Visual COBOL チュートリアル

COBOLとJava の相互運用

1 目的

ビジネスの基幹システムで使用されている開発言語として COBOL は長く利用し続けられていますが、Web システムなど 他システムでは Java をはじめ、様々な開発言語で記述されています。このことは、開発言語の優劣を表すものではなく、 それぞれの開発言語の得意分野を有効に選択している表れです。例えば、Web システムをあげてみますと、Java には、 様々なフレームワークが提供されており、すでに実装経験・知識を持つ開発者も多いことから、Java を選択することが最適 となる場合が多いでしょう。これとは反対に、COBOL で記述された基幹システム、多くの計算処理が含まれているようなア プリケーションの機能拡張を考えたとき、Java で新規開発するよりも COBOL で記述するほうが計算精度を保ち、保守 性を維持することができます。

これからのシステムは、以前のような単独で稼働する形態から、様々なシステム・アプリケーションとの連携が求められます。また、基幹システムが稼働を開始した後、別システムの開発・運用を行っていく中で、ある機能が基幹システムでも利用できたら、と感じることもあるかもしれません。基幹システムで利用するために、同じ機能を新たに COBOL で開発となると、保守性の劣化が懸念されますが、新規開発が不要、もしくは、容易に導入できるのであれば、どうでしょうか。 本チュートリアルでは、このようなシステム間連携を見据え、COBOL アプリケーションが Java 資産を利用する方法について、また、Java アプリケーションから COBOL 資産を利用する方法について紹介します。

2 前提

- 本チュートリアルで使用したマシン: Windows Server 2019
- Adoptium OpenJDK17
- Visual COBOL 9.0J for Eclipse 製品をインストールし、COBOL 開発が行える環境

本チュートリアルでは、一部の手順において、下記リンク先のサンプルファイルを使用します。 事前にダウンロードをお願いしま す。

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 COBOL から Java 資産を呼び出す
 - 3.1 SYSTEM ルーチンを利用した Java プログラムを起動
 - 3.2 Java 資産をサービスとして運用し、COBOL から利用
 - 3.2.1 前提条件
 - 3.2.2 サービス定義ファイルからのクライアントプログラムの生成
 - 3.2.3 クライアントプログラムの動作確認
 - 3.3 COBOL/Java 相互運用機能を利用
 - 3.3.1 COBOL/Java 相互運用機能のプロジェクトの利用
 - 3.3.2 COBOL プロジェクトから外部の Java 資産の利用
 - 3.4 COBOL アプリケーションの実行環境を Java 仮想マシンに移して Java 資産とともに Java として利用
- 4 Java から COBOL 資産を呼び出す
 - 4.1 Runtime.exec() を利用して COBOL プログラムを呼び出す
 - 4.2 製品に付属する COBOL 専用のアプリケーションサーバーを利用して、COBOL 資産をサービスとして利用
 - 4.3 COBOL/Java 相互運用機能を利用
 - 4.3.1 COBOL/Java 相互運機能のプロジェクトを利用
 - 4.3.2 COBOL と Java を別プロジェクトで利用
 - 4.4 COBOL アプリケーションの実行環境を Java 仮想マシンに移して Java 資産とともに Java として運用
- 5 Visual COBOL for Eclipse 上の文字コード設定について
 - 5.1 ワークスペースに対する文字コード設定
 - 5.2 プロジェクトに対する設定

3 COBOL から Java 資産を呼び出す

COBOL から Java 資産を呼び出す代表的な方法は以下になります。

• SYSTEM ルーチンを利用した Java プログラムを起動

3.1 SYSTEM ルーチンを利用した Java プログラムを起動

最も簡単な Java プログラムの起動方法は、SYSTEM ルーチンを利用した Java プロセスの起動です。例えば、以下のプログラムでは、 java のバージョン情報を表示します。

working-storage section. 01 cmd pic x(50) value "java -version". procedure division. call "system" using cmd.

補足)

java に限らず、任意のプログラムを実行することができます。

しかし、別プロセスとしての起動となることに加え、起動する Java プログラムへ情報を渡そうとした場合、実行時引数、もしくは、ファイルやデータベースの利用などが必要となることから、単純なプログラム起動以外には適しません。

以降に紹介する方法では、このような問題を解決する方法を紹介します。

- Java 資産をサービスとして運用し、COBOL から利用
- COBOL/Java 相互運用機能を利用
- COBOL アプリケーションの実行環境を Java 仮想マシンに移して Java 資産とともに Java として利用

3.2 SYSTEM ルーチンを利用した Java プログラムを起動

最も簡単な Java プログラムの起動方法は、SYSTEM ルーチンを利用した Java プロセスの起動です。例えば、以下のプログラムでは、java のバージョン情報を表示します。

working-storage section. 01 cmd pic x(50) value "java -version". procedure division. call "system" using cmd.

補足)

java に限らず、任意のプログラムを実行することができます。

しかし、別プロセスとしての起動となることに加え、起動する Java プログラムへ情報を渡そうとした場合、実行時引数、もしくは、ファイルやデータベースの利用などが必要となることから、単純なプログラム起動以外には適しません。

以降に紹介する方法では、このような問題を解決する方法を紹介します。

3.3 Java 資産をサービスとして運用し、COBOL から利用

一般的なシステム間連携機能として Web API、サービスがあげられますが、このような機能を提供することで、 COBOL を含めた様々な開発言語、アプリケーションとの連携が可能になります。本節では、JSON データを 送受信する REST API を利用する方法について紹介します。なお、Java 資産をサービスとして運用する方法については、別途インターネット文献などを参照ください。

本節では、サービスプロバイダが提供するサービスの定義を利用して COBOL クライアントプログラムを生成し、 そのクライアントを用いたサービス連携を行います。

補足)

本チュートリアルでは、Java 資産のサービス運用を前提に紹介していますが、サービスの開発言語は Java に限定されず、C# などの .NET 言語、Node.js なども利用できます。

3.3.1 前提条件

このチュートリアルは、REST API でアクセス可能なサービスの開発、稼働については対象外です。また、 以降で説明する手順では、以下のオンラインで公開されているテストサービスの1つを利用します。 https://jsonplaceholder.typicode.com/

パス: posts/{postId}/comments

HTTP メソッド: GET

例) https://jsonplaceholder.typicode.com/posts/1/comments

多くの開発言語では、公開 API に対するクライアントプログラムは、API が提供するサービスを定義 したファイルから生成できます。これをスキーマ駆動開発と呼びますが、Visual COBOL 製品を利用す ることで COBOL でもスキーマ駆動開発を利用できます。本チュートリアルで使用するサービスに対応 するサービス定義ファイルは、サービス提供サイトからは提供されていないため、上記サービスに対応する 定義をサンプルファイル内に get_post.xml として用意しています。このファイルは、OpenAPI 仕様 に沿った yaml 形式で記述されています。OpenAPI については、以下の公式サイトなどを参照くだ さい。

https://www.openapis.org/

3.3.2 サービス定義ファイルからのクライアントプログラムの生成

- スタートメニューより、[Micro Focus Visual COBOL] > [Visual COBOL コマンドプロンプト (64-bit)]を選択します。
- 2) プロンプト上で、サンプルファイルを解凍したフォルダ配下の apiservice フォルダに移動します。
- プロンプト上で、以下のコマンドを実行して、クライアントプログラムを生成します。
 imtkmake -genclientjson clientjson=get post.yaml

C:¥COBOLJavaInteroperability¥apiservice>imtkmake -genclientjson clientj son=get_post.yaml Micro Focus Interface Mapping Toolkit v9.0.00167 (C) Copyright 1984-2023 Micro Focus or one of its affiliates. C:¥COBOLJavaInteroperability¥apiservice>

生成される COBOL ファイルとコピーブックは以下になります。

ファイル名	説明
get_post-app.cbl	サービスとの接続確認を行う対話形式のコンソールアプリケ
	ーション
get_post-proxy.cbl	get_post-app.cbl から呼び出され、サービスとの通信を
	行うプログラム
get_post-copy.cpy	上記 2 ファイルより参照されるコピーファイル

3.3.3 クライアントプログラムの動作確認

1) 前手順に引き続き、Visual COBOL コマンドプロンプト上で、以下のコマンドを実行し、プログラ ムのコンパイルを行います。

cobol get_post-app.cbl gnt;

cobol get_post-proxy.cbl gnt;

C:\COBOLJavaInteroperability\Papiservice>cobol get_post-app.cbl gnt; Micro Focus COBOL Version 9.0 (C) Copyright 1984-2023 Micro Focus or one of its affiliates. * チェック終了:エラーはありません - コード生成を開始します * Generating get_post-app * Data: 138352 Code: 5716 Literals: 2288 C:\COBOLJavaInteroperability\Lapiservice>cobol get_post-proxy.cbl gnt; Micro Focus COBOL Version 9.0 (C) Copyright 1984-2023 Micro Focus or one of its affiliates. * チェック終了:エラーはありません - コード生成を開始します * Generating get_post-proxy * Data: 1296 Code: 4338 Literals: 3

2) プロンプト上で、以下のコマンドを実行します。

runw get_post-app.gnt

表示された画面上で、以下の入力を行ってください。

Service Address: 何も入力せず Enter キーを押す Supplemental Query String: 何も入力せず Enter キーを押す Username: 何も入力せず Enter キーを押す Password: 何も入力せず Enter キーを押す Operation: "1"を入力して Enter キーを押す Path Parameters: id: "1"を入力して Enter キーを押す

サービスからの応答結果が表示されます。



[OK] をクリックして、アプリケーションを終了します。

3.4 COBOL/Java 相互運用機能を利用

3.3 で紹介した方法は、Java 資産がサービスとして運用され、クライアント環境からアクセスできることが必要 です。しかし、運用環境によっては、サービスやシステムが別環境上で稼働、環境間にファイアウォールが存在す るなど、サービスの通信ポートへのアクセスがブロックされていることがあります。また、新たにサービスを立ち上げた くない、というケースも考えられます。このような課題をクリアしつつ、COBOL から Java 資産を呼び出すことが できる方法が、本節で紹介する COBOL/Java 相互運用機能です。

3.4.1 COBOL/Java 相互運用機能のプロジェクトの利用

本項で使用するプロジェクトは、COBOL 資産と Java 資産を同じプロジェクト配下で管理します。 しかし、Eclipse IDE 上でデバッグ実行ができる対象は COBOL 資産に限定されます。Java 資産 については、別途 Java プロジェクトなどを作成したうえで、デバッグ作業を実施してください。

- Windows スタートメニューより、[Micro Focus Visual COBOL] > [Visual COBOL for Eclipse] を選択して、Visual COBOL for Eclipse を起動します。 ワークスペースは任意のフォルダでかまいません。以降の手順では、c:¥workspaceinteroperability を使用します。
- ワークスペースに対する文字コード設定を行います。
 5.1 の手順を実施してください。
- 3) [ファイル(F)] > [新規(N)] > [COBOL/Java 相互運用機能のプロジェクト] を選択します。

ファイ	ル(F)	編集(E)	リファクタリング	ナビゲート(N)	検索 プロシ	ジェクト(P) 実行(R) ウィンドウ(W) ヘルプ(H)
	新規(N)		1	Alt+シフト+N >	2	COBOL プロジェクト
ファイルを開く(.)		2	COBOL コピーファイル プロジェクト				
😂 ファイル・システムからプロジェクトを開く		贒	リモート COBOL プロジェクト				
最近のファイル >		礅	リモート COBOL コピーファイル プロジェクト				
	閉じる	(C)			Ctrl+W	C Ê	COBOL ユニット テスト プロジェクト
	すべて	閉じる(L)		C	trl+シフト+W	e	COBOL/Java 相互運用機能のプロジェクト
						J*-9	COROL IVM /US2T//N

以下の入力を行い、[次へ(N)]をクリックします。

プロジェクト名: "COBOLJavaProj1"

プロジェクト名(<u>P</u>): COBOLJavaProj1	
プロジェクト テンプレートを選択	
どうして、「シブレート [32 ビット]	
USE Micro Focus テンプレート [64 ビット]	
<u>-</u> <u></u>	<u>ートの設定を構成</u>
□ 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 ~	
☑ デフォルト・ロケーションの使用(型)	
ロケーション(<u>L</u>): C¥workspace-interoperability¥COBOLJavaProj1	参照(<u>R</u>)
ファイル・システムを選択(Y): デフォルト 🗸	
(?) < 戻う(B) 次へ(N) > 終了(F)	キャンヤル

4) JRE に [実行環境 JRE の使用] を選択し、[終了(F)] をクリックします。

JRE ・ 家行環境 JRE の使用(<u>U</u>): ・ プロジェクト固有の JRE を使用(<u>S</u>): ・ Use def <u>a</u> ult JRE 'AdoptOpenJDK' and	行環境 JRE の使用(<u>V</u>): 1ジェクト固有の JRE を使用(<u>S</u>): e def <u>a</u> ult JRE 'AdoptOpenJDK' and workspace compiler preferences		JavaSE-17 ∨ AdoptOpenJDK ∨		
				<u>JRE を構成</u>	
?	<mark>< 戻</mark> る(<u>B</u>) 次	^(<u>N</u>) >	終了(<u>F</u>)	キャンセル	

COBOLJavaProj1 プロジェクトが作成されます。

ファイル(F)	編集(E)	リファクタリ	リング
: 📬 👻 🗐	1	- 0 -	Q
B COBOL	× ٩	5. Naviga	t I
> 🛃 COB	OLJavaPro	oj1	

5) プロジェクトに対する文字コードの設定を行います。

5.2の手順を実施してください。

COBOLJavaProj1 プロジェクトを選択し、マウスの右クリックによりコンテキストメニューを開き、
 [新規作成(N)] > [COBOL プログラム] を選択します。



そのまま、[終了(F)]をクリックします。

含まれるプロジェクト: COBOLJavaProj1	参照
新規ファイル名: Program1.cbl	
テンプレートを選択:	
📄 Micro Focus テンプレート	
	テンプレートの設定を構成
□ テンプレートの参照	
場所: ファイルシステムを選択: default ~	参照
?	終了(<u>F)</u> キャンセル

作成された Program1.cbl を、サンプルファイルを解凍したフォルダ内の COBOLtoJava フォ ルダ配下の Program1.cbl の内容で上書きしてください。

 COBOLJavaProj1 プロジェクトを選択し、[ファイル(F)] > [新規(N)] > [その他(o)] を選 択します。

:\$	CICS Web サービス	
<u> 1</u>	Application Analysis Server への接続	
	サンプル(X)	
⊡	その他(o) Ctrl+N	

8) [Java] > [クラス] を選択して、[次へ(N)] をクリックします。

ウィザード(W):	
フィルタ入力	
V 🔁 Java	^
(学 Java フロジェクト Java ローキング・セット	
<u> </u>	
© 252	
●3 ワース・フォルタ ● パッケージ	
影 既存 Ant ビルド・ファイルからの Java プロジェクト	
 (び)記録 (№)注釈 	
 ● 列挙型 	
> 눧 Java の実行/デバッグ	×
(P) 終了(F) <	キャンセル

以下の入力を行ったうえで、[終了(F)]をクリックします。

パッケージ: "com.sample"

名前: "COBOLJava"

ソース・フォルダ(D):	COBOLJavaProj1/src		参照(o)
パッケージ(K):	com.sample		参照(W)
□ エンクロージング型(Y):			参照(W)
名前(M):	COBOLJava		
修飾子:	public(P) Opackage(C) Oprivate(V) Oprote	ected(T)	1
	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			
	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ここを参照)		
	□ コメントの生成(G)		
$\langle \rangle$	< 戻る(B) 次へ(N) > 終了	(F)	キャンセル

COBOLJavaProj1 プロジェクト配下に src¥com¥sample フォルダが作成され、 COBOLJava.java が作成されます。



作成された COBOLJava.java を、エディター上でサンプルファイルを解凍したフォルダ内の COBOLtoJava フォルダ配下の COBOLJava.java の内容で上書き保存してください。

9) COBOLJavaProj1 プロジェクトを選択し、[実行(R)] > [実行構成(N)] を選択します。

実行	i(R)	ウィンドウ(W)	ヘルプ(H)	
R	実行点をリセット			
Q (10)	実行(R) デパッグ(D)			
0	実行 実行	テ履歴(T) 〒(S)		
	実行	亍構成(N)		

[COBOL/Java 相互運用機能のアプリケーション] を選択したうえで、マウスの右クリックによりコ ンテキストメニューを開き、[新規構成(W)] を選択します。



10) 以下の入力を行い、[実行(R)] をクリックします。

名前: COBOLJavaProj1

名前(N): COBOLJavaProj1							
- 般 シース 歴 環境 共通()	C) 🛋 JRE 🌭 クラスパス		℡∥ デバッグシンボル	人 動的分析	두 6일 CTF 🥑 3	コンテナー	
▼ COBOL プロジェクト(P)		Prizetana	-y 11011100	12 30 100 0	· = • •		
COBOLJavaProj1							
▼ 主プログラム							
☑ プログラムはプロジェクトビルド構成の一	-部: New Configu 〜						
New_Configuration.bin/COBOLJava	Pro 参照						
▼ 開始オプション							
コマンド行引数:							
	^						
	~						
作業ティレクトリ:							
	参照						
▶ 実行オプション							
				前回伊	保管した状態に戻る	‡(V)	適用(Y)
					実行()		問じる
					£1)(N)	19102

COBOL から Java が呼び出され、COBOL からの情報が出力されます。続いて、Java から 戻された値が COBOL から出力されます。

オリジナル配列順序>>>
赤
禄
青
橙
藍
Java でのソート結果>>>
橙
禄
藍
赤
青
Java でセットされた値の表示>>>
橙
黄
青
藍
紫
続行するには何かキーを押してください

何かキーを押して、アプリケーションを終了します。

3.4.2 COBOL プロジェクトから外部の Java 資産の利用

前項では、COBOL 資産と Java 資産を同じプロジェクト内に配置しました。しかし、同時に開発を 行わない限り、一般的には Java 資産は別なフォルダ、すなわち、プロジェクト外に保存されます。 本項では、COBOL, Java の2つのプロジェクトを使用した COBOL/Java 相互運用機能を利用す る方法を紹介します。

注意)

以降の手順で参照する Java クラスは 3.4.1 で作成したものです。こちらの手順の実施前に、 3.4.1 を実施してください。

1) Visual COBOL for Eclipse の起動

Windows スタートメニューより、[Micro Focus Visual COBOL] > [Visual COBOL for Eclipse] を選択して、Visual COBOL for Eclipse を起動します。 ワークスペースは任意のフォルダでかまいません。以降の手順では、c:¥workspace-

interoperability を使用します。

2) [ファイル(F)] > [新規(N)] > [COBOL プロジェクト] を選択します。

ファイル(F)	編集(E)	リファクタリング	ナビゲート(N)	検索	プロジ	፤//	P) 実行(R)	ウィンドウ(W)	ヘルプ(H
新規(N)		A	lt+シフト	+N >	B	COBOLプロジ	ジェクト	
ファイノ	ルを開く(.)					2	COBOL 그ド-	ファイル プロジェ	クト

以下の入力を行い、[終了(F)]をクリックします。

プロジェクト名: "COBOLJavaProj2"

プロジェクトテンプレート: "Micro Focus テンプレート(64 ビット)"

プロジェクト名(P) COBOLJavaProj2	
 プロジェクト テンプレートを選択	
1号 Micro Focus テンプレート [32 ピット] 1号 Micro Focus テンプレート [64 ピット]	
	テンプレートの設定を構成
□ 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 ~	
☑ デフォルト・ロケーションの使用(D)	
ロケーション(L): C:¥workspace-interoperability¥COBOLJavaProj2	参照(R)
ファイル・システムを選択(Y): デフォルト ~	
? 終7	(F) キャンセル
COBOLJavaProj2 プロジェクトが作成されます。	
🔓 COBOL 🗙 🔂 Navigat	
N 12 COBOL JavaProi1	
> 🔁 COBOLJavaProj2	

- プロジェクトに対する文字コード設定を行います。
 5.2 の手順を実施してください。
- COBOLJavaProj2 プロジェクトを選択し、マウスの右クリックによりコンテキストメニューを開き、 [新規作成(N)] > [COBOL プログラム]を選択します。

> E COBOLJavaPro	j1	L				
		新規作成(N)	>	얟	COBOL JVM プロジェクト	
		表示方法(W)	Alt+シフト+W >	<mark>₽</mark> ₽	COBOL JVM ユニット テスト プロジェクト	
		אר-	Ctrl+C	2	COBOL コピーファイル プロジェクト	
	ĥ	貼り付け	Ctrl+V	2	COBOL フロジェクト	
	×	削除(D)	削除	Eŭ Jes	COBOL ユニット テスト フロシェクト COBOL Unixe 相互運用機能のプロジェクト	
		移動(V)			Uモート COBOLIVM プロジェクト	
		名前を変更(M)	F2		リモート COBOL コピーファイル プロジェクト	
		タスクのスキャン		1	リモート COBOL プロジェクト	
		コード分析	>	R	リモート COBOL ユニット テスト プロジェクト	
		インポート(i)	>	1	プロジェクト(R)	
	4	エクスポート(0)		RŶ	COBOL コピーファイル	
	8	更新(F)	F5	đ	COBOL プログラム	
そのまま、[終 ⁻	了(F	=)] をクリックしま	す。			
含まれるプロジェ	:/ነ	COBOLJavaProj2			参照	1
新規ファイル名:		Program1.cbl				
テンプレートを運	択:					
Micro	Foc	us テンプレート				
					<u>テンプレートの設定を構成</u>	
□ テンプレー	トのき	参照				
場所:					参照	
ファイト	697	テムを選択: defaul	t ~			
27.12						
				-		_
?					終了(F) キャンヤル	1
J						1

作成された Program1.cbl を、エディター上でサンプルファイルを解凍したフォルダ内の COBOLtoJava フォルダ配下の Program1.cbl の内容で上書き保存してください。

5) [実行(R)] > [実行構成(N)] を選択します。



[COBOL アプリケーション] を選択したうえで、マウスの右クリックによりコンテキストメニューを開き、 [新規構成(W)] を選択します。



6) 以下の入力を行い、[実行(R)]をクリックします。

```
名前: "COBOLJavaProj2"
```

```
[環境] タブを選択
```

[追加(A)]をクリックし、以下の環境変数を追加します。

• JAVA_HOME

"C:¥Program Files (x86)¥Micro Focus¥Visual COBOL¥AdoptOpenJDK"

• CLASSPATH

"%CLASSPATH%;..¥..¥COBOLJavaProj1¥bin"

前(N) COBOLJavaProj2 → 一般 「シソース 四 環境 □ 共通(C) ▶● 実行時 1 (注: ここで定義された変数は、任意の現在の設定値、または 環境スクリプト内の設定値を上書きします。) 変数 JAVA_HOME CLASSPATH	「シテバッグシンボル 」 動的分析 「松」 CTF ■ コンテナー 任意の指定された 値 C¥Program Files (x86)¥Micro Focus¥Visual COBOL¥Adopt %CLASSPATH%¥.¥COBOLJavaProj1¥bin	追加(A) 編集(E) 創除(R)
実行する環境スクリプト: 場所:		₱₩
ファイルがフロジェクト内にある場合、絶対バスはオ パラメータ:] 関連付けられたプロジェクトのビルド環境から値を継承	前回保管した状態に戻す(り 適用(Y)
	(単行の)	BI'S

注意)

上記の JAVA_HOME 環境変数は、製品のデフォルトインストールした場合のパスです。 インストール先を変更した場合や、3.4.1 の手順で JRE 環境の設定を変更した場合は、設 定した環境を指定してください。

3.4.1 と同じ結果が戻されます。

オリジナル配列順序>>>		
赤		
緑		
青		

橙
藍
Java でのソート結果>>>
橙
緑
藍
赤
青
Java でセットされた値の表示>>>
橙
黄
青
藍
紫
続行するには何かキーを押してください

なにかキーを押して、アプリケーションを終了します。

3.5 COBOL アプリケーションの実行環境を Java 仮想マシンに移して Java 資産とともに Java として利用

この方法は、製品が提供する JVM COBOL 機能を利用します。別途チュートリアルが提供されていますので、 以下のチュートリアルを参照ください。

https://www.amc.rocketsoftware.co.jp/manuals/VC90/Eclipse/index.html?t=GUID-D10DC512-FDEF-44CF-8A9B-32839729B493.html

Visual COBOL 9.0 for Eclipse のチュートリアルトップからは、以下のように進んでください。 [ここからはじめよう] > [Getting Started] > [JVM COBOL チュートリアル]

4 Java から COBOL 資産を呼び出す

Java から COBOL 資産を呼び出す代表的な方法は以下になります。

- Runtime.exec() を利用して COBOL プログラムを呼び出す
- 製品に付属する COBOL 専用のアプリケーションサーバーを利用して、COBOL 資産をサービスとして利用
- COBOL/Java 相互運用機能を利用
- COBOL アプリケーションの実行環境を Java 仮想マシンに移して Java 資産とともに Java として運用

4.1 Runtime.exec() を利用して COBOL プログラムを呼び出す

Java は、別なプロセスを起動するための API として、Runtime クラスの exec メソッドを提供しています。 最も簡単な COBOL プログラムの起動方法は、このメソッドの利用です。例えば、以下のプログラムでは、 MyCOBOLApp.exe モジュールを実行します。



補足)

この方法は、COBOL に限らず、任意のプログラムを実行することができます。なお、上記方法では、実行時に COBOL アプリケーションが実行できる環境設定が行われている必要があります。

しかし、起動する COBOL プログラムへ情報を渡そうとした場合、exec() メソッドの第二引数、上記サンプル では args の使用、もしくは、ファイルやデータベースの利用などが必要となるため、単純なプログラム起動以外 には適しません。

以降に紹介する方法では、このような問題を解決する方法を紹介します。

4.2 製品に付属する COBOL 専用のアプリケーションサーバーを利用して、COBOL 資産をサー ビスとして利用

Visual COBOL 製品には、COBOL 専用のアプリケーションサーバー機能が提供されており、このサーバー上 で COBOL 資産を容易にサービスとして運用することができます。このサービスは、Eclipse IDE 上で、サービ スの新規開発からテスト、デプロイまで作業を完結できます。 こちらの方法は、別途チュートリアルが提供されていますので、以下のチュートリアルを参照ください。 https://www.amc.rocketsoftware.co.jp/manuals/CMN/MFVC_900_ECLWSVC01.pdf Visual COBOL 9.0 for Eclipse のチュートリアルトップからは、以下のように進んでください。 [ここからはじめよう] > [Getting Started] > [ネイティブ COBOL チュートリアル] > [Interface Mapping Toolkit - RESTful Web サービスによる COBOL 資産の再利用]

4.3 COBOL/Java 相互運用機能を利用

COBOL/Java 相互運用機能を利用することで、COBOL 資産のサービス化を行うことなく、Java から COBOL 資産を呼び出すことができます。

4.3.1 COBOL/Java 相互運機能のプロジェクトを利用

- Visual COBOL for Eclipse の起動 Windows スタートメニューより、[Micro Focus Visual COBOL] > [Visual COBOL for Eclipse] を選択して、Visual COBOL for Eclipse を起動します。 ワークスペースは任意のフォルダでかまいません。以降の手順では、c:¥workspacejavacollaborate を使用します。
- 2) ワークスペースに対する文字コードの設定を行います。

5.1の手順を実施してください。

3) [ファイル(F)] > [新規(N)] > [COBOL/Java 相互運用機能のプロジェクト] を選択します。

ファイ	ル(F)	編集(E)	リファクタリング	ナビゲート(N)	検索	プロジ	፤//	P) 実行(R)	ウィンドウ(W)	ヘルプ(H)
	新規(N)		А	lt+シフト	+N >	2	COBOL プロジ	ジェクト	
	ファイノ	レを開く(.)					2	COBOL 그ド-	ファイル プロジェ	クト
۵,	ファイノ	レ・システム	からプロジェクトを	開く			िंद	リモ−ト COBO	L プロジェクト	
	最近(のファイル				>	۲ ۲	リモ−ト COBO	レコピーファイル	プロジェクト
	問じる	5(C)			Ctrl	+W	S	COBOL 1	小 テスト プロジュ	: / }
	すべて	(に) (閉じる(L)		Ct	rl+シフト	+W	e	COBOL/Java	相互運用機能	のプロジェクト
						_	12	COBOL JVM	フロジェクト	

以下の入力を行い、[次へ(N)]をクリックします。

プロジェクト名: "JavaCOBOLProj1"

プロジェクトテンプレート: "Micro Focus テンプレート(64 ビット)"

プロジェクト名(P): JavaCOBOLProj1	
プロジェクト テンプレートを選択	
132 ビット] 132 ビット] 1997 Micro Focus テンプレート [64 ビット]	
テンプレートの設定を構成	<u> ‡</u>
□ 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 ~	
✓ デフォルト・ロケーションの使用(D)	
ロケーション(L): C:¥workspace-interoperability¥JavaCOBOLProj1 参照(R)	
ファイル・システムを選択(Y): デフォルト \vee	
(2) (東石(R)) (次人(N))、 (総マノに) ませいけい	

4) JRE に [実行環境 JRE の使用] を選択し、[終了(F)] をクリックします。

JRE 東行環境 JRE の使用(V): プロジェクト固有の JRE を使用(S): Use default JRE 'AdoptOpen JDK' and 	workspace compile	rpreferences	JavaSE-17 AdoptOpenJD	∨ K ∨
	nonspace comple	, preferences		<u>JRE を構成</u>
?	< 戻る(B)	次へ(N) >	終了(F)	キャンセル

JavaCOBOLProj1 プロジェクトが作成されます。

🔓 COBOL	×	ੴ. Navigat
> 🛃 COBOL	Java Java	Proj1 Proj2
> 🔁 COBOL	BOLI	Proj1

5) プロジェクトに対する文字コード設定を行います。

5.2の手順を実施してください。

 JavaCOBOLProj1 プロジェクトを選択し、マウスの右クリックによりコンテキストメニューを開き、 [プロパティ(R)]を選択します。

構成	>
ソース(S)	>
プロパティ(R)	Alt+Enter

左側のツリーより [Micro Focus] > [プロジェクト設定] > [COBOL] を選択し、以下の選択

を行ったうえで、[適用して閉じる]をクリックします。

出カパス: "src"

パッケージ名: "com.sample"

補足)

出カパス "src" が、Java プログラムのソースフォルダになります。このフォルダ配下に、COBOL プログラムにアクセスするためのラッパープログラムが生成されます。

ード・スタイル		
ンパイラー	設定	值 '
ビルド・パス	▼ 一般	
Focus	文字セット	ASCII
ダー	ソース エンコーディング	ANSI
ドパス	COBOL 方言	Micro Focus
ド構成	ソース フォーマット	固定
江クト設定	デバッグ用にコンパイル	はい
DL	EXIT PROGRAM を GOBACK として処理	ANSI
-	「「「」「「」「」「」「」「」「」「」「」」「」「」」「」」「」」「」」「」」	いいえ
	.GNT にコンパイル	いいえ
	✓ 出力	
	指令ファイルを生成する	いいえ
	リストファイルを生成	いいえ
	コードカバレッジを有効にする	
	プロファイラを有効にする	
	✓ Java Interoperability	
	出力パス	src
	パッケージ名	com.sample
	✓ エラ-/警告	
	警告レベル	回復可能なエラーを含める(レベル E)
	最大エラー数	100
	パッケージ名 Java ソース パッケージ名を指定します	
	COBOL コンパイル設定:	
	CHARSET"ASCII" SOURCE-ENCODING"ANSI" DIALI EXITPROGRAM"ANSI" java-output-path"src" java- ERROR"100"	ECT"MF" SOURCEFORMAT"fixed" NOLIST anim package-name"com.sample" WARNING"1" MAX-
		デフォルトの復元(T) 適用(I

 JavaCOBOLProj1 プロジェクトを選択し、マウスの右クリックによりコンテキストメニューを開き、 [新規作成(N)] > [COBOL プログラム]を選択します。

🔓 COBOL 🗙 🗠 Na	ovigat	😫 Applicat	🔜 サーバー	🛄 Analysis	-	
> 🛃 COBOLJavaProj1				▼ □	z [⊕] ŵ	8
> 🛃 JavaCOBOLProj1		新規作成(N)		>	솔	COBOL JVM プロジェクト
		表示方法(W)		Alt+シフト+W >		COBOL JVM ユニット テスト プロジェクト
		コピー 貼り付け 削除(D) 移動(V) 名前を変更(M) タスクのスキャン コード会た		Ctrl+C Ctrl+V 削除 F2		COBOL コピーファイル フロジェクト COBOL ブロジェクト 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 コピーファイル プロジェクト リモート COBOL コピット テスト プロジェクト
		インポート(i)		>		プロジェクト(R)
	4	エクスポート(O)			BŶ	COBOL コピーファイル
	8	更新(F)		F5	đ	COBOL プログラム

そのまま、[終了(F)]をクリックします。

COBOL プログラム

エディタで開くことができる COBOL プログラムを新規作成します。

含まれるプロジェクト:	JavaCOBOLProj1		参照
新規ファイル名:	Program1.cbl		
テンプレートを選択:			
📄 Micro Foc	us テンプレート		
		テンプレートの)設定を構成
🗌 テンプレートの参	参照		
場所:			参照
ファイルシス	テムを選択: default ∨		
0		総プロ	ナルバル 11.
()		/≪ J (F)	11/200

作成された Program1.cbl を、サンプルファイルを解凍したフォルダ内の JavaToCOBOL フォ ルダ配下の Program1.cbl の内容で上書きしてください。

 JavaCOBOLProj1 プロジェクトを選択の上、[ファイル(F)] > [新規(N)] > [その他(o)] を 選択します。

	68	KEST Web 9-CA				
	29	CICS Web サービス				
	<u></u>	Application Anal	ysis Server への接続			
		サンプル(X)				
		その他(o)		Ctrl+N		
9)	[Jav	ra] > [クラス]	を選択して、[次へ(N)]	をクリックします	t.	
2	- לול					
	71	ルタ入力				
	>	≽ J2EE			^	
	~	🔁 Java de Java プロジェク	75			
		る Java ワーキング	ブ・セット			
		創 ソース・フォルダ	r			
		 ・ ピッケージ ・ ジ ・ ジ ・ デ ・ ビージ ・ ビージ ・ ビージ ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・	ド・ファイルからの Java プロジェクト			
		⑥ 記録	1 77 177 50 300 20271			
		G /if±			*	
				-		-
	?)	< 戻る(B) 次へ(N) >	終了(F)	キャンセル	
	以下	の入力を行っ	たうえで、「終了(F)] を	フリックします。		
	11°m	t 2"	a comple"			
	11.95	$J = \mathcal{I}$. Con	n.sample			
			·			
	名前	ม์: "JavaCC	BOLMain"			
	名前 Jav	ົງ: "JavaCC a ງງວ	BOLMain"			
	名前 Jav 新	前: "JavaCC a クラス 見 Java クラスを作成し	BOLMain"			C
	名前 Jav 新想	① : "JavaCC a クラス 見 Java クラスを作成し ス・フォルダ(D):	BOLMain" ## JavaCOBOLProj1/src			②
	名前 Jav 新想	①:"JavaCC a クラス 見 Java クラスを作成し ス・フォルダ(<u>D</u>): _{T-ジ(K)} :	BOLMain" ## JavaCOBOLProj1/src			(Q) 参照(Q)
	名前 Jav 新想 ソーフ パック	 "JavaCC a クラス 見 Java クラスを作成し ス・フォルダ(D): ケージ(L): 	BOLMain" ## JavaCOBOLProj1/src			⑦ ● ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○
	名前 Jav 新想 ソーフ パッ?	① : "JavaCC a クラス 見 Java クラスを作成し ス・フォルダ(<u>D</u>): ケージ(<u>K</u>): にンクロージング型(Y):	BOLMain" att. JavaCOBOLProj1/src com.sample			⑦ ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ●
	名前 Jav 新想 ソージ パッパ 日 コ 名前	①: "JavaCC a クラス 見 Java クラスを作成し ス・フォルダ(<u>D</u>): ケージ(<u>K</u>): にソクロージング型(<u>Y</u>): (<u>M</u>):	BOLMain" ### JavaCOBOLProj1/src com.sample			② 参照(②) 参照(₩) 参照(₩)
	名前 Jav 新想 ソーフ パッパ 二 1 名師	①:"JavaCC a クラス 見 Java クラスを作成し ス・フォルダ(D): ケージ(<u>()</u>): [(<u>M</u>): 時子:	BOLMain" ます。 JavaCOBOLProj1/src com.sample JavaCOBOLMain () public(P) () package(S) O private(V)		② 参照(◎) 参照(◎) 参照(₩)
	名 f Jav 新 ガ フーン 、 、 、 、 、 、 、 、 、 、 、 、 、	①:"JavaCC a クラス 見 Java クラスを作成し ス・フォルダ(<u>D</u>): ケージ(<u>K</u>): にソクロージング型(<u>Y</u>): i(<u>M</u>): i子:	BOLMain" atj. JavaCOBOLProj1/src com.sample JavaCOBOLMain) ○ private(⊻) □ static(⊆)		(Q) 参照(Q) 参照(W)
	名 f Jav 新 ソーフ パ 一 る 節 スー/	①: "JavaCC a クラス 見 Java クラスを作成し ス・フォルダ(<u>D</u>): ケージ(<u>K</u>): にンクロージング型(<u>Y</u>): (<u>(</u>)): 時子: ((-クラス(<u>S</u>):	DBOLMain") _ private(V) _ static(C)		② ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ●
	名 f Jav 新 ソージ パッパ 一 名 修 スーパ パ	① : "JavaCC a クラス 見 Java クラスを作成し ス・フォルダ(D): ケージ(近): にンクロージング型(Y): 時子: (「-クラス(<u>S</u>): ターフェース(i):	BOLMain" at JavaCOBOLProj1/src [com.sample] JavaCOBOLMain ④ public(P) ○ package(S □ abstract(]) □ final(L) [java.lang.Object) O private(V) static(C)		参照(o) 参照(W) 参照(W) 参照(L) 違加(A)
	名 f Jav 新 ソ パッパ コ 	①: "JavaCC a クラス 見 Java クラスを作成し ス・フォルダ(D): ケージ(近): にソクロージング型(Y): 「(<u>M</u>): ほ子: (「-クラス(<u>S</u>): ターフェース(j):	BOLMain" atj. JavaCOBOLProj1/src com.sample JavaCOBOLMain) ○ private(V) □ static(C)		②● ● 新照(④) ● 新照(●) ● 新照(●) ● 新照(●) ● 新照(●) 」 道加(▲) ■ 影 ま(□)
	名 fi Jav 新 ソープ マパ コ 一 名 修 スーパン	①: "JavaCC a クラス 見 Java クラスを作成し ス・フォルダ(<u>D</u>): ケージ(<u>K</u>): にソクロージング型(<u>Y</u>): (<u>(一クラス(S)</u>): ターフェース(<u>)</u>):	BOLMain") O private(V) Static(C)		 参照(Q) 参照(W) 参照(W) 参照(U) 参照(E) 追加(A) 除去(B)
	名 f Jav 新 ソ パ ツ し コ ー 前 舗 ス イン 作 の の の の の の の の の の の の の	① : "JavaCC a クラス 見 Java クラスを作成し ス・フォルダ(<u>D</u>): ケージ(<u>C</u>): たックロージング型(<u>Y</u>): 可(<u>M</u>): 時子: (「-クラス(<u>S</u>): ターフェース(<u>i</u>):	BOLMain" ます。 JavaCOBOLProj1/src com.sample JavaCOBOLMain ④ public(P) ○ package(G □ abstract(]) □ final(L) java.lang.Object) O private(V) static(C)	<pre>protected(])</pre>	 参照(w) 参照(W) 参照(W) 参照(L) 違加(A) 除去(R)
	名 Jav Jav デ ソ パ ロー名修 スイン 作	 "JavaCC a クラス オリンスを作成し オージ(ビ): オージ(ビ): エンクロージング型(ビ): ゴ(M): ボテ: ボークラス(S): ターフェース(i): ボオるメソッド・スタブの: 	BOLMain" ます。 JavaCOBOLProj1/src com.sample JavaCOBOLMain ④ public(P) ○ package(S □ abstract(I) □ final(L) java.lang.Object 鍵訳 □ public static void main(Strii □ ス-パークラスからのコンストラグ) private(V) static(C) static(C) rg[] args)(V) 7-(C)	<pre>protected(I)</pre>	《②… 参照(④)… 参照(W)… 参照(W)… 参照(E)… 追加(A)… 除去(B)
	名 Jav Jav 新 ソ パ □ 名修 スイソ 作	①: "JavaCC a クラス 見 Java クラスを作成し ス・フォルダ(<u>D</u>): ケージ(<u>K</u>): にンクロージング型(<u>Y</u>): ご(<u>M</u>): i子: ((-クラス(<u>S</u>): ターフェース(i): たするメソッド・スタブの:	BOLMain" ます。 JavaCOBOLProj1/src com.sample JavaCOBOLMain ④ public(P) ○ package(S □ abstract(D) □ final(L) java.lang.Object 躍択 □ public static void main(Strii □ スーパークラスからのコンストラク ✓ 継承された抽象メソッド(H)	2) ○ private(⊻) □ static(⊆) ng[] args)(⊻) Ø=(⊆)		 参照(Q) 参照(W) 参照(W) 参照(L) 違加(A) 除去(B)
	名 f i Jav 新 ソープ パ 二 名 修 スーパ 作 エ メン パ 二 、 、 、 、 、 、 、 、 、 、 、 、 、	 (1): "JavaCC a クラス a クラス ス・フォルダ(D): ケージ(近): たックロージング型(Y): ゴ(価): デテ: パークラス(S): ターフェース(i): たするメソッド・スタブの: パトを追加しますか?(デ 	BOLMain" ます。 JavaCOBOLProj1/src com.sample JavaCOBOLMain ④ public(P) ○ package(G □ abstract(]) □ final(L) java.lang.Object 躍択 □ public static void main(String □ スーパークラスからのコンストラク・ ☑ 継承された抽象メソッド(H) ンプレートの構成およびデフォルト値	 private(V) static(C) ing[] args)(V) ター(C) についてはごごを参照) 	<pre>protected(])</pre>	 参照(w) 参照(W) 参照(W) 参照(L) 遠加(A) 除去(R)
	名 Jav Jav デ ソ パ ロー名修 スイン 作 コン	 (1): "JavaCC a クラス a クラス ス・フォルダ(D): ケージ(近): ケージ(近): (1): (1):	BOLMain" ます。 JavaCOBOLProj1/src com.sample JavaCOBOLMain ④ public(P) ○ package(G abstract(]) □ final(L) java.lang.Object U はないになったいのはのは、Striu コスーパークラスからのコンストラク ど 継承された抽象メソッド(L) シブレートの構成およびデフォルト値 □ コメントの生成(G)) _ private(火)] static(() ing[] args)(火) ター(C) についてはごを参照)	<pre>protected(1)</pre>	 参照(Q) 参照(W) 参照(W) 参照(E) 追加(A) 除去(B)
	名 Jav Jav 新 ソープ ッ? コー 前舗 スイン 作 JX2	 "JavaCC a クラス a クラス a クラス a ノラス a ノラス b ノー c ノー	BOLMain" ます。 JavaCOBOLProj1/src com.sample JavaCOBOLMain の public(P) の package(s abstract(D) 「final(L) java.lang.Object 躍択 □ public static void main(Striit □ スーパークラスからのコンストラク ※ 継承された抽象メソッド(H) シブレートの構成およびデフォルト値 □ コメントの生成(G)	 private(V) static(C) ng[] args)(V) ター(C) についてはごを参照) 		 参照(g) 参照(g) 参照(g) 参照(g) 参照(g) 参照(g) 除去(g)
	名 Jav メープ パロー名修 スイン 作 JX	 (1): "JavaCC a クラス a クラス ス・フォルダ(D): ケージ(K): エンクロージング型(Y): ゴ(M): ボテ: パークラス(S): ターフェース(j): なするメソッド・スタブの: ハトを追加しますか?(デ 	BOLMain" ます。 JavaCOBOLProj1/src com.sample JavaCOBOLMain ④ public(P) ○ package(G abstract(D) ☐ final(L) java.lang.Object 躍択 □ public static void main(String □ スーパークラスからのコンストラク・ ⑦ 継承された抽象メソッド(L) シブレートの構成およびデフォルト値 □ コメントの生成(G)	 private(火) static(C) static(Q) ng[] args)(火) ター(C) についてはごごを参照) 	<pre> protected(]) </pre>	 参照(w) 参照(W) 参照(W) 参照(L) 違加(A) 除去(R)

JavaCOBOLProj1 プロジェクト配下の src¥com¥sample の下に JavaCOBOLMain.java が作成されます。 ¹COBOL... × ¹Navigat... ¹Compose App ²COBOLJavaProj1 ³COBOLJavaProj2 ⁴CoBOLJavaProj2 ⁴CoBOL プログラム ³Do DED プログラム ³Do DED プログラム ³Do DED プログラム ³Do DED Compose C

作成された JavaCOBOLMain.java を、エディター上でサンプルファイルを解凍したフォルダ内の JavatoCOBOL フォルダ配下の JavaCOBOLMain.java の内容で上書き保存してください。

10) JavaCOBOLProj1 プロジェクトを選択し、[実行(R)] > [実行構成(N)] をクリックします。

	実行	(R)	ウィンドウ(W)	ヘルプ(H)
	P	実行	テ点をリセット	
4	Q,	実行	, (R)	Ctrl+F11
	椮	デバ	ッグ(D)	F11
		実行	亍履歴(T)	>
	0	実行	T(S)	>
		実行	亍構成(N)	

[Java アプリケーション] を選択したうえで、マウスの右クリックによりコンテキストメニューを開き、 [新規構成(W)] を選択します。

11) 以下の入力を行い、[実行(R)] をクリックします。

名前: "JavaCOBOLMain"

メイン・クラス: "com.sample.JavaCOBOLMain"

名前(N): JavaCOBOLMain		
④ メイン (⋈)= 引数 ■ JRE % 依存関係	系 🧤 ソース 🖾	瓓城
プロジェクト(P):		
JavaCOBOLProj1		
メイン・クラス(M):		
com.sample.JavaCOBOLMain		

[引数] タブを選択

VM 引数: "-Djava.library.path=New_Configuration.bin"

名前(N): JavaCOBOLMain		
④メイン (⋈= 引数 副 JRE % 依存関係 りつ	τ 🗖	
プログラムの引数(A):		
-Diava,library.path=New Configuration,bin		
5 51 - 5		
以下の結果がコンソールビューに表示されま	す。	
COBOL 00000001 書		
COBOL 000000003 赤		
COBOL 000000004 緑		
Prime number from Java		
2		
3		
5		
7		
11		
13		
17		
17		
19		
23		
27		

このサンプルでは、Java から色名称の配列を COBOL に渡し、COBOL 側で出力しています。 COBOL からは素数のリストを Java に戻し、その結果を Java 側で出力しています。

4.3.2 COBOL と Java を別プロジェクトで利用

前項では、Java 資産と COBOL 資産を同じプロジェクト内に配置しました。しかし、同時開発を行わない限り、別々に管理されることが一般的です。本項では、COBOL, Java の2つのプロジェクトを使用した COBOL/Java 相互運用機能を利用する方法を紹介します。

1) Visual COBOL for Eclipse の起動

Windows スタートメニューより、[Micro Focus Visual COBOL] > [Visual COBOL for Eclipse] を選択して、Visual COBOL for Eclipse を起動します。

ワークスペースは任意のフォルダでかまいません。以降の手順では、c:¥workspacejavacollaborate を使用します。

2) ワークスペースに対する文字コード設定を行います。

5.1の手順を実施してください。

3) [ファイル(F)] > [新規(N)] > [その他(o)] を選択します。

<u>1</u>	Application Analysis Server への接続	
	サンプル(X)	
	その他(o)	Ctrl+N

4) [Java] > [Java プロジェクト] を選択したうえで [次へ] をクリックします。

ウィザード(W):	
フィルタ入力	
> 🥭 J2EE	^
Sava ワーインワ・ビット (2) インターフェース	
Ø 177 72 X	
₩ ソース・フォルダ	
₩ パッケージ	
※ 既存 Ant ビルド・ファイルからの Java フロジェクト	
10 記録 10 注釈	
 ◎ 列挙型 	,

以下の入力を行い、[終了(F)]をクリックします。
 プロジェクト名: "JavaCOBOLProj2J"

[実行環境 JRE の使用] を選択

プロジェクト名(P): JavaCOBOLProj2J			
☑ デフォルト・ロケーションの使用(D)			
ロケーション(L): C:¥workspace-interoperability¥JavaCOBC	OLProj2J		参照(R)
JRE			
● 実行環境 JRE の使用(V):	Java	SE-17	~
○ プロジェクト固有の JRE を使用(S):	Add	ptOpenJDK	~
Use default JRE 'AdoptOpenJDK' and workspace co	ompiler preferences		<u>JRE を構成</u>
プロジェクト・レイアウト			
 ブロジェクト・フォルダをソースおよびクラス・ファイルのルートと ソースおよびクラス・ファイルのフォルダーを個別に作成(C) 	として使用(U)		<u>既定値を構成</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(ロ):			~ 選択(E)
モジュール module-info.java を作成(M)			
2	< 戻る(B)	次^(N) > 約	3了(F) キャンセル
		をクロックレ.キオ	
(ースペクティブの切り替えタイアログで 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 ラー、型階層、および Java 固有のナビ	は、[UUUス(N)] ポートするために設計 【ゲーション・アクション?	されています。パック を提供します。	ージ・エクスプロー
(ースペクティブの切り替えタイアログで 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 ラー、型階層、および Java 固有のナビ コ 常にこの設定を使用する(R)	は、 しいいん(N)」 ポートするために設計 プゲーション・アクション?	されています。パック を提供します。	r−ジ・エクスプロ−
(ースペクティブの切り皆スタイアログで 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 ラー、型階層、および Java 固有のナビ]常にこの設定を使用する(R)	は、 [いいえ(N)] ポートするために設計 ゲーション・アクション・ パースペクティブを	されています。パック を提供します。 開く(O)	∽−ジ・エクスプロ− いいえ(N)
 スペクティブの切り替えタイアログで 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 ラー、型階層、および Java 固有のナビ 常にこの設定を使用する(R) avaCOBOLProj2J プロジェクトが作 	は、 [いいえ(N)] ポートするために設計 パーション・アクション パースペクティブを F成されます。	されています。パック を提供します。 開く(O)	r−ジ・エクスプロ− いいえ(N)
(ースペクティブの切り替えタイアログで 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 ラー、型階層、および Java 固有のナビ □常にこの設定を使用する(R) avaCOBOLProj2J プロジェクトが作 とg COBOL × 哈 Navigat	は、 しいしん(N)」 ポートするために設計 ゲーション・アクション・ パースペクティブを F成されます。	されています。 パック を提供します。 開く(O)	r−ジ・エクスプロ− いいえ(N)
 (ースペクティブの切り替えタイアログで 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 ラー、型階層、および Java 固有のナビ 常にこの設定を使用する(R) avaCOBOLProj2J プロジェクトが作 COBOLJavaProj1 ご COBOLJavaProj1 ご COBOLJavaProj1 ご JavaCOBOLProj2J ご JavaCOBOLProj2J 	は、 [いいえ(N)] ポートするために設計 ゲーション・アクション パースペクティブを F成されます。	されています。 パック を提供します。 開く(O)	rージ・エクスプロー いいえ(N)
 (ースペクティブの切り替えタイアログで 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 ラー、型階層、および Java 固有のナビ コ 常にこの設定を使用する(R) avaCOBOLProj2J プロジェクトが作 COBOLJavaProj1 ご COBOLJavaProj1 ご COBOLJavaProj1 ご COBOLJavaProj1 ご COBOLJavaProj1 ご COBOLJavaProj1 ご JavaCOBOLProj2J JavaCOBOLProj2J avaCOBOLProj2J プロジェクトが表 	は、 [いいえ(N)] ポートするために設計 ゲーション・アクション? パースペクティブを F成されます。	されています。パック を提供します。 開く(O)	rージ・エクスプロー いいえ(N)
 (ースペクティブの切り替えタイアログで 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を開きますか? このパースペクティブは、Java 開発をサ ラー、型階層、および Java 固有のナビ コ 常にこの設定を使用する(R) avaCOBOLProj2J プロジェクトが依 COBOLJavaProj1 ご COBOLJavaProj1 ご COBOLJavaProj1 ご COBOLJavaProj1 ご COBOLJavaProj2 ご JavaCOBOLProj2J avaCOBOLProj2J プロジェクトが表 こクスプローラービューの右上をクリックし 	は、「しいしん(N)」 ポートするために設計 ゲーション・アクション? パースペクティブを F成されます。	されています。パック を提供します。 開く(O) 「 マイズ(F)]をク	Tージ・エクスプロー いいえ(N)
 (ースペクティブの切り替えタイアログで 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 ラー、型階層、および Java 固有のナビ コ 常にこの設定を使用する(R) avaCOBOLProj2J プロジェクトが作 COBOL × 哈 Navigat ご COBOLJavaProj1 ご COBOLJavaProj1 ご COBOLJavaProj1 ご COBOLJavaProj1 ご COBOLJavaProj2 ご JavaCOBOLProj2J avaCOBOLProj2J プロジェクトが表 こクスプローラービューの右上をクリックし COBOL × 哈 Navigat ●	は、「しいしん(N)」 ポートするために設計 ゲーション・アクション? パースペクティブを F成されます。 長示されない場合 ,、「フィルタとカスタ サーバー ▲ Analysis	されています。パック を提供します。 開く(O) ロマイズ(F)]をク	rージ・エクスプロー いいえ(Ν)
 (ースペクティブの切り替えタイアログで 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 ラー、型階層、および Java 固有のナビ コ常にこの設定を使用する(R) avaCOBOLProj2J プロジェクトが作 COBOL × 哈- Navigat ご COBOLJavaProj1 ご COBOLJavaProj1 ご JavaCOBOLProj2J プロジェクトが速 COBOLJavaProj2 ご JavaCOBOLProj2J プロジェクトが速 COBOLJavaProj2 avaCOBOLProj2J プロジェクトが速 COBOLLavaProj1 ご COBOLLavaProj2 avaCOBOLProj2J プロジェクトが速 COBOLLavaProj1 ご COBOLLavaProj1 ご COBOLLavaProj1 ご COBOLLavaProj1 ご COBOLLavaProj1 ご COBOLLavaProj1 ご COBOLJavaProj1 ご COBOLJavaProj1 … · ご COBOLJavaProj1 · ご COBOLJavaProj1 · ご COBOLJavaProj1 · ご · COBOLJavaProj2 · ご · COBOLJavaProj1 · ご · COBOLJavaProj2 · ご · COBOLJavaProj1 · ご · COBOLJavaProj2 · COBOLJavaProj2 · COBOLJavaProj1 · · · · · · · · · · · · · · ·	は、「しいしん(N)」 ポートするために設計 ゲーション・アクション? パースペクティブを F成されます。 東示されない場合 ,、「フィルタとカスタ サーバー ▲ Analysis マ 日 �	されています。/(ッケ を提供します。 開く(O) 「マイズ(F)」をク 見、、 このBol J	rージ・エクスプロー いいえ(N) リックします。
 (ースペクティブの切り替スタイアログで 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 ラー、型階層、および Java 固有のナビ コース、型階層、および Java 間角のナビ コース コース、型階層、および Java 間角のナビ コース コー コー コー	は、「しいしん(N)」 ポートするために設計 ゲーション・アクション? 「パースペクティブを 手成されます。 「スペクティブを テ、されない場合 、「フィルタとカスタ サーバー… ▲ Analysis… ▼ ■ �	されています。パック を提供します。 開く(O) 「マイズ(F)]をク 「、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、	Tージ・エクスプロー いいえ(N) リックします。 (M プロジェクト表示(R) レ要素(T)
 (ースペクティブの切り替えタイアログで 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 ラー、型階層、および Java 固有のナビ コースへの分ティブは、Java 開発をサ ラー、型階層、および Java 固有のナビ コースへの設定を使用する(R) avaCOBOLProj2J プロジェクトが作 COBOLJavaProj1 ご COBOLJavaProj1 ご COBOLJavaProj1 ご JavaCOBOLProj2J プロジェクトが基 プロジェクトが基 こクスプローラービューの右上をクリックし COBOLJavaProj1 ご COBOLJavaProj1 ご COBOLJavaProj2 avaCOBOLProj2J プロジェクトが基 こクスプローラービューの右上をクリックし ご COBOLJavaProj1 ご COBOLJavaProj1 ご COBOLJavaProj1 ご COBOLJavaProj1 ご GOBOLJavaProj1 ご GOBOLJavaProj1 ご GOBOLJavaProj1 ご GOBOLJavaProj1 ご GOBOLJavaProj1 ご JavaCOBOLProj2 ご GOBOLJavaProj1 ご COBOLJavaProj1 ご COBOLJavaProj1 ご COBOLJavaProj2 ご JavaCOBOLProj2 ご JavaCOBOLProj2 ご COBOLJavaProj2 ご JavaCOBOLProj2 ご COBOLJavaProj2 ご JavaCOBOLProj2 ご COBOLJavaProj2 ご JavaCOBOLProj2 ご この この この	は、「しいしん(N)」 ポートするために設計 ゲーション・アクション? 「パースペクティブを F成されます。 「スペクティブを F成されます。	されています。パック を提供します。 開く(O) ワマイズ(F)]をク いてイズ(F)]をク しています。 (No Comparison of the second s	rージ・エクスプロー いいえ(N) いいえ(N) リックします。 (M プロジェクト表示(R) レ要素(T) セットの選択解除(K) ロットの選択解除(K) ロットック選択解除(K)
 (ースペクティブの切り替えタイアログで 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 ラー、型階層、および Java 固有のナビ コ常にこの設定を使用する(R) avaCOBOLProj2J プロジェクトが作 GCBOL × 哈- Navigat ご? COBOLJavaProj1 ご? COBOLJavaProj1 ご? JavaCOBOLProj2J プロジェクトが速 Zクスプローラービューの右上をクリックし GCBOL × 哈 Navigat ● COBOLJavaProj1 COBOLJavaProj1 COBOLJavaProj2 JavaCOBOLProj2J プロジェクトが速 COBOLLavaProj1 ご? COBOLJavaProj2 Z COBOL × 哈 Navigat ● COBOLJavaProj2 Z COBOL × 哈 Navigat ● Applicat ● COBOLJavaProj1 Z COBOL × 哈 Navigat ● Applicat ● COBOLJavaProj1 Z COBOLJavaProj1 Z COBOLLavaProj1 Z COBOLJavaProj1 Z Z COBOLProj2 Z JavaCOBOLProj2 Z S COBOLProj2 Z S COBOLE Z S S S S S S S S S S S S S S S	は、しいしん(N)」 ポートするために設計 グーション・アクション? パースペクティブを F成されます。 東示されない場合 ハ、[フィルタとカスタ サーバー… ▲ Analysis…	されています。パック を提供します。 開く(O) ママイズ(F)]をク 呼吸の の にのBOL J トップレベリ ワーキング・ アクティブ ジ (1)ウィンド	アージ・エクスプロー いいえ(N) リックします。 ペM プロジェクト表示(R) レ要素(T) セットの選択解除(K) 3ワーキング・セットの編集(E) ・ワーキング・セットの編集(E)

> >

🍸 วีปซ่าหาว่ามด่า 🍸 ユーザー・วามด่า 🕻	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	目は隠されます):		
1			
☑ 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			
□ 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		~	
	ОК	キャンセル	
	U.K.	112 C/V	

 JavaCOBOLProj2J プロジェクトを選択したうえで、マウスの右クリックによりコンテキストメニュー を開き、[プロパティ(R)]を選択します。

構成	>
ソース(S)	>
プロパティ(R)	Alt+Enter

左側のツリーより [Java のビルド・パス] を選択し、[ライブラリー(L)] を選択します。



[COBOL JVM 実行時システム] を選択し、[次へ(N)] をクリックします。

COBOL JVM 実行時システム

ロケーション: C:¥Progr	am Files (x86)¥Micro	o Focus¥Visual (COBOL	
?	< 戻る(B)	次へ(N) >	終了(F)	キャンセル

"COBOL JVM 実行時システム" が追加されたことを確認したうえで、[適用して閉じる] 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(O) 🟮 モジュール依衣	字関係(M)
ビルド・パス上の JAR およびクラス・フォルダー(T):	
✓ ♣ モジュールパス	JAR の追加(J)
> ≥ 1/1 × 0/2 × 0	外部 JAR の追加(X)
> 🛋 COBOL JVM 実行時システム	変数の追加(V)
	ライブラリーを追加(i)
	クラス・フォルダの追加(C)
	外部クラス・フォルダーを追加(D)
	編集(E)
	除去(R)
	JAR ファイルのマイグレーション(M)
	適用(L)
	適用して閉じる キャンセル

- 7) JavaCOBOLProj2J プロジェクトを選択したうえで、[ファイル(F)] > [新規(N)] > [その他
 - (o)] を選択します。

<u> – 1</u>	Application	Analysis	Server	への接続
-------------	-------------	----------	--------	------

	サンプル(X)	
Ċ	その他(o)	Ctrl+N
[]		++

[Java] > [クラス] を選択し、[次へ(N)] をクリックします。

ウィザード(W):	
74ルタ入力	
> 🔁 Gradle	^
> 🤁 J2EE	
_ <u><u></u><u></u><u></u><u></u><u></u><u></u><u></u><u></u><u></u><u></u><u></u><u></u><u></u><u></u><u></u><u></u><u></u><u></u><u></u></u>	
C 252	
● ジース・フォルダ	
※ 既存 Ant ビルト・ファイルからの Java フロシェクト ◎ 記録	
	J
	_
(?) < 戻る(B) 次へ(N) > 終了(F) キャンセル	
以下の入力を行い、「終了(F)] をクリックします。	

)人ノノで1」レッ、 レルミ 」 F)] でクリックしまり。

パッケージ: "com.sample"

名前: "JavaCOBOLMain"

Java クラス 新規 Java クラスを作成します。				
ソース・フォルダ(D):	JavaCOBOLProj2J/src	参照(o)		
パッケージ(K):	com.sample	参照(W)		
□ エンクロージング型(Y):		参照(W)		
名前(M): 修飾子:	JavaCOBOLMain			
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・スタブの選択 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) 必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ごごを参照) コメントの生成(G)				
?	< 戻る(B) 次へ(N) > 終了(F)	キャンセル		

作成された JavaCOBOLMain.java を、サンプルファイルを解凍したフォルダ内の JavatoCOBOL フォルダ配下の JavaCOBOLMain.java の内容で上書きしてください。 この時点では、9 行目がエラーとなりますが無視してください。

8) [ファイル(F)] > [新規(N)] > [COBOL プロジェクト] を選択します。

ファイ	ʹル(F)	編集(E)	リファクタリング	ナビゲート(N)	検索	プロシ	፲/፲	P)	実行(R)	ウィンドウ(W)	۸J
	新規((N)		А	lt+シフト	+N >	B	CO	BOLプロシ	ジェクト	
	ファイ	ルを開く(.)					2	CO	BOL JR-	ファイル プロジェ	:クト
	ファイ	ル・システム	からプロジェクトを	開く			<u>ع</u>	IJŦ	- h COBC)L プロジェクト	

以下の入力を行い、[終了(F)] をクリックします。 プロジェクト名:"JavaCOBOLProj2C"

プロジェクトテンプレート: "Micro Focus テンプレート(64 ビット)"

COBOL プロジェクト

ワークスペースまたは外部の場所にCOBOL プロジェクトを作成します。

	プロジェクト名(P): JavaCOBOLProj2C プロジェクト テンプレートを選択 「PA Micro Focus テンプレート [32 ビット] での Micro Focus テンプレート [64 ビット]		<u>テンプレートの設定を構成…</u>
			关照
	ファイルシステムを選択: default >		37 Attess
	JITTY JITT DECKED (
	✓ デフォルト・ロケーションの使用(D)		
	ロケーション(L): C:¥workspace-interoperability¥Ja	vaCOBOLProj2C	参照(R)
	ファイル・システムを選択(Y): デフォル	~ ~	
	?	終	了(F) キャンセル
	JavaCOBOLProj2C プロジェクトが作成さ	れます。	
	°coboL… × धि- Navigat…		
	 ² COBOLJavaProj1 ² COBOLJavaProj2 ² COBOLJavaProj2 ² JavaCOBOLProj1 ² JavaCOBOLProj2C ² JavaCOBOLProj2J ² 		
9) 7	プロジェクトに対する文字コード設定を行いま	す。	
	5.2 の手順を実施してください。		
10) J	avaCOBOLProj2C プロジェクトを選択し	マウスの右クリ	リックによりコンテキストメニューを開き、
[プロパティ(R)] を選択します。		
	構成	>	
	ソース(S)	>	
	プロパティ(R)	Alt+Enter	

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。

[Micro Focus] > [プロジェクト設定] > [COBOL] を選択

追加指令:半角スペースをデリミタとして、以下の2つを入力

- java-output-path"..¥JavaCOBOLProj2J¥src"
- java-package-name"com.sample"

7ィルタ入力	COBOL
 > リソース Coverage > Micro Focus ビルダー ビルドパス > ビルド構成 > プロジェクト設定 > COBOL コンテナー ビルド環境 	フィルタテキス 設定 ▼ 一般 文字セ ソース: COBO
	1

設定	值
▼ 一般	
文字セット	ASCII
ソース エンコーディング	ANSI
COBOL 方言	Micro Focus
ソース フォーマット	固定
デバッグ用にコンパイル	はい
EXIT PROGRAM を GOBACK として処理	ANSI
詳細	いいえ
.GNT にコンパイル	いいえ
✓ 出力	
指令ファイルを生成する	いいえ
リストファイルを生成	いいえ
コードカバレッジを有効にする	false
プロファイラを有効にする	false
✓ Iラ-/警告	
警告レベル	回復可能なエラーを含める(レベル E)
最大エラー数	100
✔ 追加指令	
追加指令	java-output-path"¥JavaCOBOLProj2J¥src" java-package-name"com.sample"

[Micro Focus] > [ビルド構成] > [リンク] を選択

ターゲットの種類: "単一ネイティブライブラリファイル"

7ኅルタ入力	リンク	
> リソース Coverage マ Micro Focus	New Configuration [使用中]	✓ 構成の管理
ビルダー ビルド パス 〜 ビルド編成	フィルタテキストを入力	
> COBOL イベント	設定 → Linkage	值
ディブロイ	出力の名前	JavaCOBOLProj2C
- 115775	出力パス	New Configuration.bin
> 929	エントリポイント	
> COBOL	ターゲットの種類	単一 ネイティブライブラリ ファイル

11) JavaCOBOLProj2C プロジェクトを選択し、マウスの右クリックによりコンテキストメニューを開き、 [新規作成(N)] > [COBOL プログラム] を選択します。

> 🛃 JavaCOBOLProj2C					public static void ma:
> 📂 JavaCOBOLProj2		新規作成(N)	>	ピ	COBOL JVM プロジェクト
		表示方法(W)	Alt+シフト+W >		COBOL JVM ユニットテスト プロジェクト
	D	วย่-	Ctrl+C	1990 1990	
C	Ē	貼り付け	Ctrl+V		
1	×	削除(D)	削除	200 129	COBOL/Lava 相互運用機能のプロジェクト
		移動(V)		2	リモート COBOL JVM プロジェクト
		名前を変更(M)	F2	1	リモート COBOL コピーファイル プロジェクト
		タスクのスキャン		1	リモート COBOL プロジェクト
		コード分析	>	R	リモート COBOL ユニット テスト プロジェクト
		インポート(i)	>	2	プロジェクト(R)
R	4	エクスポート(O)		R\$	COBOL コピーファイル
	ะ	更新(F)	F5	đ	COBOL プログラム

そのまま、[終了(F)]をクリックします。

COBOL プログラム

エディタで開くことができる COBOL プログラムを新規作成します。

含まれるプロジェクト:	JavaCOBOLProj2C		参照
新規ファイル名:	Program1.cbl		
テンプレートを選択:			
📄 Micro Foci	us テンプレート		
		テンプレート	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 ~		
?		終了(F)	キャンセル

作成された Program1.cbl を、サンプルファイルを解凍したフォルダ内の JavatoCOBOL フォ ルダ配下の Program1.cbl の内容で上書きしてください。

自動でビルドが行われ、前手順で指定した追加指令によって、JavaCOBOLProj2J プロジェクト配下の src¥com¥sample フォルダ配下に Program1.native_sig が作成されます。

COBOL X Kar Navigat 😤 Applicat
> 🛱 COBOLJavaProj1
> 🛃 COBOLJavaProj2
> 🛃 JavaCOBOLProj1
✓ 🛃 JavaCOBOLProj2C
> 垣 COBOL プログラム
> 🗁 New_Configuration.bin
✓ → JavaCOBOLProj2J
> 🗁 bin
🗸 🗁 src
🗸 🗁 com
🗸 🗁 sample
🚺 JavaCOBOLMain.java
Program1.native_sig

補足)

表示されない場合は、src¥com¥sample フォルダを選択し、マウスの右クリックによりコンテキス

トメニューを開き、「更新	(F)	1 を選択してください。	
--------------	-----	--------------	--

 ✓ JavaCOBOLProj2J > → bin > → src > → (→ com) 				
✓		新規作成(N)		>
		表示方法(W)		Alt+シフト+W >
		⊐Ľ-		Ctrl+C
	Ē	貼り付け		Ctrl+V
	×	削除(D)		削除
		移動(V)		
		名前を変更(M)		F2
アウトライン × mp プログラ		タスクのスキャン		
		インポート(i)		>
> 🕑 Program1	4	エクスポート(O)		
	8	更新(F)	更新	F5

12) COBOL 呼出しに必要なラッパープログラムを生成するため、[実行(R)] > [外部ツール(E)] > [外部ツールの構成(E)] を選択します。

プロジェクト(<u>P</u>)	実行	(<u>R)</u> ウィンドウ(<u>W</u>) ヘル	プ(<u>H</u>)					
🗖 🕮 🗖 🗄	R	実行点をリセット			> -			
🛄 Analysis	Q,	実行(R)	Ct	trl+F11				
¥ 🗏 🙀	裪	デバッグ(D)		F11				
		実行履歴(T)		>	•••	••••••	••••5••	• ••
	0	実行(S)		>	le)			
		実行構成(N)			"P	rograml".		
		デバッグ履歴(H)		>	•			
	*	デバッグ(G)		>				
		デバッグの構成(B)			cur	s 10 value 2	,3,5,7,	11
	Q,	項目を検査						
	Θ	プログラム ブレークポイントを	E追加					
		ツール		>	urs 5.	4.		
	9	外部ツール(E)		>		(起動履歴なし)		
		Θ	procedure divisi	on usi		≢/开(₽)		`
		Θ	perform vary	ing i		外部ツールの構成(E)	

13) [プログラム]をダブルクリックしたうえで、以下の入力を行い、[実行(R)] をクリックします。

名前: "genjava-for-JavaCOBOLProj2C"

ロケーション:

"C:¥Program Files (x86)¥Micro Focus¥Visual COBOL¥bin64¥genjava.exe" 作業ディレクトリー:

"C:¥workspace-interoperability¥JavaCOBOLProj2J¥src"

引数:

"JavaCOBOLProj2C -p Program1 -k com.sample"

補足)

ロケーション

上記で指定している genjava.exe は、Visual COBOL 製品のインストール先がデフォルトの 場合となります。異なるフォルダにインストールした場合は、<製品インストールフォルダ

>¥bin64¥genjava.exe を指定してください。

作業ディレクトリー

さきほど作成した JavaCOBOLProj2J プロジェクト配下の src フォルダまでの絶対パスを指 定してください。

引数

最初に指定している JavaCOBOLProj2C は、さきほど作成した COBOL プロジェクト "JavaCOBOLProj2C"の成果物である JavaCOBOLProj2C.dll を指定しています。

名前(N): genjava-for-JavaCOBOLProj2C			
□ ハフ ()* 更新 (<u>ob</u> C) / レア (ob 境境 (回 共通(C)) □ケーション(L):			
C:¥Program Files (x86)¥Micro Focus¥Visual COBOL¥bin64¥genjava.exe			
	ワークスペースの参照(P)	ファイル・システムの参照(E)	変数(I)
作業ディレクトリー(D):			
C:¥workspace-interoperability¥JavaCOBOLProj2J¥src			
	ワークスペースの参照(K)	ファイル・システムの参照(M)	変数(B)
引数(A): JavaCOBOLProj2C -p Program1 -k com.sample			<
注: スペースを含む引数は二重引用符 (*) で囲んでください。		[変数(S)
	コマンド行を表示(W) 前回保管した状態に戻す(V)	適用(Y)
		実行(R)	閉じる

JavaCOBOLProj2J プロジェクト配下の src¥com¥sample 配下を更新すると、

progs.java が生成されます。

°¦a cobol ×	₽ <mark>5.</mark> Navigat	Pe Applicat
 ² ²	Proj1 Proj2 Proj1 Proj2C グログラム nfiguration.bin	
V 🔁 JavaCOBOLI	Proj2J	
> 🗁 bin		
✓ ✓ → Com		
🗸 🗁 sar	mple	
J	JavaCOBOLMai	n.java
	Program1.nativ progs.java	e_sig

14) JavaCOBOLProj2J プロジェクトを選択したうえで、[実行(R)] > [実行構成(N)] を選択し

す。		
実行	(<u>R)</u> ウィンドウ(<u>W</u>) ヘルプ(<u>H</u>)	
3	実行点をリセット	
Q,	実行(R)	Ctrl+F11
核	デバッグ(D)	F11
	実行履歴(T)	>
0	実行(S)	>
	実行構成(N)	
	す。 実行 へ。 へ。 へ。	 ま行(R) ウインドウ(W) ヘルプ(H) 案 実行」点をリセット 案行(R) デバッグ(D) 実行展歴(T) 実行(S) 実行構成(N)

[Java アプリケーション] をダブルクリックします。

以下の入力を行い、[実行(R)]をクリックします。

名前: "JavaCOBOLProj2J"

メイン・クラス: "com.sample.JavaCOBOLMain"

名前(N) JavaCOBOLProj2J ● メイン (ω= 引数 ▲ JRE & 依存関係 リットス 電 環境 □ 共通(C) ● プロトタイプ プロジェクト(P): JavaCOBOLProj2J ● 第照(B)
◎ メイン (№= 引数 ▲ JRE % 依存関係 1 ソース ■ 環境 □ 共通(C) P プロトタイプ プロジェクト(P): JavaCOBOLProj2J 参照(B)
プロジェクト(P): JavaCOBOLProj2J 参照(B)
JavaCOBOLProj2J 参照(B)
メイン・クラス(M):
com.sample.JavaCOBOLMain 検索(S)
□ メイン・クラスの検索時にシステム・ライブラリーを組み込む(E)
□ メイン・クラスの検索時に継承されたメインを組み込む(H)
□ メインで停止(0)

[引数] タブを選択

VM 引数:

"-Djava.library.path=..¥JavaCOBOLProj2C¥New_Configuration.bin"

名前(N): JavaCOBO	OLProj2J
G メイン (⋈)= 引数	↓ JRE 🕎 依存関係 🧤 ソース 🚾 環境 🔲 共通(C) 🖻 プロトタイプ
ープログラ <mark>ムの51数(P</mark>	A);
	^
	v
	変数(1)
VM 引数(G):	
-Djava.library.pa	ath=¥JavaCOBOLProj2C¥New_Configuration.bin
	v
	変数(S)
Use the -XX:+	ShowCodeDetailsInExceptionMessages argument when launching
 記動時に @arg 	gfile を使用(R)
- 作業ディレクトリー:	
● デフォルト(U):	\${workspace_loc:JavaCOBOLProj2J}
○ その他(H):	
	ワークスペース(O) ファイル・システム(F) 変数(E) マ
	コマンド行を表示(W) 前回保管した状態に戻す(V) 適用(V)
	東行(R) 閉じる

4.3.1と同様の結果がコンソールビューに表示されます。

COBOL 000000001 青
COBOL 000000002 黄
COBOL 000000003 赤
COBOL 000000004 緑
Prime number from Java
2
3
5
7
11
13
17
19
23
27

4.4 COBOL アプリケーションの実行環境を Java 仮想マシンに移して Java 資産とともに Java として運用

この運用方法は、製品が提供する JVM COBOL 機能を利用します。別途チュートリアルが提供されていますので、以下のチュートリアルを参照ください。

https://www.amc.rocketsoftware.co.jp/manuals/VC90/Eclipse/index.html?t=GUID-D10DC512-FDEF-44CF-8A9B-32839729B493.html

Visual COBOL 9.0 for Eclipse のチュートリアルトップからは、以下のように進んでください。

[ここからはじめよう] > [Getting Started] > [JVM COBOL チュートリアル]

5 Visual COBOL for Eclipse 上の文字コード設定について

最新の Eclipse IDE 環境における文字コードのデフォルトは UTF-8 ですが、COBOL 資産は長年利用されていること から、多くは UTF-8 ではなく SJIS が採用されています。文字コード設定は、ワークスペース全体の設定と、プロジェクト 毎の設定の2つがあります。これらの設定と、プログラムファイルの文字コードに不整合があると、文字化けの原因となります。 本チュートリアルで使用するサンプルファイルは、文字コード SJIS を採用しているため、Eclipse IDE 上で SJIS 資産を 正しく扱うための設定手順について紹介します。

5.1 ワークスペースに対する文字コード設定

1) Visual COBOL for Eclipse を起動したうえで、[ウィンドウ(W)] > [設定(P)] をクリックします。

/ドウ(W) ヘルプ(H)	
新規ウィンドウ(N)	
エディター	>
外観	>
ビューの表示(V)	>
パースペクティブ(R)	>
ナビゲーション(G)	>
Spies 設定	>
設定(P)	
	パウ(W) へルブ(H) 新規ウインドウ(N) エディター 外観 ビューの表示(V) パースペクティブ(R) ナビゲーション(G) Spies 設定

 [一般] > [ワークスペース] を選択し、[テキスト・ファイル・エンコード] に "デフォルト(MS932)" を選 択したうえで、[OK] をクリックします。

フィルタ入力	ባ – ካአペース 🗘 🖛 🗧		
✓ 一般 ▲ Capabilities	ワークスペースの開始およびシャットダウン設定については、 <u>開始およびシャットダウン</u> を参照してください。		
Schema Associations	□ ネイティブのフックまたはポーリングを使用して更新(R)		
User Storage Service	✓ アクセス時に更新(S)		
Web ブラウザ			
> Iディタ	ワークスペース保管間隔 (分)(W): 5		
+-			
クイック検索	ウィンドウのタイトル		
コンテンツ・タイプ	✓ ワークスペース名を表示(E): workspace-interoperability		
サービス・ポリシー	□ パースペクティブ名を表示(T)		
> セキュリティー	ロークスパーフのコルパフを表示(F)。 C-Yworkspace-interoperability		
トレース			
> ネットワーク接続			
ハンドラーをリンク			
ハースペクテイノ プロジェクト・ネーチャ	プロジェクトを問く際に、参照するプロジェクトを問く、プロンプト		
ワークスペース			
	不明なフロジェクトの性質を以下のように報告(A): 唐告 ~		
> 外観			
検索	システム・エクスプローラーを起動するコマンド(X): explorer /E,/select=\${selected_resource_loc}		
比較/パッチ			
> Ant	ニキフレ・ファイル・エンコーピック		
AspectJ Compiler	テキスト・ファイル・コフュート(1) 新規デキスト・ファイルの1」を切り文子(F) ② デフォリト (1) (M(s dawn))		
	(€) 7 7 / 1/P(E) (Windows)		
> Java	○その他(O): MS932 ∨ ○その他(H): Windows ∨		
> Java 永続化			
< >	デフォルトの復元(T) 適用(L)		
? 迠 🖌 💿 📀	適用して閉じる キャンセル		

Preference Recorder のダイアログが表示された場合は、[Recorder enabled] のチェックを外し、 [キャンセル] をクリックします。

5.2 プロジェクトに対する設定

エクスプローラービュー上で、対象のプロジェクトを選択し、マウスの右クリックによりコンテキストメニューを開き、[プロパティ(R)]を選択します。

プロパティ(R)	Alt+Enter
ソース(S)	>
構成	>
比較対象(A)	>
チーム(E)	>

[Micro Focus] > [プロジェクト設定] > [COBOL] を選択し、以下の選択を行ったうえで、[適用して閉じる] をクリックします。

ソース エンコーディング: "ANSI"

71ルタ入力	COBOL		
> リソース			
Coverage			
Javadoc ロケーション			
> Javaエディタ	フィルタテキストを入力		
> Java コード・スタイル			
> Java コンパイラー	設定	值 ^	
Java のビルド・パス	▼ 一般		
✓ Micro Focus	文字セット	100	
ビルダー	ソース エンコーディング	ANSI	
ビルド パス	COBOL 方言	Micro Focus	
> ビルド構成	ソース フォーマット	固定	
✓ プロジェクト設定	デバッグ用にコンパイル	はい	
COBOL	EXIT PROGRAM を GOBACK として処理	ANSI	
コンテナー	詳細	いいえ	
ビルド環境	.GNT にコンパイル	いいえ	
指令の確定	✓ 出力		
> 実行時構成	指令ファイルを生成する	いいえ	
サーバー	リストファイルを生成	いいえ	
タスク・タグ	コードカバレッジを有効にする		
ビルダー	プロファイラを有効にする		
プロジェクト・ネーチャー	 Java Interoperability 		
プロジェクト・ファセット	出力パス	src	
プロジェクト参照	パッケージ名	com.microfocus.COBOL	
> 検証	✓ Iラ-/警告		
実行/デバッグ設定	警告レベル	回復可能なエラーを含める(レベル E)	
	最大エラー数	100 🗸	
	ソースエンコーデイング SOURCE-ENCODING はソースプログラムのエンコーディングをコンパイラに渡します。その後、RUNTIME-ENCODING 増金が増売されていたい、原引 室行袖のエンコーディングの注意に使用されます。ソースファイルに ITTE-R ITTE- COBOL コンパイル設定: CHARSET*ASCII* SOURCE-ENCODING*ANSI* DIALECT*MF* SOURCEFORMAT*fixed* NOLIST anim EXITPROGRAM*ANSI* java-output-path*src* java-package-name*com.microfocus.COBOL* WARNING*1* MAX-ERROR*100*		
?		デフォルトの復元(T) 適用(L) 適用して閉じる キャンセル	

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。