

# Visual COBOL チュートリアル

1

# 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

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 クラスヘ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

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。

- 開発環境
   Windows 10
   Visual COBOL 10.0 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、任意のフォルダー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



# 内容

- 1 目的
- 2 前提
- 3 チュートリアル手順
  - 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

**Rocket** software

# 3 チュートリアル手順

## 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)]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。
  - [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	イル(F)	編集(E)	リファクタリング	ナビゲート(N)	検索	プロジ	エクト	·(P) 実行(R)	ウィンドウ(W)	ヘルプ(H)	
	新規(	N)		Al	t+シフト	+N >	2	COBOL プロジ	ェクト		
	ファイノ	レを開く(.)					赵	COBOL JL-7	ファイル プロジェ	クト	
۵,	🤰 ファイル・システムからプロジェクトを開く					۲ġ	リモート COBO	L プロジェクト			
	Recent Files					>	Ê	リモート COBOL コピーファイル プロジェクト			
	88187	(0)			<b>C</b> 1		e	COBOL/Java	相互運用機能	のプロジェクト	
	閉しる	(C)			Ctri	+vv	<b>C</b> Ê	COBOL 7 - W	トテストプロジェ	·7ト	
	すべて	閉じる(L)		Ctrl	+シフト	+W	2	COBOL JVM 7	プロジェクト		
	保存(	S)			Ctr	I+S	<b>B</b> Y	リモート COBO	L ユニット テスト	・プロジェクト	

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

<b>COBOL JVM プロシェクト</b> ワークスペースまたは外部の場所に COBOL	JVM プロジェクトを作成しま?	ŧ.			E
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:¥workspace_jvmcob	oI¥JVMC				参照(R)
プロジェクト テンプレートを選択					
				<u> <del>-</del></u>	ノートの設定を構成
□ テンプレートの参照					
場所:					参照
ファイルシステムを選択: default	$\sim$				
JRE					
<ul> <li>実行環境 JRE の使用(V):</li> </ul>	JavaSE-17				~
○ プロジェクト固有の JRE を使用(S):	AdoptOpenJDK				$\sim$
O Use default JRE 'AdoptOpenJDk	and workspace compile	r preferences			<u>JRE を構成</u>
ワーキング・セット					
ワーキング・セットにプロジェクトを追加	(T)				新規(W)
ワーキング・セット(O):				~	選択(E)
٢		= 7 (0)		457(5)	he state
$\bigcirc$		< 戻る(B)	次へ(N) >	終了(F)	キャンセル

- 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、コンテクストメニューから、[新規作成(N)] > [COBOL JVM パッケージ] を選択 します。



$  \mathbf{P} \mathbf{c} \mathbf{v}   \mathbf{R}  $	÷ п.	. » – –			
		新規作成(N)	>	않	COBOL JVM プロジェクト
> ピ JVMC		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 COBOL 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	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	COBOL ユニット テスト プロジェクト COBOL/Java 相互運用機能のプロジェクト リモート COBOL JVM 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 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)	F2		プロジェクト(R)
		タスクのスキャン		Bî	COBOL コピーファイル
ד × E	4	インポート(i) エクスポート(O)	>		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
アウトラインを打 はありません。	\$) 	更新(F) プ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)	F5		COBOL JVM Delegate COBOL JVM Enum
		Source	>	G	COBOL JVM インターフェイス
	0	リモートシステムビューで表示 Coverage As	>	6) R	COBOL JVM クラス COBOL JVM ソースフォルダ
	0	宝行(R)	>	₿	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" を入力して [終了(F)] をクリックし

ます。

COBOL JVM / COBOL JVM /ಉ			
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):	JVMC/src		参照(o)
名前(M):	my.pack		
?		終了(F)	キャンセル

「src」の配下に「my.pack」パッケージが作成されます。



- 3) COBOL プログラムを作成します。
  - ① 作成された「my.pack」パッケージを右クリックし、コンテクストメニューから [新規作成(N)] > [COBOL プログラム]

4



#### を選択します。

<sup>2</sup> ₀C× <sup>1</sup> ₀プ <sup>1</sup> ₀A ✓ ▼ <sup>1</sup> ∞ <sup>1</sup> WMC	»2				
> A COBOL JVM		新規作成(N)	>	않	COBOL JVM プロジェクト
> 🛋 JRE システム・ マ 👛 src		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 COBOL プロジェクト
田 my.pack ■ 参照ライブラリ		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 リファクタリング(T)	Ctrl+C Ctrl+V 削除 Ctrl+Alt+シフト+下 >	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	COBOL ユニット テスト プロジェクト COBOL/Java 相互運用機能のプロジェクト リモート COBOL JVM 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 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)
₽ ア × ■プロ		インポート(i) エクスポート(O)…	>	D) D	COBOL プログラム COBOL プログラム

 「COBOL JVM プログラムの新規作成のウィザード」が表示されます。[名前] に "Calculator.cbl" を入力し、 [終了(F)] をクリックします。

#### COBOL JVM プログラムの新規作成のウィザード

COBOL JVM プログラムを作成します

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	0
COI	BOL 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 "lnk-" ←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	Ê
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 というクラスファイ	ルが自動的に生成される。「Lnk- lをカットする指令が指定されているため、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 lnk-arg1 to lnk-arg2 giving lnk-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

2

- 1) 呼び出し元の Java プロジェクトを作成します。
  - ① 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。

<b>2</b>	サンプル(X)	
<u>-</u>	その他(o)	Ctrl+N
[Jav	ra] > [Java プロジェクト] を選択し、[次へ(N)]	をクリックします。
ウイ	ザードを選択	
Jav	ra プロジェクトの作成	

ウィザード(W):				
フィルタ入力				
✓ ➢ Java	クト			^
<ul> <li></li></ul>	グ・セット ス ダ			- 1
ポパッケージ	N			~
?	< 戻る(B)	次へ(N) >	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、以下の設定を行ったうえで、[終了(F)]をクリックします。 プロジェクト名: "CALC"

Module-info.java を作成: チェックを外す



#### Java プロジェクトの作成

Java フロジェクトをワークスペースまたは外部	ロケーションに作成します。				
プロジェクト名(P): CALC	7				^ ^
✓ デフォルト・ロケーションの使用(D)					
ロケーション(L): C:¥workspace_jvmcob	oI¥CALC				参照(R)
JRE					
<ul> <li>実行環境 JRE の使用(V):</li> </ul>	JavaSE-17				~
〇 プロジェクト固有の JRE を使用(S):	AdoptOpenJDK				$\sim$
O Use default JRE 'AdoptOpenJDK	(' and workspace compil	er preferences			<u>JRE を構成</u>
プロジェクト・レイアウト					
○ プロジェクト・フォルダをソースおよびク	ラス・ファイルのルートとして使	:用(U)			
◉ ソースおよびクラス・ファイルのフォルダ	−を個別に作成(C)			1	既定値を構成
ワーキング・セット					
□ ワーキング・セットにプロジェクトを追加	(T)				新規(W)
ワーキング・セット(0):				$\sim$	選択(E)
モジュール					
module-info.java を作成(M)					
<ul> <li>コメントの生成(G)</li> </ul>					•
?		< 戻る(B)	次へ(N) >	終了(F)	キャンセル

パースペクティブの切り替えのダイアログが表示された場合は、[パースペクティブを開く(O)]をクリックします。

Java パースペクティブを開きますか?		
このパースペクティブは、Java 開発をサポートするため 階層、および Java 固有のナビゲーション・アクションを	に設計されています。パッケ・ を提供します。	-ジ・エクスプローラー、型
□ 常にこの設定を使用する(R)		
	パースペクティブを開く(O)	เงเงิส(N)

- 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クリックして、コンテクストメニューから[プロパティ(R)]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)]をクリックします。

Java のビルド・パス	← ➡ ⇒ <
● ソース(S)      「 ブロジェクト(P)      ■ ライブラリー(L)      ◇ 順序およびエクスポート(Q)      ④ モジュール依存関係(M)     □	
ヒルド・バス上に必要なフロジェクト( <u>R</u> ):	
30 ± 2 1 = 1/1 Å № 572 / 1 = 1/1 Å	追加( <u>D</u> )
	編集( <u>E</u> )
	除去( <u>M</u> )

④ JVMC プロジェクトにチェックを入れて、[OK] をクリックします。



	追加するプロジェクトを選択してください:
	ע ביין אאע אדע אדע אדע אדע אדע אדע אדע אדע אדע
	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(1)タブをクリックします。
J	
	Java のビルド・パス
	ピックース(S)      ビュン・ション・ション・ション・ション・ション・ション・ション・ション・ション・ショ
	ビルド・パフトに心面かずロジェクト/DV



- ⑥ [クラスパス]を選択し、[ライブラリーを追加(i)]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)] をクリックし、次の画面でそのまま [終了(F)] をクリックします。

ラ <b>イブラリーの追加</b> 追加するライブラリー・タ	イプを選択します。			ā
COBOL JVM 実行時ジ CXF ランタイム EAR ライブラリー JRE システム・ライブラリー JUnit Maven Managed Dep Web App ライブラリー サーパー・ランタイム ブラグインの依存関係 ユーザー・ライブラリー 接続可能性ドライパー定	ステム endencies 義			
?	< 戻る( <u>B</u> )	次へ( <u>N</u> ) >	終了(E)	キャンセル
[適用して閉じる]	をクリックします。			



🎙 ソース(S) 🔁 プロジェクト(P) 🛋 ライブラリー(L) 🍫 順序およびエクスポート(O) 🧕 モジュール依	₹存関係( <u>M</u> )
ブルド・パス上の JAR およびクラス・フォルダー(I):	
< % モジュールパス	JAR の追加(J)
> 🛋 JRE システム・ライブラリー [JavaSE-17] 🗙 🌭 クラスパス	外部 JAR の追加( <u>X</u> )
> A COBOL JVM 実行時システム	変数の追加(⊻)
	ライブラリーを追加( <u>i</u> )
	クラス・フォルダの追加( <u>C</u> )
	外部クラス・フォルダーを追加(D
	編集( <u>E</u> )
	除去( <u>R</u> )
	JAR ファイルのマイグレーション(M
	済田小

- 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」プロジェクトを選択し、マウスの右クリックでコンテクストメニューを開き、[新規(W)] > [クラス] を選択します。



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、以下の入力を行い、[終了(F)]をクリックします。

パッケージ: "com.calc"

名前: "MainClass"

- 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 に定義した変数へアクセスするためのクラス

arguments.setArg1(4); ←Linkage section に定義した変数へアクセスする 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	>			
$\mathbf{O}$	実行(R)	>	<b>1</b>	1 サーバーで実行	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)	

② コンソールに計算結果の 6が出力されます。

🖹 問題 @ Javadoc 🗟 宣言	🗐 א-עעב 🖳	
<終了> MainClass [Java アプリ	ケーション] C:¥Prog	gram F
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 画面右上のアイコンをクリックして、パースペクティブを COBOL に変更します。

	Q : 🖻	<u>ه</u> ا	
■ タスク一覧 ×		- 0	
💣 🛨	🔁 🕼   🐲   🗙 👫 🖂	3	
検索	▶ すべて ▶ アクティブ	こす (	1

② COBOL エクスプローラーの設定アイコンをクリックし、 [フィルタとカスタマイズ(F)…] を選択します。



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



🍸 プリセット・フィルター 🍸 ユーザー・フィルター 🍃 コ	ンテンツ	
適用するフィルターを選択してください(一致する項目	は隠されます):	
✓ RSE 内部プロジェクト		^
☑ カテゴリ外の空のフォルダ		
☑ 内部 Micro Focus プロジェクト		
☑ 内部 TD プロジェクト		
□ 一 合成メンバ		
☑ 空のカテゴリ		
🗌 🗌 空のパッケージ		
☑ 空の親パッケージ		
✓ 継承 COBOL プログラム		
□ 閉じたプロジェクト		
□ 非 Micro Focus プロジェクト		
□ 非 public メンバ		~
		da ci a ci a ll
	UK UK	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 [Micro Focus] > [ビルド構成] を選択し、JVM セクションは以下の項目に以下の設定を行ったうえで、[適用して 閉じる] をクリックします。
     パッケージターゲットを作成する:「はい」を選択
     ビルド後に JAR ファイルを作成する:「はい」を選択
     JAR 出力フォルダ: "dist"



フィルタ入力	ビルド構成	(p 🔻 🗘 🥆
> リソース		
Coverage		
✓ Micro Focus	フィルタテキストを入力	
JVM ビルド パス		
> SQL プリプロセッサ	設定	值 /
コピーファイル	✓ 出力	
ビルダー	指令ファイルを生成する	いいえ
ビルド環境	リストファイルを生成	いいえ
ビルド構成	→ JVL	
ビルド優先順位	動的呼び出しを使用	いいえ
指令セット参照	パッケージ名にディレクトリ階層を反映する	(dt)
WikiText	インクリメンタルドルドの使用	
コンテナー	バッケージターケットを作成する	はい
タスク・タグ	ヒルト後に JAR ファイルを作成する	เสบ
> タスク・リボジトリー	JAR 出力フォルタ	dist
ビルダー	JAR Jアイル名	JVMC.jar
フロジェクト・ネーチャー	出力ディレクトリ名	bin
フロジェクト・ファセット	✓ 17-/警告	
フロジェクト参照	警告レベル	重大な⊥フーたけ(レベル S)
> 検証	版大山フー数 14 1-11 14 A	100
実行/デバック設定	✓ 追加指令	· · · · · · · · · · · · · · · · · · ·
	JAR 出力フォルダ 生成される JAR ファイルを配置するフォルダです	0
	COBOL コンパイル設定:	
	CHARSET"ASCII" SOURCE-ENCODING"UTF8 packageToFolderMapping createJar createJa ERROR"100"	" DIALECT"MF" SOURCEFORMAT"variable" NOLIST anim EXITPROGRAM"ANSI" arAfterBuild jarFolder=dist JVMC.jar outputDirectory=bin NOWARNING MAX-
		デフォルトの復元(D) 適用(L)
?		適用して閉じる キャンセル

# 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。

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。

- ① COBOL エクスプローラーにて、「JVMC」プロジェクトを選択します。
- Eclipse メニューより、[プロジェクト(P)] > [プロジェクトのビルド(B)]を選択します。
- ③ ビルドが終了すると「dist」フォルダーに JVMC.jar ファイルが作成されます。

### 🗸 🖾 JVMC

- > 🛋 COBOL JVM 実行時システム
- 🗸 🗁 dist
  - 📄 JVMC.jar
- > 🛋 JRE システム・ライブラリー [JavaSE-17]
- 🗸 🕭 src
  - > 🖶 my.pack
  - 🛋 参照ライブラリー

# 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。



° <mark>≌</mark> COB × [	ک ک	コ   🗟 Appl 🔳 サーバ	📇 Anal 📃 🗖	Cal	culat	or.cbl	MainClass.java ×
> 🔁 CALC			<ul> <li>E</li> <li>E</li></ul>	pa	acka	ge com.	calc;
🗸 🛃 JVMC		新規作成(N)		>	알	COBOL	IVM プロジェクト
> 🛋 COBO		表示方法(W)	Alt+シ	フト+W >		COBOL	IVM ユニット テスト プロジェクト
🗸 🗁 dist		בא-		Ctrl+C	- 📂	COBOL	Jビーノアイル ノロシエクト プロミジェクト
JV 📄	Ē	貼り付け		Ctrl+V	en e	COBOL	フロシェント ユニット テスト プロジェクト
> 🛋 JRE 🖇	×	削除(D)		削除	2	COBOL/	Java 相互運用機能のプロジェクト
v ⊯ m	<u>_0_</u>	コンテキストから除去	Ctrl+Alt+シ	フト+下	<b>e</b>	リモートロ	OBOL JVM プロジェクト
> ■ 111		移動(V)			<b>(</b>	リモートロ	COBOL コピーファイル プロジェクト
		名前を変更(M)		F2		リモートロ	COBOL ブロジェクト
		タスクのスキャン			-	リモートロ	
		インポート(i)		>		ブロジェク	ŀ(R)
	4	エクスポート(O)…				スタンドア	ロンファイル
	8	更新(F)		F5	Ľ	リモートフ	マタンドアロン ファイル
		プロジェクトを閉じる(S)				SQL 774	1JL
腔 アウトライン	イン	無関係なプロジェクトを閉じる(U	(L			ファイル	
		-			-	77709-	

- ② フォルダ名に "lib" を入力し、[終了(F)] をクリックします。
- ③「JVMC」プロジェクトにある「JVMC.jar」を右クリックで選択し、コンテクストメニューから[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、コンテクストメニューから「貼り付け」を選択します。
   ファイルが「lib」フォルダーにコピーされます。

В СОВ ×	🔁 プロ	🗟 Appl	-
V 😂 CALC			
> 🗁 bin			
🗸 🗸 🗠 lib			
<b>.</b>	JVMC.jar		
> 🗁 src			
🗸 📽 JVMC			

### 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 コンテクストメニューにて「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(R)] 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(P)] タブより「JVMC」を選択し、[除去(M)]をクリックします。

Java のビルド・パス	← → ⇒
🥭 ソース(S) 🗁 プロジェクト(P) 🛋 ライブラリー(L) 🍫 順序およびエクスポート(Q) 🥥 モジュール依存関係( <u>M</u> )	
ビルド・パス上に必要なプロジェクト( <u>R</u> ):	
☆ モジュールパス	追加( <u>D</u> )
✓ <sup>4</sup> √ クラスパス	
> CE JVMC	編集( <u>E</u> )
	除去( <u>M</u> )

④ 次に[ライブラリー(L)] タブを選択したうえで、クラスパスを選択後、[JAR の追加(J)…] をクリックします。



Java のビルド・パス	
(B) ソース(S) G プロジェクト(P) A ライブラリー(L) 🍫 順序およびエクスポート(Q) O モジュール依存関係(M)	
ビルド・パス上の JAR およびクラス・フォルダー( <u>T</u> ):	
<ul> <li>◆ そジュールパス</li> <li>&gt; ▲ IRE システム・ライブラリー [JavaSE-17]</li> <li>◆ クラスパス</li> <li>&gt; ▲ COBOL JVM 実行時システム</li> </ul>	JAR の追加( <u>J</u> )
	外部 JAR の追加( <u>X</u> )
	変数の追加(⊻)

⑤ 「CALC」プロジェクトを展開し、「lib」フォルダーから CALC プロジェクト配下の「lib/JVMC.jar」を選択し、[OK]

### 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):	
7ብሥን入力		
✓ 🛱 CALC		
> 🗁 .settings		
> 🗁 bin		
V 🗁 lib		
> 🦻 src		
<ul> <li>.classpath</li> <li>.project</li> </ul>		
InternalTDProject		
> 😂 JVMC		
> 🗁 RemoteSystemsTempFiles		
	ОК	キャンセル

⑥ [適用して閉じる] をクリックします。

Java のビルド・パス	↓ ↓ ↓ 8
😬 ソース(S) 😂 プロジェクト(P) 🛋 ライブラリー(L) 🍫 順序およびエクスポート(Q) 😡 モジュール依存関係( <u>M</u> )	
ビルド・パス上の JAR およびクラス・フォルダー( <u>T</u> ):	
<ul> <li>◆ モジュールパス</li> <li>&gt; ▲ JRE システム・ライブラリ- [JavaSE-17]</li> <li>◆ クラスパス</li> <li>&gt; 區 JVMC.jar - CALC/lib</li> <li>&gt; ▲ COBOL JVM 実行時システム</li> </ul>	JAR の追加( <u>J</u> )
	外部 JAR の追加(X)
	亦 粉 の 같은 뉴미 () 이
	2. 致(0)追/川(⊻)
	ライブラリーを追加(i)…
	クラス・フォルダの追加( <u>C</u> )
	外部クラス・フォルダーを追加( <u>D</u> )
	編集( <u>E</u> )
	除去( <u>R</u> )
	JAR ファイルのマイグレーション( <u>M</u> )
	適用( <u>L</u> )
	適用して閉じる キャンセル



## 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。
  - 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 「MainClass com.calc」を選択して、[OK] 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):			
**		×	
一致する項目(M):			
Calculator - my.pack			
# com.calc			
?	OK	キャンセル	

さきほど同様、コンソールに結果が出力されます。



#### 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。