

# 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 10
- Adoptium OpenJDK17
- Visual COBOL 10.0 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

© Rocket Software, Inc. or its affiliates 1990–2024. All rights reserved. Rocket and the Rocket Software logos are registered trademarks of Rocket Software, Inc. Other product and service names might be trademarks of Rocket Software or its affiliates.



## 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

# **Rocket** software

## 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
- 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

## 3.1 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。

#### 3.2 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。



## 3.2.1 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/

## 3.2.2 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

- 1) 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。
- 3) 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 clientjson=get\_post. yaml Micro Focus Interface Mapping Toolkit v10.0.00259 (C) Copyright 1984-2024 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

## 3.2.3 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¥apiservice>cobol get_post-app.cbl gnt;						
Micro Focus	COBOL					
Version 10.0	(C) Copyr	ight 1984-20	024 Micro F	ocus or one o	of its affiliates.	
* チェック終了:	エラーはありま	ません - コード生	主成を開始しま	<u>व</u> े		
* Generating	get_post-	арр				
* Data:	138352	Code:	5727	Literals:	2288	
C:¥COBOLJav	aInterope	ability¥apise	ervice>cobol	get_post-pro	xy.cbl gnt;	
Micro Focus	COBOL					
Version 10.0	(C) Copyr	ight 1984-20	024 Micro F	ocus or one o	of its affiliates.	
* チェック終了:	エラーはありま	ません - コード生	主成を開始しま	ਰ		
* Generating get_post-proxy						
* Data: 1296 Code: 4338 Literals: 368						
C:¥COBOLJavaInteroperability¥apiservice>						
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。

© Rocket Software, Inc. or its affiliates 1990–2024. All rights reserved. Rocket and the Rocket Software logos are registered trademarks of Rocket Software, Inc. Other product and service names might be trademarks of Rocket Software or its affiliates.

## **Rocket** software

## 3.3 COBOL/Java 相互運用機能を利用

3.2 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。

#### 3.3.1 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。

1) 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) 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	Jル(F) 編集(E) リファクタリング ナビゲー	ト(N) 検索 プロジェク	ንト(	P) 実行(R) ウィンドウ(W) ヘルプ(H)
	新規(N)	Alt+シフト+N > 🧧	8	COBOL プロジェクト
	ファイルを開く(.)		ŝ	COBOL コピーファイル 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	1¢	Ì,	リモート COBOL プロジェクト
	最近のファイル	> ធ្វើ	ÌÌ.	リモート COBOL コピーファイル プロジェクト
	問じる(0)	Ctrl+W	ŝ	COBOL ユニット テスト プロジェクト
	すべて閉じる(L)	Ctrl+シフト+W	Î	COBOL/Java 相互運用機能のプロジェクト
		2	3	COROLINM ADATAL "

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



4)

プロジェクト名( <u>P</u> ): COBOLJavaProj1	
ブロジェクト テンプレートを選択 <sup>25</sup> Micro Focus テンプレート [32 ビット] <sup>26</sup> Micro Focus テンプレート [64 ビット]	
<u>テンプレートの参照</u>	<u>-トの設定を構成</u>
場所: ファイルシステムを選択: default ~	参照
✓ デフォルト・ロケーションの使用(D)	
ロケーション( <u>L</u> ): C:¥workspace-interoperability¥COBOLJavaProj1 ファイル・システムを選択( <u>Y</u> ): デフォルト 〜	参照( <u>R</u> )
(P)         (P)         終了(E)	キャンセル
IRE に [実行環境 JRE の使用] を選択し、[終了(F)] をクリックします	0

JRE			
<ul> <li>実行環境 JRE の使用(<u>V</u>):</li> </ul>		JavaSE-17	$\sim$
○ プロジェクト固有の JRE を使用( <u>S</u> ):		AdoptOpenJE	ж
O Use default JRE 'AdoptOpenJDK' and	workspace compiler preferen	ces	
			<u>JRE を構成</u>
<u></u>			
?	< 戻る( <u>B</u> ) 次へ( <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。





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。

6) 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 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	▶照		
場所:		参照	
ファイルシス	テムを選択: default ~		
?		終了(E) キャン	ソセル

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。

7) 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。

☐° 2°	ファイル フォルダー	
	サンプル(X)…	
Ŷ	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					
✓	フト ブ・セット			^	
● 1/2-2+-ス ③ クラス ● ソース・ノオルダ ● パッケージ					
<ul> <li>※ 既存 Ant ビルド・ファイルからの Java プロジェクト</li> <li>(1) 記録</li> <li>(2) 注釈</li> </ul>					
● 列挙型 >  → Java の実行/	デバッグ			~	
?	< 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/sro	c			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)	package(C)	O private(V)	<ul> <li>protected(T)</li> </ul>	•
	abstract(T) fi	nal(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	L 瞿択				
	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う	デフォルト値にた	ついては <u>ここ</u> 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

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	(R)	ウィンドウ(W)	ヘルプ(H)				
Ē	R	実行点をリセット						
٨	Q	実行	実行(R)					
	核	デバッグ(D)						
		実行	亍履歴(T)					
	0	実行	₸(S)					
		実行	亍構成(N)					
	**	デバ 実行 実行 実行	ッグ(D) 〒履歴(T) 〒(S) 〒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。

1				
	📑 🗗 闷 🗎 🗶 🖻 🍸 🗸	このダイアログから起動設定を		
	フィルタ入力			📑 - 選択した種類の構成を
	🕆 Apache Tomcat		^	🖻 - 選択した種類の起動構
	AJ AspectJ/Java Application		🎧 281日 も供成たてクラポ	
	AspectJ Load-Time Weaving Application		20 - 進択した構成をエクス小	
	🔤 COBOL/Java 相互運用機能のアプリケーシ	ΞV		📄 - 選択した構成をコピーす
	JVM COBOL JVM アプリケーション	-	新	県構成(W)
1	COBOL JVM ユニット テスト		-111.0	
	COBOL JVM リモート アプリケーション	P	新	現フロトタイフ(P)
		6	エク	<sup>7</sup> スポート(X)
		-		

10) 以下の入力を行い、[実行(R)] をクリックします。

名前: COBOLJavaProj1

主プログラム: [参照] をクリックし、"COBOLJavaProj1.dll" を選択



名前(N): COBOLJavaProj1	
- 🗔 一般 🦻 ソース 🐻 環境 🔲 共通(C) 🛋 JRE 🗞 クラスパス 🔎 実行時 💱 デバッグシンボル 🔬	動的分析 🤮 CTF 🥑 コンテナー
▼ COBOL プロジェクト(P)	
COBOLJavaProj1 参照	
▼ 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/COBOLJavaPrc 参照	
▼開始オブション	
作業ディレクトリ:	
参照	
▶ 実行オフション	
	前回保管した状態に戻す(V) 適用(Y)
	実行(R) 閉じる

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



何かキーを押して、アプリケーションを終了します。

# **Rocket** software

#### 3.3.2 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 3.3.1 で作成したものです。こちらの手順の実施前に、3.3.1 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	ェクト	P) 実行(R)	ウィンドウ(W)	ヘルプ(H
新規(	(N)		A	lt+シフト	+N >	Ø	COBOLプロS	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

```
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 ドット]	
[送 Micro Focus テンプレート [64 ビット]	
テンプ	<u>ノートの設定を構成</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(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 🗸	
? 終了(F)	キャンセル

COBOLJavaProj2 プロジェクトが作成されます。

© Rocket Software, Inc. or its affiliates 1990–2024. All rights reserved. Rocket and the Rocket Software logos are registered trademarks of Rocket Software, Inc. Other product and service names might be trademarks of Rocket Software or its affiliates.



	🔓 COBOL .	🗙 🕾 Navigat			
	> 🛃 COBC > 🛃 COBC	DLJavaProj1 DLJavaProj2			
3)	プロジェクトに対	する文字コード設定を行い	います。		
	5.2 の手順を	実施してください。			
4)	COBOLJava	Proj2 プロジェクトを選択し	し、マウスの	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。			
	> COBOLJavaProi そのまま、[終] 含まれるプロジェ 新規ファイル名: テンプレートを選 回 テンプレートを選 場所: ファイ川	。 新規作成(N) 表示方法(W) □ 2ℓ- ■ 払り付け ■ 払り付け ■ 利除(D) 参勒(V) 4 朝を変更(M) クスクのスキャン □-ド分析 127.ポート(D) 127.ポート(C) ■ 更新(F) T (F)]をクリックします。 ● でOBOLJavaProj2 Program1.cbl 比け ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・	Alt+シブト+W > Ctrl+C Ctrl+V 前原 52 54 54 54 54 54 54 54 54 54 54 54 54 54	<ul> <li>COBOL JVM ブロジェクト</li> <li>COBOL JVM ユニット テスト ブロジェクト</li> <li>COBOL Jビーフィ/II ブロジェクト</li> <li>COBOL ユニット テスト プロジェクト</li> <li>COBOL JLニット テスト プロジェクト</li> <li>COBOL JLニット テスト プロジェクト</li> <li>マモート COBOL コピーフィ/II プロジェクト</li> <li>マモート COBOL コピーフィ/II プロジェクト</li> <li>マモート COBOL コピーファイIII</li> <li>COBOL JビーファイIII</li> <li>COBOL JビーファイIII</li> <li>COBOL ブログラム</li> </ul> <b>デンプレートの</b> 設定を構成… 参照…	
	?			終了(F) キャンセル	

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。



実行	( <u>R</u> )	ウィンドウ( <u>W</u> )	ヘルプ( <u>H</u> )
R	実行	テ点をリセット	
Q	実行	Ţ(R)	
椮	デバ	、ッグ(D)	
	実行	亍履歴(T)	
0	実行	₸(S)	
	実行	亍構成(N)	

[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。

フィルタ入力			
🔤 COBOLJavaProj1			^
IVM COBOL JVM アプリケーション	,		
🚾 COBOL JVM ユニット テスト			
搣 COBOL JVM リモート アプリク	アーショ	シ	_
🗔 COBOL アプリケーション			
🔄 COBOL ユニット テスト	Ľ	新規構成(W)	
😨 Debug Adapter Launcher	P	新規プロトタイプ	(P)

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"

名前(N) COBOLJavaProj2		
🗔 一般 🧤 ソース 🔤 環境 📑 共通(C) 🔊 実行時 🖏 デバ	ッグシンボル 💪 動的分析 😂 CTF 🥑 コンテナー	
(注:ここで定義された変数は、任意の現在の設定値、または任意の指 環境スクリプト内の設定値を上書きします。)	旨定された	
変数 JAVA_HOME CLASSPATH	値 C:¥Program Files (x86)¥Micro Focus¥Visual COBOL¥Adopt %CLASSPATH%;¥.¥COBOLJavaProj1¥bin	<b>追加(A)</b> 攝集(E) 削除(R)
実行する環境スクリブト: 場所: ファイルがブロジェクト内にある場合、絶対バスは相対バス パラメータ: 回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	になります。	参照
	前回保管した状態に戻す	[V] 適用(Y)
	実行(R)	閉じる
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、3.3.1 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

© Rocket Software, Inc. or its affiliates 1990–2024. All rights reserved. Rocket and the Rocket Software logos are registered trademarks of Rocket Software, Inc. Other product and service names might be trademarks of Rocket Software or its affiliates.



### してください。

3.3.1 と同じ結果が戻されます。

オリジナル配列順序>>>
赤
禄
青
橙
鞋
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
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。

## 3.4 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

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.microfocus.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。

## **Rocket** software

## 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.microfocus.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:¥workspace-interoperability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。



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	
10号 Micro Focus テンプレート [32 ピット]	
Micro Focus 7770-F [64 29F]	
	a 20. m + 44 - 4
$\overline{r}$	<u>≻の設定を構成</u>
□ テソフレートの参照	
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 🗸	
? < 戻る(B) 次へ(N) > 終了(F)	キャンセル

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				
● 実行環境 JRE の使用(V):			JavaSE-17	~
<ul> <li>プロジェクト固有の JRE を使用(S);</li> </ul>			AdoptOpenJDk	( v
O Use default JRE 'AdoptOpenJDK' and y	workspace compile	r preferences		
©				JRE を構成
	= 7 (0)	25 A (81) -	45 Z (F)	de constante II
(1)	< 戻(B)	沢へ(N) >	終∫(F)	キャンセル

JavaCOBOLProj1 プロジェクトが作成されます。

🔓 COBOL	×	🔁 Navigat
> 🛃 COBOL	Java	Proj1
> 1 COBOL	Javal	Proj2
> 📂 JavaCO	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。

6) 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。

出力パス: "src"

```
パッケージ名: "com.sample"
```

補足)

出力パス "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。

値 ASCII ANSI Micro Focus 固定 はい ANSI いいえ いいえ いいえ いいえ
値 ASCII ANSI Micro Focus 固定 はい ANSI いいえ いいえ いいえ
値 ASCII ANSI Micro Focus 面定 (は) ANSI いいえ いいえ いいえ
値 ASCII ANSI Micro Focus 固定 はい ANSI いいえ いいえ いいえ いいえ
1個 ASCII ANSI Micro Focus 回定 はい ANSI いいえ いいえ いいえ いいえ
ASCII ANSI Micro Focus ฏิธิ (๕Ს ANSI มะบริ มะบริ มะบริ มะบริ
ASCII ANSI Micro Focus 固定 はい ANSI いいえ いいえ いいえ いいえ
ANSI Micro Focus 固定 はい ANSI いいえ いいえ いいえ いいえ
Micro Focus 固定 はい ANSI いいえ いいえ いいえ
固定 はい ANSI いいえ いいえ いいえ
はい ANSI いいえ いいえ いいえ いいえ
ANSI いいえ いいえ いいえ いいえ
いいえ いいえ いいえ いいえ
いいえ いいえ いいえ
いいえ いいえ
いいえ いいえ
いいえ
src
com.sample
回復可能なエラーを含める(レベル E)
100
CT"MF" SOURCEFORMAT"fixed" NOLIST anim package-name"com.sample" WARNING"1" MAX-
デフォルトの復元(T) 適用

7) 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。



Es COBOL × E- Na > En COBOLJavaProj1 > E COBOLJavaProj2	ivigat	🔁 Applicat 📠 サーバー	🔄 Analysis 👻 📄 🔄 🛺		8
> 🚰 JavaCOBOLProj1		新規作成(N)	>	알	COBOL JVM プロジェクト
		表示方法(W)	Alt+シフト+W >	<u>l</u> ∰ EŪ	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)	Ctrl+C Ctrl+V 削除 F2	i to to to to to	COBOL コピーファイル プロジェクト COBOL ブロジェクト COBOL ユニット テスト ブロジェクト COBOL/Java 相互運用機能のプロジェクト リモート COBOL J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
		タスクのスキャン コード分析	>	(학 (다)	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
		インポート(i)	>		プロジェクト(R)
	4	エクスポート(O)		BŶ	COBOL วษ์-ว <sub>ซ</sub> ิรไม
	8	更新(F)	F5	ð	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 Focus テンプレート	
	<u>テンプレートの設定を構成</u>
□ テンフレートの参照 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。

8) 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。

89	REST Web サービス	
20	CICS Web サービス	
<u></u>	Application Analysis Server への接続	
	サンプル(X)	
	その他(o)	Ctrl+N

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



フィルタ入力			
<ul> <li>&gt; ションジェンジェンジェンジェンジェンジェンジェンジェンジェンジェンジェンジェンジェン</li></ul>	7ト ቻ・ゼット ጚ ታ ዞド・ファイルからの Java プロジェクト	~	
?	< 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)] をクリックします。		
パッケージ:"cor	n.sample"		
名前: "JavaCC	)BOLMain"		
.lava カラス			
新規 Java クラスを作成し	ます。		C
1-7•7+∥.ď(∩).	lavaCOBOL Proi1/src		参照(o)
-X-74,09( <u>0</u> ):	Javacoboeriojiysie		
パッケージ( <u>K</u> ):	com.sample		参照( <u>W</u> )
パッケージ( <u>K</u> ): 「エンクロージング型(Y):	com.sample		<b>参照(<u>W</u>)</b> 参照( <u>W</u> )
パッケージ( <u>K</u> ): □ エンクロージング型(Y): 	com.sample		参照( <u>W</u> )
<ul> <li>パッケージ(<u>K</u>):</li> <li>エンクロージング型(<u>Y</u>):</li> <li>名前(<u>M</u>):</li> <li>修飾子:</li> </ul>	Image: State of the state o		参照( <u>₩</u> ) 参照( <u>₩</u> )
パッケージ( <u>K</u> ): □ エンクロージング型( <u>Y</u> ): 	JavaCOBOLMain <ul> <li>public(P) O package(C) O private(V)</li> <li>abstract(D) final(L) Static(C)</li> </ul>		参照( <u>W</u> )
パッケージ( <u>K</u> ): □ エンクロージング型( <u>Y</u> ): 	Image: static	protected(])	参照( <u>W</u> ) 参照( <u>W</u> ) 参照( <u>E</u> )
パッケージ( <u>K</u> ): □ エンクロージング型( <u>Y</u> ): 	Image: Static (C)       JavaCOBOLMain       Image: Static (D)       Image: St		参照( <u>W</u> ) 参照( <u>W</u> ) 参照( <u>E</u> ) 追加( <u>A</u> )
パッケージ( <u>K</u> ): □ エンクロージング型( <u>Y</u> ): 	Image: static		参照( <u>W</u> ) 参照( <u>W</u> ) 参照( <u>E</u> ) 追加( <u>A</u> )
パッケージ( <u>K</u> ): □ エンクロージング型( <u>Y</u> ): 	Image: Second state of the se		参照( <u>W</u> ) 参照( <u>W</u> ) 参照( <u>E</u> ) 追加( <u>A</u> ) 除去( <u>B</u> )
パッケージ( <u>K</u> ): 「エンクロージング型( <u>Y</u> ): 名前( <u>M</u> ): 修飾子: スーパークラス( <u>S</u> ): インターフェース(j): 作成するメソッド・スタブの	JavaCOBOLMain            • public(P)	O protected(])	参照( <u>W</u> ) 参照( <u>W</u> ) 参照( <u>E</u> ) 追加( <u>A</u> ) 除去( <u>R</u> )
パッケージ( <u>K</u> ): □ エンクロージング型( <u>Y</u> ): 名前( <u>M</u> ): 修飾子: スーパークラス( <u>S</u> ): インターフェース( <u>i</u> ): 作成するメソッド・スタブの	JavaCOBOLMain          ● public(P) ○ package(C) ○ private(V)	) protected[]	参照( <u>W</u> ) 参照( <u>W</u> ) 参照( <u>E</u> ) 追加( <u>A</u> ) 除去( <u>R</u> )
パッケージ( <u>K</u> ): □ エンクロージング型( <u>Y</u> ): 名前( <u>M</u> ): 修飾子: スーパークラス( <u>S</u> ): インターフェース( <u>j</u> ): 作成するメソッド・スタブの	JavaCOBOLMain            ・         ・         ・	O protected(])	参照( <u>W</u> ) 参照( <u>W</u> ) 参照( <u>E</u> ) 追加( <u>A</u> ) 除去( <u>R</u> )
パッケージ( <u>K</u> ): □ エンクロージング型( <u>Y</u> ): 名前( <u>M</u> ): 修飾子: スーパークラス( <u>S</u> ): インターフェース( <u>i</u> ): 作成するメソッド・スタブの コメントを追加しますか?(;	JavaCOBOLMain            ・         ・         ・	O protected(])	参照( <u>W</u> ) 参照( <u>W</u> ) 参照( <u>E</u> ) 追加( <u>A</u> ) 除去( <u>R</u> )
<ul> <li>パッケージ(<u>K</u>):</li> <li>「エンクロージング型(<u>Y</u>):</li> <li>名前(<u>M</u>):</li> <li>修飾子:</li> <li>スーパークラス(<u>S</u>):</li> <li>インターフェース(<u>)</u>):</li> <li>作成するメソッド・スタブの</li> <li>コメントを追加しますか?(:</li> </ul>	JavaCOBOLMain            ・         ・         ・		参照( <u>W</u> ) 参照( <u>W</u> ) 参照( <u>E</u> ) 追加( <u>A</u> ) 除去( <u>R</u> )
パッケージ(近): パッケージ(近): コンクロージング型(ゾ): 名前(M): 修飾子: スーパークラス( <u>S</u> ): インターフェース(j): 作成するメソッド・スタブの コメントを追加しますか?(;	JavaCOBOLMain            ・         ・         ・	<pre>     protected(]) </pre>	参照( <u>W</u> ) 参照( <u>W</u> ) 参照( <u>E</u> ) 追加( <u>A</u> ) 除去( <u>R</u> )
パッケージ( <u>K</u> ): 「エンクロージング型( <u>Y</u> ): 名前( <u>M</u> ): 修飾子: スーパークラス( <u>S</u> ): インターフェース( <u>i</u> ): 作成するメソッド・スタブの コメントを追加しますか?(;	JavaCOBOLMain            ・         ・         ・	) protected(])	参照( <u>W</u> ) 参照( <u>W</u> ) 参照( <u>E</u> ) 追加( <u>A</u> ) 除去( <u>R</u> )
パッケージ(近): パッケージ(近): コンクロージング型(ゾ): 名前(M): 修飾子: スーパークラス(S): インターフェース(j): 作成するメソッド・スタブの コメントを追加しますか?(:	JavaCOBOLMain            ・         ・         ・	O protected(])	参照( <u>W</u> ) 参照( <u>W</u> ) 参照( <u>E</u> ) 追加( <u>A</u> ) 除去( <u>R</u> )

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。



🔓 COBOL 🛛 🗙	₽ <mark>5.</mark> Navigat	- 🔁 Арр			
> ﷺ COBOLJav. > ﷺ COBOLJav. > ﷺ JavaCOBO	aProj1 aProj2 LProj1 プログラム				
> 🗁 bin	> 🦢 bin				
> 📂 New_Co	onfiguration.bin				
🗸 🗁 src					
🗸 🄁 com					
🗸 🧁 si	ample				
	JavaCOBOLMai	in.java			
	Program1.nativ	e_sig			
J	progs.java				

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)
	2	実行	テ点をリセット	
4	<b>%</b>	実行 デバ	〒(R) ッグ(D)	Ctrl+F11 F11
	0	実行 実行	亍履歴(T) テ(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) 以下の入力を行います。

名前: "JavaCOBOLMain"

メイン・クラス: "com.sample.JavaCOBOLMain"

名前(N): JavaCOBOLMain	
🕝 メイン ⋈= 引数 🛋 JRE 🍫 依存関係 🧤 ソ	ス 📧 環境
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
<b>9</b> メイン	🙀 🐴 JRE 🍫 依存関係 🧤 ソース 🦉
プログラ	ムの引数(A):
-VM 31	数(G):
-Djav	a.library.path=New_Configuration.bin

### 12) [実行(R)] をクリックします。

以下の結果がコンソールビューに表示されます。

COBOL 000000001 青	COBOL 0000
000002 黄	
COBOL 000000003 赤	
COBOL 000000004 緑	
Prime number from Java	
2	
3	
5	
7	
11	
13	
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:¥workspace-interoperability 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。



5)

- 3) [ファイル(F)] > [新規(N)] > [その他(o)] を選択します。
  - 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				]
<ul> <li>&gt; 22E</li> <li>&gt; 22E</li> <li>&gt; 23va</li> <li>24va プロジェクト</li> <li>25 Java ワーキング・セット</li> <li>ダ インターフェース</li> <li>ダ クラス</li> <li>ジ ソース・フォルダ</li> <li></li></ul>	ี่ขับว่า		^	
【● 列季型			Ý	]
? < 戻る(B) 次	^(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。			
プロジェクト名: "JavaCOBOLProj2」	//			
[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¥JavaCOBOLP	roj2J		参照	瘕(R)
JRE				
● 実行環境 JRE の使用(V):		JavaSE-17		~
○ プロジェクト固有の JRE を使用(S):		AdoptOpenJDK		
O Use default JRE 'AdoptOpenJDK' and workspace comp	iler preferences		JRE	を構成。
ブロジェクト・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ーを個別に作成(C)			<u>既定值</u>	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):			~ 選択	(E)
モジュール				
□ module-info.java を作成(M)				

パースペクティブの切り替えダイアログでは、[いいえ(N)]をクリックします。



CONTENT OF THE LAW BREETH - 143 DOUBLENT OUT :	Java パースペクティブを開きますか?	
□ #LC20092752B((0)       LULX(N)         JavaCOBOLProj2J JDジrJhが4Fr, Makitata         □ \$C080LavaProj1         > \$C080LavaProj2         \$C080LavaProj2 </td <td>このパースペクティブは、Java 開発をサポートするために設計 ラー、型階層、および Java 固有のナビゲーション・アクション・</td> <td>されています。 パッケージ・エクスプロー を提供します。</td>	このパースペクティブは、Java 開発をサポートするために設計 ラー、型階層、および Java 固有のナビゲーション・アクション・	されています。 パッケージ・エクスプロー を提供します。
IdvacCOBOLProj2J プロジェクトが作成されます。            GOBOL	□ 常にこの設定を使用する(R)	
JavaCOBOLProj2J プロジェクトが作成されます。            GOBOL ×          Navigat             GOBOLWavProj2             GOBOLWavProj2             JOSOBOLProj2J プロジェクトが表示されない場合)             Lクスプローラービューの右上をクリックし、[フィルタとカスタマイズ(F)] をクリックします。             COBOLWavProj2             LocoboLProj2J プロジェクトが表示されない場合)             Lクスプローラービューの右上をクリックし、[フィルタとカスタマイズ(F)] をクリックします。             COBOLWavProj             LocoboLProj2J             LocoboLProj2J         LocoboLProj2J             LocoboLProj2J             Locobo	パースペクティブを	2開く(O) いいえ(N)
COBOL × × Navigat     COBOLIvaProj2     COBOLIvaProj2     JovaCOBOLProj1     JovaCOBOLProj2     JovaCOBOLProj2  JAVACOBOLProj2  JAVA	JavaCOBOLProj2J プロジェクトが作成されます。	
<ul> <li></li></ul>	පි¦ COBOL × ඎ Navigat	
> と JavaCOBOLProj2) JavaCOBOLProj2) JavaCOBOLProj2) JavaCOBOLProj2) JavaCOBOLProj2) JavaCOBOLProj2) JavaCOBOLProj2) JD27D-ラービューの右上をクリックし、[フィルタとカスタマイズ(F)]をクリックします。  * COBOLANAProj2 COBOLANAPRO	> 🚰 COBOLJavaProj1 > 🛃 COBOLJavaProj2	
JavaCOBOLProj2J プロジェクトが表示されない場合) エクスプローラービューの右上をクリックし、[フィルタとカスタマイズ(F)] をクリックします。 <sup>©</sup> COBOLMAPOI <sup>●</sup> COBO	> 🔁 JavaCOBOLProj2J	
エクスプローラービューの右上をかりいクし、[フイルタとカスタマイズ(F)]をクリックします。         © C080L × © Naviget ② Applicat ③ ワーバー ④ Analysia ③ (C080LNM TD)207+表示(R) )         > ③ C080LawaProj         > ④ C080LawaProj         > ③ C080LawaProj         > ③ C080LawaProj         > ④ C080LawaProj         > ⑦ C080LawaProj         ● C080LawaProj	JavaCOBOLProj2J プロジェクトが表示されない場合	·)
© COBOL → X © Navigati	エクスプローラービューの右上をクリックし、[フィルタとカスタ	マイズ(F)]をクリックします。
<ul> <li></li></ul>	ີ 😪 COBOL 🗙 🗞 Navigat 🤮 Applicat 📑 サ−/ໃ− 💻 Analysis	
<ul> <li>○ (2000/sephoj2)</li> <li>○ (2000/sephoj2)<td>&gt; ∰r COBOLJavaProi1</td><td></td></li></ul>	> ∰r COBOLJavaProi1	
	> COBOLJavaProj2	COBOLISM フロシェクト表示(K) >
- キソク・セック選択解除(K) アク・マガク・モック セック・セット デ フルタン ウ・マング・セック選集(E) デ フルタン カマイズ(P) 「非 Micro Focus プロジェクト]のチェックを外したうえで、[OK]をクリックします。 「 プリセット・フィルター ア ユーザー・フィルター を コンテンツ 適用するフィルターを選択してください(一致する項目は得されます): グ カデゴリ外の空のフォルダ グ 内部 Micro Focus プロジェクト ク 内部 TD プロジェクト 会 成メンバ 空 空のカテゴリ 空 空のディジ 空 空の親パッケージ 空 空の親パッケージ 空 空の親パッケージ テ アレックのたっていま プロジェクト トーマックト ・ ポMicro Focus プロジェクト ・ ポMicro Focus プロジェクト ・ アレーレーンテキスト マークを開き 「竹	> 🔄 JavaCOBOLProj1	ワーキング・セットの選択(W)
P3r/30-19/20 せかの頃集(E) 19(2)ドゥ・キング・セット 19(2)ドゥ・キング・セット アメルタとカスタマイスの)		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・セットの編集(E)
マリセット・フィルターマローザー・フィルターマローザー・フィルターマロージャンス くく、[OK] セクリックリンス さく、[OK] セクリックレンス さく、[OK] セクリンス さく、[OK] セクリンス さく、[OK] セクリックレンス さく、[OK] セクリンス うく、[OK] セクリンス さく、[OK] セクリンス さく、[OK] セクリンス うく、[OK] セクリンス うく、[OK] セクリンス うく、[OK] ロンス うく、[OK] セクリンス うく、[OK] ロンス うく、		
アリセット・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):         (*)         (*)         カテゴリ外の空のフォルダ         (*)		
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):          ② カテゴリ外の空のフォルダ       ▲         ③ 内部 Micro Focus プロジェクト       ▲         ④ 内部 TD プロジェクト       ▲         ○ 含成メンパ       ② 空のカテゴリ         ② 空の別「ッケージ       ② 空の親パッケージ         ○ 空の親パッケージ       ●         ● 推動 COBOL プログラム       ■         ■ 閉じたプロジェクト       ●         ● 非 Micro Focus プロジェクト       ●         ● 水口 「       ●         ● ロレビア ●       ●         ●       ●         ●       ●         ●       ●         ●       ●         ●       ●         ●       ●         ●       ●         ●       ●         ●       ●	ア ブリセット・フィルター ア ユーザー・フィルター 2 コンテンツ	
○ カテゴリ外の空のフォルダ ○ 内部 Micro Focus ブロジェクト ○ 内部 D プロジェクト ○ 含成メンパ ○ 空のカテゴリ ○ 空のパッケージ ○ 空の親パッケージ ○ 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 メンパ( ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・	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	·):
<ul> <li>○ カテゴリ外の空のフォルダ</li> <li>○ 内部 Micro Focus ブロジェクト</li> <li>○ 内部 TD プロジェクト</li> <li>○ 合成メンパ</li> <li>○ 空のカテゴリ</li> <li>○ 空の親パッケージ</li> <li>○ 空の親パッケージ</li> <li>○ 空の親パッケージ</li> <li>○ 建の親パッケージ</li> <li>○ 推承 COBOL プログラム</li> <li>□ 閉じたプロジェクト</li> <li>□ 非 Micro Focus プロジェクト</li> <li>□ 非 public メンパ(</li> </ul>		
<ul> <li>○ 内部 Micro Focus プロジェクト</li> <li>○ 内部 TD プロジェクト</li> <li>○ 合成メンパ</li> <li>○ 空のカテゴリ</li> <li>○ 空の親パッケージ</li> <li>○ 空の親パッケージ</li> <li>○ 登の親パッケージ</li> <li>○ 推示 COBOL プログラム</li> <li>□ 閉じたプロジェクト</li> <li>□ 非 Micro Focus プロジェクト</li> <li>□ 非 public メンパ</li> </ul> OK キャンセル avaCOBOL Proi21 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、「プロジェクト」	✓ カテゴリ外の空のフォルダ	^
<ul> <li>○ 内部 TD プロジェクト</li> <li>○ 含成メンバ</li> <li>○ 空のガテゴリ</li> <li>○ 空のパッケージ</li> <li>○ 空の親パッケージ</li> <li>○ 空の親小ッケージ</li> <li>○ 空の親小ッケージ</li> <li>○ 空の親小ッケージ</li> <li>○ (○ (○ (○ (○ (○ (○ (○ (○ (○ (○ (○ (○ (○</li></ul>	│	
<ul> <li>✓ 合成メンバ</li> <li>✓ 空のカテゴリ</li> <li>□ 空のパッケージ</li> <li>✓ 空の親パッケージ</li> <li>✓ 空の親パッケージ</li> <li>✓ 建取るCOBOL プログラム</li> <li>□ 閉じたプロジェクト</li> <li>□ 非 Micro Focus プロジェクト</li> <li>□ 非 public メンバ</li> </ul>	☑ 内部 TD プロジェクト	
<ul> <li>② 空のカテゴリ</li> <li>□ 空のパッケージ</li> <li>② 空の親パッケージ</li> <li>② 塗の親パッケージ</li> <li>○ 継承 COBOL プログラム</li> <li>□ 閉じたプロジェクト</li> <li>□ 非 Micro Focus プロジェクト</li> <li>□ 非 public メンパ</li> </ul> OK キャンセル avaCOBOL Proi21 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問き 「プロジェクト」	□ 合成メンバ	
□ 空のパッケージ □ 空の親パッケージ □ 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 メンパ ・ × OK キャンセル avaCOBOL Proi21 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問き 「プ	□ 空のカテゴリ	
	□ 空のパッケージ	
○ 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 ○ K キャンセル avaCOBOL Proi21 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、「プロジェクトを発行したうえて、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、「プロジェクト」	──	
□ 閉じたプロジェクト □ 非 Micro Focus プロジェクト □ 非 public メンバ OK キャンセル avaCOBOL Proi21 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問き 「プ		
□ 非 Micro Focus プロジェクト □ 非 public メンパ OK キャンセル avaCOBOL Proi 21 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	── 閉じたプロジェクト	
□ 非 public メンバ OK キャンセル avaCOBOL Proi21 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問き 「プロ	□ 非 Micro Focus プロジェクト	
OK キャンセル avaCOBOLProi21 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問き、「プロ	□ 非 public メンバ	×
OK キャンセル avaCOBOLProi21 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問き 「プロ		
OK キャンセル avaCOBOLProi21 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問き 「プロ		
OK キャンセル avaCOBOLProi21 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問き、「プロ		
avaCOBOLProi21 プロジェクトを選択したうえで マウスの右クリックに上りコンテキストメニューを思き 「プロ		
		OK キャンセル

(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。

フィルタ入力	Java のビルド・パス				
> リソース Coverage	(歩 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) 【		
Javadoc ロケーション > Javaエディタ	Build class path order and exported (Exported entries are contributed	ed entries: to dependent proje	ects)		
> Java コード・スタイル > Java コンパイラー Java のビルド・パス	□ ■ JRE システム・ライブラリー [Jav ■ 通 JavaCOBOLProj2J/src	vaSE-17]			
[クラスパス] を選択したうえ	で、[ライブラリーを追加(i)] を	ミクリックします。			
Java のビルド・パス			↓ ↓ ↓ 8		
(伊) ソース(S) (学) プロジェクト(P) ビルド・パス上の JAR およびクラス・フォル	▲ ライブラリー(L) 😽 順序およびエクスポート(C ダー(T):	)) 😡 モジュール依存	関係(M)		
✓ <sup>0</sup> √ モジュールパス	05.401		JAR の追加(J)		
<ul> <li> <ul> <li></li></ul></li></ul>	waSE-1/]		外部 JAR の追加(X)		
			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)					
[COBOL JVM 実行時シ	ステム] を選択し、[次へ(N)]	をクリックします	- o		
<b>ライブラリーの追加</b> 追加するライブラリー・タイプを選択	します。	å			
COBOL JVM 実行時システム					

COBOL JVM 実行時システム			
CXF ランダイム EAR ライブラリー JRE システム・ライブラリー JUnit Maven Managed Dependencie Web App ライブラリー サーバー・ランタイム 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義	es		
? < 戻	る(B) 次へ <b>(N) &gt;</b>	終了(F)	キャンセル

そのまま、[終了(F)]をクリックします。



## COBOL JVM 実行時システム

ロケーション: C:¥Prog	ram Files (x86)¥Mio	ro Focus¥Visual CC	BOL	
?	< 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	← → ⇒ 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(O) 🥥 モジュール依存	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)
	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	
Ċ	サンプル(X)	
Ľ	その他(o)	Ctrl+N
<b>F1</b> -1		

[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):		
フィルタ入力		
> 🔁 Gradle		~
> 🤁 J2EE		
🗸 🗁 Java		
😕 Java プロジェ	クト 	
/ Java ワーキン	グ・セット	
	2	
G 77X	a	
■ パッケージ		
※ 既存 Ant ビ	レド・ファイルからの Java プロジェクト	
🚯 記録		
● 注釈		~
?	< 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。	
11° / 22 //		
バッケージ:"com.	sample"	
名前: "Java	COBOLMain"	
Iava h=1		
新規 Java クラスを1F成し	£9°	
N=7・7+Ⅲ.ダ(D),		<b>券</b> 昭(α)
/ /////////////////////////////////////	Javacobolerioj2/sic	- <u>≫,</u> ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
バッケージ(K):	com.sample	参照(W)
□ エンクロージング型(Y):		参照(W)
名前(M):	JavaCOBOLMain	
修飾子:	public(P)      package(C)      private(V)      protected(T)	
	abstract(T) final(L) static(C)	
	none     O sealed     O non-sealed     O final(L)	
スーパークラス(S):	iava.lang.Object	参昭(E)
11/2-7t-7(i)	J	
177 71 X(I)-		ぇ旦刀川(A)
		P☆ ± (D)
		际云(K)
作成するメソッド・スタブの追	、」 瞿択	
	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ここを参照)	
	コメントの生成(G)	
(?)	< 更ろ(B) ジカム(ND 、 約7/D)	キャンカル

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。





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(L)]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"



設定	値
▼ 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"

フィルタ入力	リンク	← < <>
> リソース Coverage	New Configuration [使用中]	✓ 構成の管理
✓ Micro Focus ビルダー ビルドパス ✓ ビルド構成	7111/97キストを入力	
> COBOL	設定	値
1425	✓ Linkage	
71/01	出力の名前	JavaCOBOLProj2C
見 115-25	出力パス	New Configuration.bin
> 929	エントリポイント	
COPOL	ターゲットの種類	単一 ネイティブライブラリ ファイル
> COBOL	17 1 44	

- 11) [適用して閉じる] をクリックします。
- 12) 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。



> 🖂 JavaCOBOLProi2	2C				public static void max
> 🛃 JavaCOBOLProj2	2	新規作成(N)	>	2	COBOL JVM プロジェクト
		表示方法(W)	Alt+シフト+W >	얥	COBOL JVM ユニット テスト プロジェクト
		JK-	Ctrl+C	2	COBOL コピーファイル プロジェクト
	ß	<u>-</u> 貼り付け	Ctrl+V		
	×	削除(D)	削除	100 100	COBOL ユニット ナスト フロシェクト COBOL Java 相互運用機能のプロジェクト
		移動(V)			UT-ト COBOL JVM 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	>	193	リモート COBOL ユニット テスト プロジェクト
		インポート(i)	>	<b></b>	プロジェクト(R)
	4	エクスポート(O)		RŶ.	COBOL コピーファイル
	জ	更新(F)	F5	đ	COBOL プログラム
そのまま、[終了	7(F	)]をクリックします。			
	<b>1</b> =	,			
COBOL M	<i>ו</i> כו	4			
エディタで開くこと	とがて	?きる COBOL プログラムを	新規作成します	•	
A + to スプロン/	-				* 177
含まれるノロシエ	214	JavaCOBOLProj2C			参照…
新担ファイルタ・		Program1 cbl			
オロパレノア・ロル・ロー		Flogramicol			
テンプレートを選	択:				
E Micro	Foc	い テンプレート			
E Micro	100	us / / / /			
					テンプレートの設定を進成
					ノファレートの認定で構成。
□ テンプレート	<b>の</b> 参	\$照			
LE SC					20.077
P\$6 PT:					110 月末
ファイル	ल्यत्र	テムを選択: default 〜			
27 137					
				F	
Q				L	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 🛛 🗙	🔁 Navigat	Provide Applicat
> 🛃 COBOLJava	Proj1	
> 📂 COBOLJava	Proj2	
> 🛃 JavaCOBOL	Proj1	
🗸 📂 JavaCOBOL	Proj2C	
> 🔁 COBOL 🕽	プログラム	
> 🗁 New_Cor	nfiguration.bin	
🗸 📂 JavaCOBOL	Proj2J	
> 🗁 bin		
🗸 🗁 src		
🗸 🥟 com		
🗸 🗁 sai	mple	
J	JavaCOBOLMai	n.java
	Program1.nativ	e_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、



13) 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

プロジェクト( <u>P</u> )	実行	( <u>R)</u> ウィンドウ( <u>W</u> ) ヘルフ	f( <u>H</u> )	
1 <b></b>	R	実行点をリセット		> -   📑
🛄 Analysis	Q,	実行(R)	Ctrl+F11	
× 🗉 🕏	椮	デバッグ(D)	F11	
		実行履歴(T)	>	····• ···· · 4 · · · · • · · · · 5 · · · ·
	0	実行(S)	>	le)
		実行構成(N)		"Programl".
		デバッグ履歴(H)	>	•
	*	デバッグ(G)	>	
		デバッグの構成(B)		curs 10 value 2,3,5,7,11
	Q,	項目を検査		
	0	プログラム ブレークポイントを	追加	
		ツール	>	urs 4. 5.
	<b>9</b>	外部ツール(E)	>	(起動履歴なし)
		Θ	procedure division usi	==(−(□) 、
		⊜	perform varying i	外部ツ−ルの構成(E)

成(E)] を選択します。

© Rocket Software, Inc. or its affiliates 1990–2024. All rights reserved. Rocket and the Rocket Software logos are registered trademarks of Rocket Software, Inc. Other product and service names might be trademarks of Rocket Software or its affiliates.

## **Rocket** software

14) [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	
■ Xイン  参 更新	
U7 – 9 H J(L): C:¥Program Files (x86)¥Micro Eocus¥Visual COBOL¥hin64¥geniava exe	
Carriogram nies (xoo)riviteto rocustvisuar coboettomotrigerijava.exe	
	リークスペースの参照(P) リパール・システムの参照(E) 支数(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	^
	<b>~</b>
	変数(S)
注・スペースを含む引数は二重引用符(**)で用んでください。	20-20-14-1 m
	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。



😤 COBOL 🗙 🔁 Navigat 🧏 Applicat			
<ul> <li>&gt; <sup>™</sup><sub>2</sub> COBOLJavaProj1</li> <li>&gt; <sup>™</sup><sub>2</sub> COBOLJavaProj2</li> <li>&gt; <sup>™</sup><sub>2</sub> JavaCOBOLProj1</li> <li>✓ <sup>™</sup><sub>2</sub> JavaCOBOLProj2C</li> <li>&gt; <sup>™</sup><sub>2</sub> COPOL <sup>™</sup><sub>2</sub> III<sup>™</sup><sub>2</sub> I</li> </ul>			
> 🗁 New_Configuration.bin			
V 🔁 JavaCOBOLProj2J			
V 🗁 src			
🗸 🗁 com			
🗸 🗁 sample			
🚺 JavaCOBOLMain.java			
Program1.native_sig			
progs.java			

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

	実行( <u>R)</u> ウィンドウ( <u>W</u> ) ヘルプ( <u>H</u> )		ウィンドウ( <u>W</u> )	ヘルプ( <u>H</u> )
-	◎ 実行点をリセット			
;	💫 実行(R) Ctrl+F11			
>	椮	, デバッグ(D) F11		F11
		実行	亍履歴(T)	>
	0	実行	<del>T</del> (S)	>
		実行	亍構成(N)	

- [Java アプリケーション] をダブルクリックし、以下の入力を行います。
- 名前: "JavaCOBOLProj2J"

メイン クラス: "con.sample.JavaCOBOLMain"

名前(N): JavaCOBOLProj2J	
③ メイン ⋈= 引数 副 JRE 🍫 依存関係 🧤 ソース 🚾 環境 🔲 共通(C) 🖻 プロトタイプ	
ブロジェクト(P):	
JavaCOBOLPTOJZJ	参照(B)
com.sample.JavaCOBOLMain	検索(S)
ー	
□ 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): JavaCOBC	OLProj2J	
G メイン (∞)= 引数	x 🔥 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(A</mark>	A):	^
	×	
	変数(1)	
VM 引数(G):		
-Djava.library.pa	ath=¥JavaCOBOLProj2C¥New_Configuration.bin	
	× 1	
	変数(S)	
Use the -XX:+	+ShowCodeDetailsInExceptionMessages argument when launching	
<ul> <li>記動時に@arg</li> </ul>	rgfile 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)	~
	コマンド行を表示(W) 前回保管した状態に戻す(V) 適用(Y)	

16) [実行(R)] をクリックします。

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 0000
000002 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.microfocus.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]

# **Rocket** software

## 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。

	ウイン	'ドウ(W)	ヘルプ(H)	
3		新規ウィ	ンドウ(N)	
		エディタ-		>
6		外観		>
1		ビューの表	長示(V)	>
		パースペ	クティブ(R)	>
		ナビゲージ	7∃ン(G)	>
		Spies	設定	>
		設定(P)		

2) [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	ワークスぺース ◇ ▼ ᢤ
✓ 一般 へ 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 Associi > Security UI フリーズ・モニタ > User Storage Se	□ 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) ☑ アクセス時に更新(S) □ 無関係なプロジェクトを常にプロンプトなしで閉じる(C)
Web ブラウザ	ワークスペース保管間隔 (分)(W): 5
> エディタ キー クイック検索 グローバル化 コンテンツ・タイプ サービス・ポリシー トレース	ウィンドウのタイトル ✓ ワークスペース名を表示(E): workspace-interoperability □ パースペクティブ名を表示(T) □ ワークスペースのフルパスを表示(F): C:¥workspace-interoperability ✓ プロダクト名を表示
<ul> <li>ネットワーク接続 ハンドラーをリンク パースペクティブ ゴロジェクト・ナー。</li> <li>ワークスペース</li> <li>開始わよびシャッ</li> <li>外観</li> </ul>	プロジェクトを開く際に、参照するプロジェクトを開く: 不明なプロジェクトの性質を以下のように報告(A): 客告 Report missing project encoding as: 警告
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}
上較パパッチ > Ant AspectJ Compiler > CSS (Wild Web De	<ul> <li>テキスト・ファイル・エンコード(T)</li> <li>新規テキスト・ファイルの行区切り文字(F)</li> <li>デフォルト(U) (windows-31j)</li> <li>○その他(O): windows-31 </li> <li>○その他(H): Windows </li> </ul>
> HTML (Wild Web I *	デフォルトの復元(T) 適用(L)
? <b>è 4</b>	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。

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

 [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。

COBOL ← → ⇒ 8 フィルタ入力 > リソース Coverage Javadoc ロケーション フィルタテキストを入力 (= > Javaエディタ > Java コード・スタイル 設定 値 > Java コンパイラ-~ ─般 Java のビルド・パス 文字セット ✓ Micro Focus ソース エンコーディング ANSI ビルダー COBOL 方言 Micro H ビルドパス > ビルド構成 ソース フォーマット 固定 ブロジェクト設定
 COBOL
 コンテナー 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 ビルド環境 いいえ ∨ 出力 指令の確定 > 実行時構成 指令ファイルを生成する いいえ リストファイルを生成 サーバー タスク・タグ いいえ 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 プロジェクト・ファセット Java Interoperability 出力パス src プロジェクト参照 パッケージ名 com.microfocus.COBOL ✓ Iラ-/警告 > 検証 警告レベル 回復可能なエラーを含める(レベル E) 実行/デバッグ設定 最大エラー数 100 ソースエンコーディング SOURCE-ENCODING はソース ブ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 ない 指金 外援等されたいない、限り、室穴時のエンコーディングの決定に使用されます。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

ソース エンコーディング: "ANSI"



### 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。

© Rocket Software, Inc. or its affiliates 1990–2024. All rights reserved. Rocket and the Rocket Software logos are registered trademarks of Rocket Software, Inc. Other product and service names might be trademarks of Rocket Software or its affiliates.