェクト名に "JVMC" を入力し、[終了(F)] ボタンをクリックします。

プロジェクト名(P)		
デフォルト・ロケーションの使用(D)		
ロケーション(L): C¥Users¥Administrator¥workspace¥JV	MC	參照(R)
プロジェクトテンプレートを選択		
		テンプレートの設定を構成
□ テンフレートの参照		
□ テンフレートの参照 場所:		参照
 サンフレートの参照 場所: ファイルシステムを選択: default ~ 		参照
□ テンフレートの参照 場所 □ ファイルシステムを選択: default 〜 JRE		参菜
□ デンフレートの参照 場所: □ フテイルシステムを選択: default 〜 JRE ● 実行環境 JRE の使用(V):	JavaSE-11	#H
□ デンフレートの参照 場所: □ フナイルシステムを選択: default → JRE ④ 実行環境 JRE の使用(V): ○ プロジェクト国有の JRE を使用(S):	JavaSE-11 AdoptOpenJDK	李詡
□ デソフレートの参照 場所: □ フテイルシステムを選択: default → JRE ④ 実行環境 RE の使用(V): ○ プロジェクト国有の JRE を使用(S): ○ Use default JRE 'AdoptOpenJDK' and workspace	JavaSE-11 AdoptOpenJDK compiler preferences	参照… ママン <u>PEを獲成。</u>
□ デソフレートの参照 場所: □ アナイルシステムを選択: default → JRE ④ 実行環境 IRE の使用(V): ○ プロジェクト国有の JRE を使用(S): ○ Use default JRE 'AdoptOpenJDK' and workspace ワーキング・セット	JavaSE-11 AdoptOpenJDK compiler preferences	参照… > > <u>PEE構成</u> 。
□ デンフレートの参照 場所 □ アナイルシステムを選択: default ■ 実行環境 IRE の使用(V): ○ プロジェクト国有の IRE を使用(S): ○ Use default JRE 'AdoptOpenJDK' and workspace ワーキング・セット □ ローキング・セット	JavaSE-11 AdoptOpenJDK compiler preferences	学語…

- 2) COBOL JVM パッケージを作成、設定変更します。
 - 「JVMC」プロジェクトを右クリックし、コンテキストメニューから、[新規作成(N)] > [COBOL JVM パッケージ] を選択します。



	新規作成(N)	>	COBOL JVM プロジェクト
	表示方法(W) 型階層を開く	Alt+シフト+W > Alt+シフト+H	 COBOL JVM ユニットテスト プロジェクト COBOL コピーファイル プロジェクト
	コピー(C) 修飾名のコピー(Y) 貼り付け(P) 削除(D) コンテキストから除去 移動(V)	Ctrl+C Ctrl+V 削除 Ctrl+Alt+シフト+下	COBOL フロジェクト GOBOL コニット テスト プロジェクト リモート COBOL JM プロジェクト リモート COBOL Jビーファイル プロジェクト リモート COBOL プロジェクト リモート COBOL プロジェクト ・ リモート COBOL ユニット テスト プロジェクト
	名前を変更(M) タスクのスキャン	F2	 プロジェクト(R) [™] COBOL コピーファイル
2	インポート(i) エクスポート(O)	>	COBOL プログラム COBOL プログラム スタンドアロン ファイル リモート スタンドアロン ファイル
8	更新(F) ブロジェクトを閉じる(S) 無関係なプロジェクトを閉じる(U)	FS	COBOL JVM Delegate COBOL JVM Enum COBOL JVM Enum COBOL JVM Value Type
0 0	リモートシステムビューで表示 Coverage As 実行(R)	>	COBOL JVM インターフェイス COBOL JVM インターフェイス COBOL JVM クラス AG COBOL JVM ソースフォルダ
*	デバッグ(D)	2	🖶 COBOL JVM パッケージ

② COBOL JVM パッケージ作成ダイアログが表示されるので名前に "my.pack" とタイプして [終了(F)] ボタンをクリックします。

COBOL JVM パッケージ COBOL JVM パッケージを新規作成します。	Ť
パッケージに対応するフォルダを作成します。 ソース フォルダ(D): ///MC/src 名前(M): my.pack	参照(o)
? 終了(F)	キャンセル

③ 「src」の配下に「my.pack」パッケージが作成されたことが確認できます。



- 3) COBOL プログラムを作成します。
 - 作成された「my.pack」パッケージを右クリックし、コンテキストメニューから [新規作成(N)] > [COBOL プログラム] を 選択します。



	新規作成(N)	> 10	COBOL JVM プロジェクト
	表示方法(W) 型階層を開く	Alt+シフト+W > ピ Alt+シフト+H M	COBOL JVM ユニットテスト プロジェクト COBOL コピーファイル プロジェクト
	コピー(C) 修飾名のコピー(Y) 貼り付け(P) 削除(D) コンテキストから除去	Ctrl+C Ctrl+V 削除 Ctrl+Alt+シフト+下	 COBOL プロジェクト COBOL ユニット テスト プロジェクト リモート COBOL JWM プロジェクト リモート COBOL コピーファイル プロジェクト リモート COBOL ブロジェクト リモート COBOL プロジェクト リモート COBOL プロジェクト
	タスクのスキャン		プロジェクト(R)
P N	インポート(i) エクスポート(O)) 🖬	ĵ COBOL プログラム

② 「COBOL JVM プログラムの新規作成」ウィザードが表示されます。 [名前] に "Calculator.cbl" を入力し、[終

COBOL JVM こ COBOL JVM プロ	Ĵログラムの新規作成のウィザード グラムを作成します	D
ソース・フォルダ(D):	JVMC/src	参照(o)
パッケージ(K):	my.pack	参照(W)
名前(M):	Calculator.cbl	
?	終了(F)	キャンセル

- ③ テンプレートの「Calculator.cbl」が展開されます。
- ④ ダウンロードしたサンプルプログラムから「Caluculator.cbl]をメモ帳等でオープンし、内容を全てコピー&ペーストします。

COBOL コードの説明)					
\$set ilnamespace "my.pack" ←Java のパッケージ化と同等の機能					
\$set ilsmartlinkage "my.pack" ←linkage section に記述されている変数に対応する Java クラスを生成					
\$set ilcutprefix "Ink-" ←_	上記 ilsmartlinkage で生成されるクラス・変数名は、デフォルトでは変数名となる				
が、本指令により名称から	「Ink-」を削除				
linkage section.					
01 args					
03 lnk-arg1	pic 9(5) comp-3.				
03 lnk-arg2	pic 9(5) comp-3.				
03 lnk-sum	pic 9(5) comp-3.				
ここでは Args というクラス	スファイルが自動的に生成される。「Lnk-」をカットする指令が指定されているので				
Java からは「Ink-」を除いた変数名でアクセスができる。					
procedure division using args.					
add Ink-arg1 to Ink-arg2 giving Ink-sum.					
Linkage section の変数を引き継いで加算処理が行われる。					



⑤ CTRL+Sを押してファイルを保存するとコンパイルが行われます。

3.3. Java プロジェクトの作成

- 1) 呼び出し元の Java プロジェクトを作成します。
 - COBOL エクスプローラーにて、Eclipse メニューより、[ファイル(F)] > [新規(N)] > [その他] を選択します。

ァイル(F)	編集(E)	ソース	リファクタリング	ナビゲート(N)	検索	1	サンプル(X)	
新規(№	۱)			Alt+シフト	+N >	-9	その他(o)	Ctrl+N
ファイル	を開く(.)							

② Java プロジェクトを選択し、[次へ(N)] ボタンをクリックします。

1월 Java プロジェクト 19 インターフェース	^						
G 252							
1歳 プラグイン・プロジェクト							
※ 既存 Ant ビルド・ファイルからの Java プロジェクト							
> 🗁 一般							
> 🗁 AspectJ							
> 🗁 Eclipse モデリング・フレームワーク							
> 🧀 EJB							
> 🗁 Git							
> 🗁 Gradle							
> 🗁 Java	~						
(ア) (ア) 終了(F)	キャンセル						

- ③ 「Java プロジェクトの作成」ウィザードが表示されるので[プロジェクト名]に"CALC"とタイプして[JRE] はデフォルトを選 択し、[終了(F)]をクリックします。
- ④ 選択したデフォルトの JRE が Java 11 (またはそれ以降)の場合、module-info.java ファイルをプロジェクトに追加 するように求められます。これはこのチュートリアルの範囲外のため、[作成しない]をクリックします。
- 2) プロパティ情報を更新します。
 - ① 「CALC」プロジェクト上で右ボタンのコンテキストメニューから[プロパティ]を選択します。
 - ② [Java のビルドパス]を選択して、[プロジェクト(P)]タブをクリックします。
 - ③ [クラスパス]を選択し、[追加(D)] ボタンをクリックします。
 - ④ JVMC プロジェクトにチェックを入れて、[OK]ボタンをクリックします。

追加するプロジェクトを選択	沢してください:	
🗹 🖉 лумс		
	すべて選択(S)	選択をすべて解除(D)
	OK	キャンセル

⑤ 次にライブラリー(L)タブをクリックします。



フィルタ入力	Java のビルド・パス	← → ⇒ *
> リソース Coverage Javadoc ロケーション > Java エディク > Java コード・スタイル > Java コンパイラー Iava のドリド・パマ	● ソース(5) ② プロジェクト(7) (1) (2) オロジェクト(7) (2) オリジェクト(7) (2) オリジェクト(7) (2) キジュール依存製(4)(M) ビルド・パス上に必要なプロジェクト(7) ◆ サジュール(7) ◆ ウスコ、(7) ◆ クスコ、(7) ◆ フジュール(7)	這加(D) 編集(E)
Project Facets Server Task Tags > Validation WikiText レルチ プロジェンマト・ネーチャー プロジェント・参照 裏行/デバッグ設定		除去(M)
		適用(L)
?	遠用して	閉じる キャンセル

- ⑥ [クラスパス]を選択し、[ライブラリーを追加(i)] ボタンをクリックします。
- ⑦ 「COBOL JVM 実行時システム」を選択し、[次へ(N)] > [終了(F)] ボタンをクリックします。
 ライブラリーの追加

追加するライブラリー・タ	イプを選択します。			
COBOL JVM 実行時少 CAR Rumanie EAR Libraries JRE システム・ライブラリー JUnit Maven Managed Dep Server Runtime Web App Libraries プラグインの依存関係 ユーザー・ライブラリー 接続可能性ドライバー定	ステム endencies 義			
?	< 戻る(B)	次へ(N) >	終了(F)	キャンセル

⑧ [適用して閉じる] ボタンをクリックします。



- 3) Main メソッドを含んだ Java アプリケーションを作成します。
 - ① 「CALC」プロジェクト上で右ボタンのコンテキストメニューから [新規(W)] > [クラス] を選択します。



	新規(W)	>		Java プロジェクト	
	次ヘジャンプ(I)			AspectJ Project	6
	新規ウィンドウで開く(N)			プロジェクト(R)	5.2
	型階層を開く(N)	F4	تنا	パッケージ	-
	表示方法(W)	Alt+シフト+W >	C	クラス	
P	74-(0	Ctrl+C	G	インターフェース	Ī

② 新規 Java クラス作成のためのウィザード画面が表示されるので [パッケージ(K)] に "com.calc" を入力、[名前

(M)]に "MainClass" を入力し、[終了(F)] ボタンをクリックします。

Java クラス 新規 Java クラスを作成し	έġ.	C
ソース・フォルダ(D):	CALC/src	参照(o)
パッケージ(K):	com.calc	参照(W)
□ エンクロージング型(Y): 		参照(W)
名前(M):	MainClass	
修飾子:	● public(P) ○ package(C) ○ private(V) ○ protected(T) □ abstract(T) □ final(L) □ static(C)	
スーパークラス(S):	java.lang.Object	参照(E)
インターフェース(i):		追加(A)
		除去(R)
作成するメソッド・スタブの	闘 択	
	 □ public static void main(String[] args)(V) □ スーパークラスからのコンストラクター(C) ☑ 継承された抽象メソッド(H) 	
コメントを追加しますか? (テ	ンプレートの構成およびデフォルト値については <u>ここ</u> を参照) □ コメントの生成(G)	
?	終了(F)	キャンセル

- ③ 雛型の「MainClass.java」が展開されます。ダウンロードしたファイルから MainClass.java をメモ帳等でオープンし、このソースコードにコピー & ペーストを行います。
- ④ CTRL+Sを押してファイルを保存するとコンパイルが行われます。

Java コードの説明)
Calculator calc = new Calculator(); ← Calculator クラス本体のインスタンス作成
Args arguments = new Args(); ←Linkage section に定義した変数ヘアクセスするためのクラス
arguments.setArg1(4); ←Linkage section に定義した変数ヘアクセスする setter メソッド
arguments.setArg2(2); ← 同上
calc.Calculator(arguments); ←計算を行うメソッドの呼び出し
System.out.println(arguments.getSum()); ←Linkage section に定義した変数へアクセスする getter メ
ソッドにて値を取得し、コンソールに表示

3.4. 作成した Java アプリケーションの実行

- 1) Java アプリケーションを実行します。
 - ① 「CALC」プロジェクトを右クリックし、コンテクストメニューから [実行(R)] > [Java アプリケーション]を選択します。



Q.	Coverage As	>			
0	実行(R)	>	N	1 Run on Server	Alt+シフト+X,R
*	デパッグ(D)	>		2 Java アプリケーション	Alt+シフト+X,J
	プロファイル(P)	>		実行 の構成(N)	
	□-カル履歴かに復元(Y)		1		

② コンソールに6が返ってくることが確認できます。

			23
<u><終</u> 了> N	lainClass [Jav	 」 ション] C:¥Prog	ram Files (
6			

3.5. COBOL パースペクティブのカスタマイズ

- 1) COBOL パースペクティブをカスタマイズします。
 - ① COBOL エクスプローラーの設定メニューをクリックします。

2	<mark>た</mark> って ポッフ	88 18-N 12 A 4 4 7 	»1 = ▶ ↓ 2 %	" ロ ゆ <mark>8</mark> マイズ	(F)…]を遅	≹択します。
		COBOL JVM プロジェクト表示(I トップレベル要素(T)	R)	> >		
	ß	ワーキング・セットの選択(W) ワーキング・セットの選択解除(K) アクティブなワーキング・セットの編 1ウィンドウ・ワーキング・セット	i集(E)			
	7	フィルタとカスタマイズ(F)				
	\$ <u>1</u> }	エディタにリンク(L)				

③ 非 Micro Focus プロジェクトのチェックを外して[OK]ボタンをクリックします。

□ 閉じたプロジェクト		
🗌 非 Micro Focus プロジェクト		
」 非 public Xン八		~
	ОК	キャンセル

3.6. JVM COBOL のパッケージ化

- 1) COBOL JVM プロジェクト「JVMC」から JAR ファイルを出力するように設定します。
 - ① 「JVMC」プロジェクト上で右クリックし、> [プロパティ]を選択します。
 - ② ビルド構成を選択します。
 - ③ 右側のパンにて[JVM]のセクションにて [パッケージターゲットを作成する] を「はい」に変更します。
 - ④ 次に、[ビルド後に JAR ファイルを作成する] も「はい」に変更します。
 - ⑤ 次に [JAR 出力フォルダ] をデフォルトの「dist」になっていることを確認します。
 - ⑥ 最後に [適用して閉じる] ボタンをクリックします。



	ビルド後に JAR ファイルを作成する ビルド後に JAR ファイルを作成します		\sim
	▶ 追加指令		~
	最大エラー数	100	_
	警告レベル	重大なエラーだけ(レベル S)	
	✓ Iラ-/警告		
	出力ディレクトリ名	bin	
実行/デバッグ設定	JAR ファイル名	JVMC.jar	
プロジェクト参照	JAR 出力フォルダ	dist	
プロジェクト・ネーチャー	ビルド後に JAR ファイルを作成する	はい	~
ビルダー	パッケージ ターゲットを作成する	はい	
タスク・リポジトリー	インクリメンタルビルドの使用	いいえ	_
コンテナー	パッケージ名にディレクトリ階層を反映する	はい	
WikiText	動的呼び出しを使用	いいえ	
Validation	VMU V		
Task Tags	リストファイルを生成	いいえ	
Project Facets	指令ファイルを生成する	いいえ	
指令セット参照	✓ 出力		
ビルド優先順位	詳細	いいえ	
ビルド構成	EXIT PROGRAM を GOBACK として処理	いいえ	

3.7. プロジェクトのビルド

- 1) JAR ファイルをビルドします。
 - ① 通常、自動でビルドが行われますが、もしビルドされない場合は下記の作業を行います。
 - ② COBOL エクスプローラーにて、「JVMC」プロジェクトを選択します。
 - ③ [プロジェクト] メニューから Eclipse メニューより、[プロジェクトのビルド(B)]を選択します。
 - ④ ビルドが終了すると「dist」フォルダに JVMC.jar ファイルが作成されていることが確認できます。



3.8. パッケージ化されたファイルのコピー

- 1) 作成した jar ファイルを Java のプロジェクトに設定します。
 - ① 「CALC」プロジェクトを右クリックし、コンテクストメニューから [新規作成]> [フォルダー] を選択します。
 - ② フォルダ名に"lib"とタイプします。
 - ③「JVMC」プロジェクトにある「JVMC.jar」右クリックで選択し、コンテキストメニューから[コピー]を選択し、その後、 「CALC」プロジェクト配下の「lib」フォルダー上にて右クリックし、コンテキストメニューから「ペースト」を選択します。ファイ ルが「lib」フォルダにコピーされていることを確認します。



3.9. パッケージ化された Jar ファイルを使用するように Java プロジェクトを変更

- 1) Java プロジェクトの変更を行います。
 - ① COBOL エクスプローラーにて「CALC」プロジェクトを右クリックし、コンテクストメニューから [プロパティ]を選択します。
 - ② Java ビルドパスをクリックします。



- ③ [ブロジェクト] タブより「JVMC」を選択し、[除去(M)]をクリックします。
- ④ 次に[ライブラリー] タブを選択します。
- ⑤ クラスパスを選択し、[JAR の追加(J)…]ボタンをクリックします。
- ⑥ 「CALC」プロジェクトを展開し、「lib」フォルダーから「JVMC.jar」を選択し、[OK]ボタンをクリックします。

ビルド・パスに追加するアーカイブを選択してください(C):							
7ィルタ入力							
V 🔁 CALC							
> 🔁 .settings							
bin Carter and Carter							
JVMC.jar							

- ⑦ クラスパスに「COBOL JVM 実行時システム」が消えている場合、追加をしてください。
- ⑧ 最後に [適用して閉じる] ボタンをクリックします。

3.10. Java プロジェクトの実行

- 1) Java アプリケーションを実行します。
 - ① 「CALC」プロジェクトを右クリックし、コンテクストメニューから [実行(R)] > [Java アプリケーション] を選択します。

Q	Coverage As	>			
0	実行(R)	>	Å	1 Run on Server	Alt+シフト+X,R
\$	デバッグ(D)	>	J	2 Java アプリケーション	Alt+シフト+X,J
	プロファイル(P)	>		実行の構成(N)	
	ローカル履歴から復元(Y)				

- ② Java アプリケーションの選択ダイアログが表示されます。
- ③ マッチングアイテムリストから「MainClass com.calc」をクリックします。

型の選択 (? = 任意の文字、* = 任意のストリング、TZ = タイム・ゾーン)(T):		8
**		×
致する項目(M):	+	ヤツシュ更新 (100%)
G Calculator - my pack G MainClass - com.calc		
🖶 com.calc		
۹	OK	al constantin

④ OKをクリックすると CALC アプリケーションが実行されコンソールに6が表示されます。

	問題	@ Javadoc	😟 宣言	📃 コンソール	×
<終	了> M	lainClass [Jav	a アプリケー:	ション] C:¥Prog	ram Files (x
6					



WHAT'S NEXT

• 本チュートリアルで学習した技術の詳細については製品マニュアルをご参照ください。

免責事項

ここで紹介したソースコードは、機能説明のためのサンプルであり、製品の一部ではございません。ソースコードが実際に動作するか、御社業務に適合するかなどに関しまして、一切の保証はございません。 ソースコード、説明、その他すべてについて、無謬性は保障されません。 ここで紹介するソースコードの一部、もしくは全部について、弊社に断りなく、御社の内部に組み込み、そのままご利用頂いても構いません。 本ソースコードの一部もしくは全部を二次的著作物に対して引用する場合、著作権法の精神に基づき、適切な扱いを行ってください。