Visual COBOL チュートリアル

Apache Tomcat と JVM COBOL コンポーネントを利用した RESTful Web サービス開発

1　目的

Visual COBOL は、他言語・他システムとの連携手段として、「Enterprise Server」を利用したサービス連携だけではなく、COBOL プログラムをそのまま利用し、Java 技術と連携可能な JVM COBOL 機能を提供しています。本機能を利用することで、Java, Scala などの Java 言語で作成した RESTful Web サービス上で COBOL を利用するといった方法が可能です。

このドキュメントでは JVM COBOL 機能を利用して Java で実装する RESTful Web サービスと COBOL の連携方法について説明します。

2　前提条件

本チュートリアルは、下記環境を前提に作成されています。

OS	Windows 11 Enterprise Edition
COBOL 環境製品	Visual COBOL 9.0J for Eclipse Patch Update 2
Java アプリケーションサーバー	Apache Tomcat 10.0.27

※ Visual COBOL 製品は Apache Tomcat 10.0 の EOS にともない、10.1 をサポートしています。しかし、製品で使用する Eclipse IDE では 10.0 のみ選択できます。これは、Eclipse IDE 側のプラグインの挙動によります。このチュートリアルで使用する 10.0.27 は製品機能紹介目的であり、運用環境においては 10.1 の使用を推奨します。

Java 言語で作成する RESTful Web サービスは JAX-RS という Web サービスを構築するための Java API を使用しています。本ドキュメントでは、Tomcat 10 を Java アプリケーションサーバーとして使用して動作確認を行いますが、Apache Tomcat に依存せず標準的な API 仕様に基づいたサンプルアプリケーションとなります。このため、本ドキュメントで紹介するようなサービスを、他の Java アプリケーションサーバー上にて開発・利用することも可能です。

また、Eclipse の Tomcat プラグインに関する設定および動作などにつきましては、弊社製品外であり、インターネットなどの各種文献をご参照ください。

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

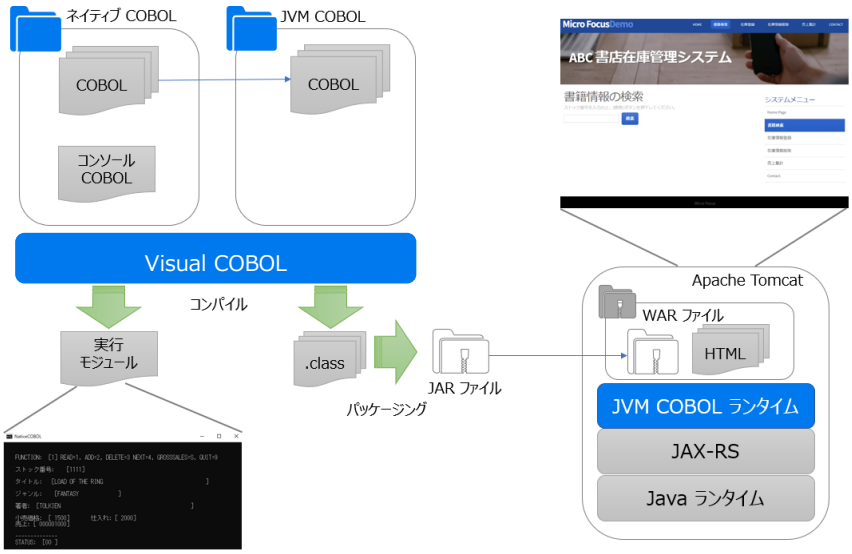
このサンプルプログラムは、COBOL で作成された簡単な書籍情報を管理するアプリケーションであり、索引ファイルを利用しています。以降の手順では、C:¥jvmRESTfulWSTutorial を解凍先のフォルダとして説明しています。

チュートリアル用のサンプルファイルのダウンロード

内容

3　チュートリアル手順の概要

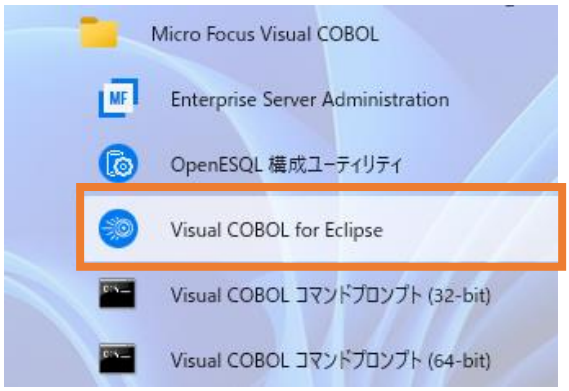
従来型の書籍情報を管理するコンソールアプリケーションを、JVM COBOL の機能を利用して COBOL プログラムそのまま流用し、Java クラスを生成します。そして、このクラスを利用して Java ベースの RESTful Web アプリケーションを構築します。最後に、Web アプリケーションを Apache Tomcat アプリケーションサーバーにデプロイし、実際の動作を確認します。



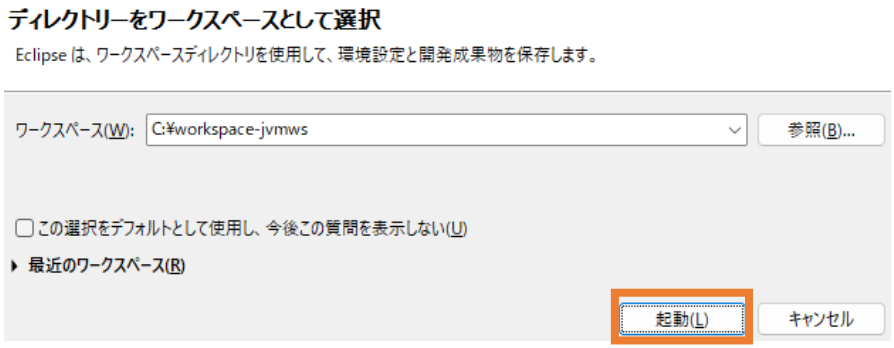
3.1　プロジェクトの作成と COBOL アプリケーションの確認

3.1.1 プロジェクトの作成

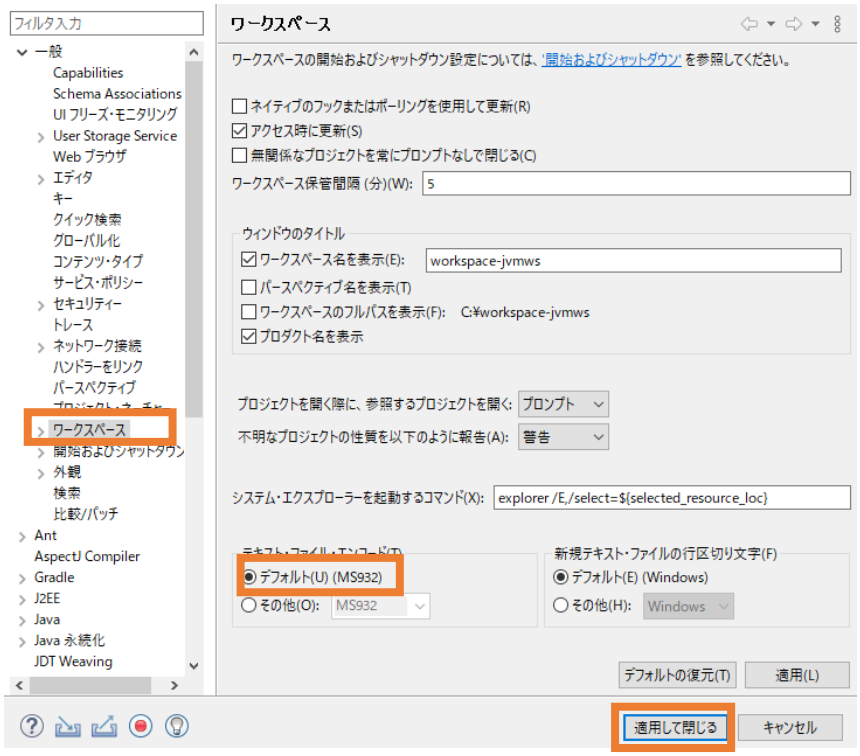
1)　Visual COBOL for Eclipse を起動します。



2)　任意のワークスペースを選択し、[起動(L)] をクリックします。

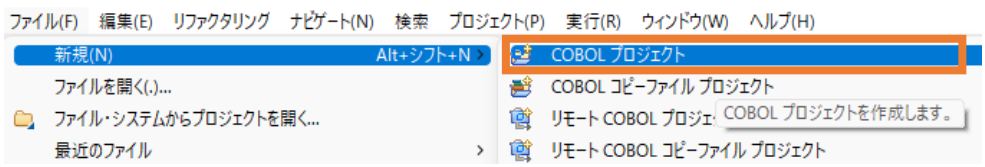


3) Visual COBOL 9.0 より Shift-JIS を指定して日本語を表示する場合、文字コードの指定を明確に行う必要があります。[Window] > [設定] より [一般] > [ワークスペース] とナビゲートし、テキストファイルエンコードを「MS932」に変更し、[適用して閉じる] をクリックします。



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

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

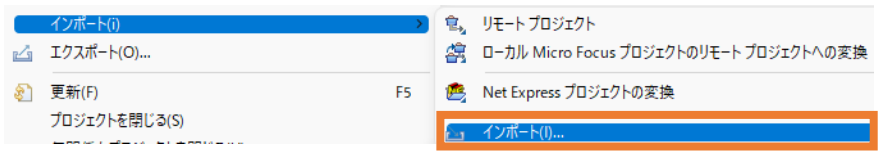


5) 以下の入力を行い、[終了(F)] をクリックします。

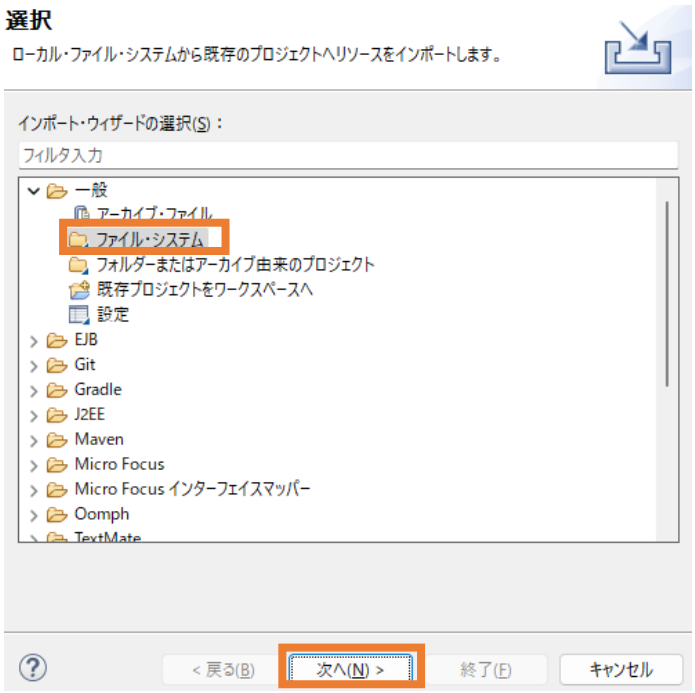
プロジェクト名： “NativeCOBOL”

プロジェクトテンプレート： 32 ビット、64 ビット、いずれを選択しても構いません

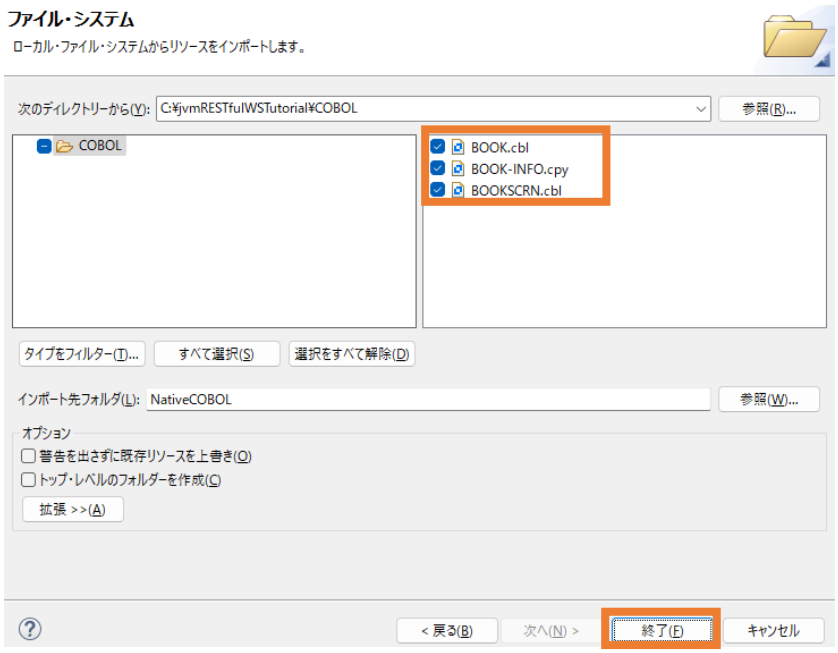
6) NativeCOBOL プロジェクトを選択し、マウスの右クリックにてコンテクストメニューを開き、[インポート(I)] > [インポート(I)] を選択します。



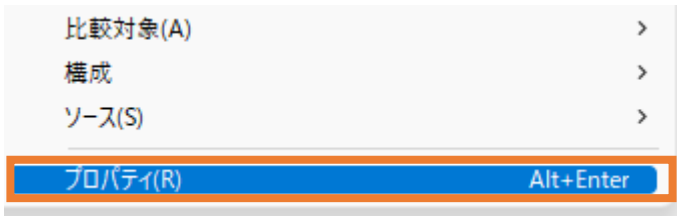
7) 一般フォルダ配下の [ファイル・システム] を選択して、[次へ(N)] をクリックします。



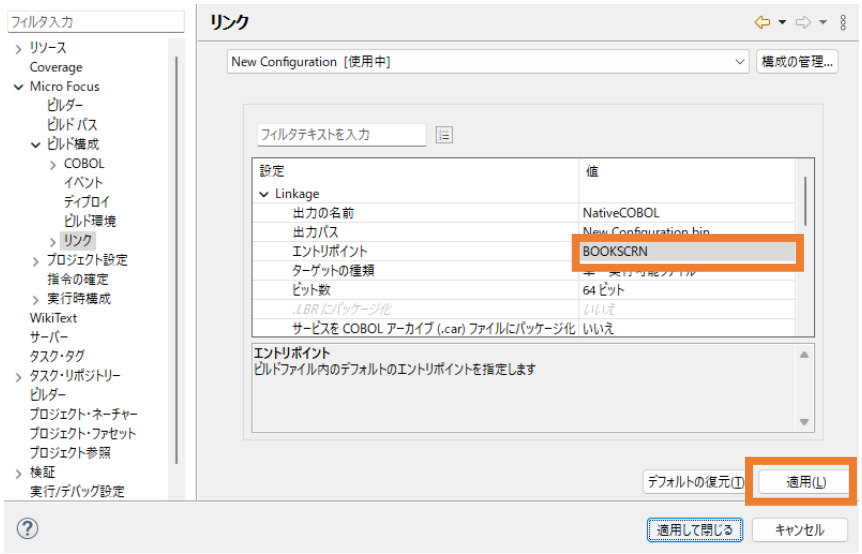
8) C:¥jvmRESTfulWSTutorial¥COBOL 配下の 3 つのファイルを選択し、[終了(F)] をクリックします。



9) NativeCOBOL プロジェクトを選択し、マウスの右クリックにてコンテクストメニューを表示した上で、[プロパティ(R)] を選択します。



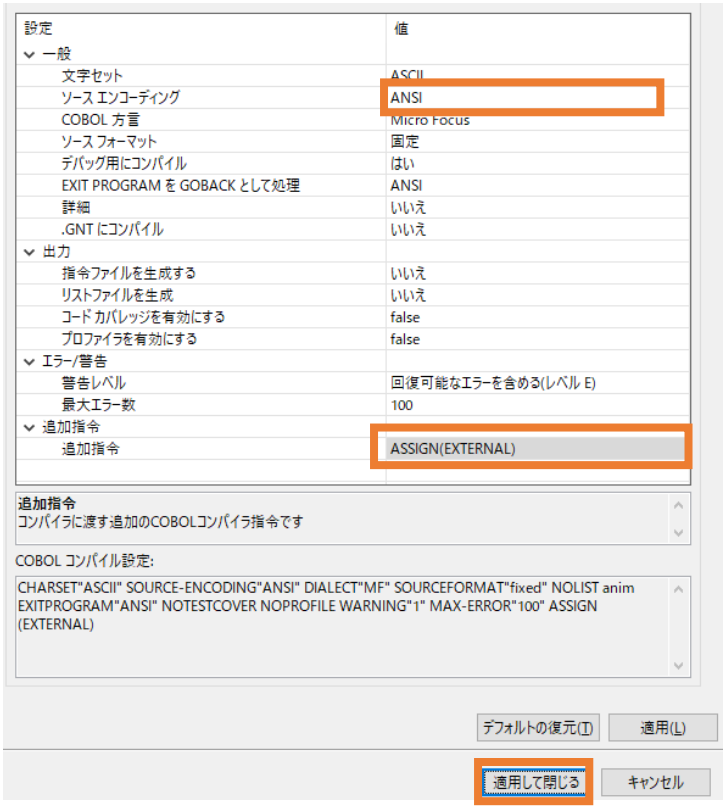
10) [Micro Focus] > [ビルド構成] > [リンク] を選択し、「エントリポイント」に “BOOKSCRN” を入力したうえで、[適用(L)] をクリックします。



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

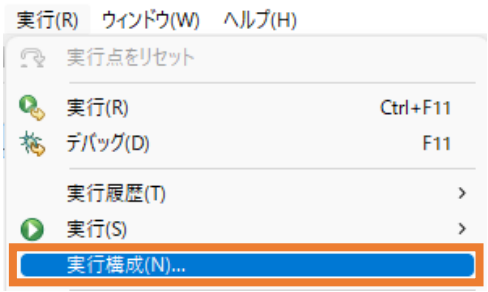
[ソースエンコーディング] : “ANSI”

[追加指令] : ”ASSIGN(EXTERNAL)”

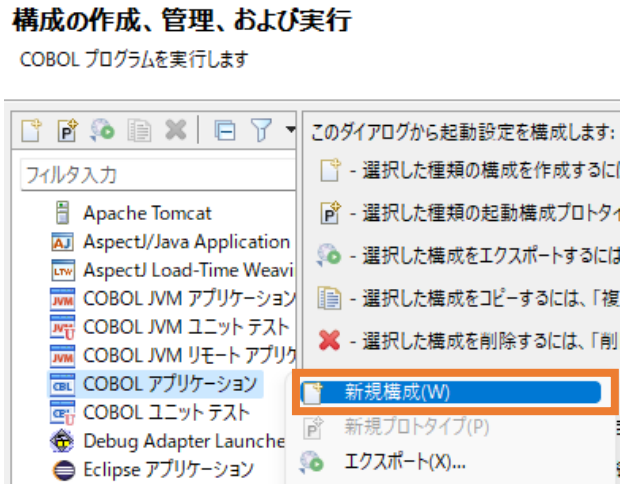


3.1.2 アプリケーションの確認

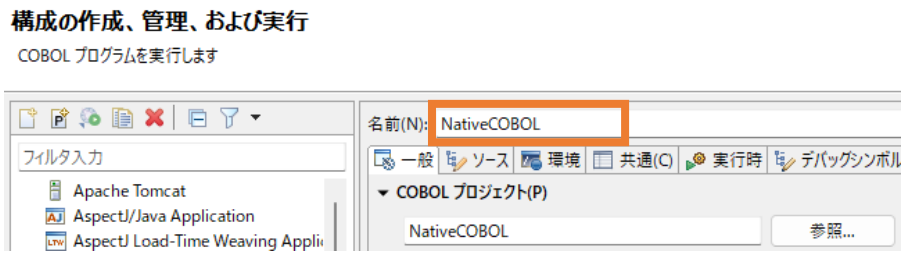
1) NativeCOBOL プロジェクトを選択し、[実行(R)] > [実行構成(N)] を選択します。



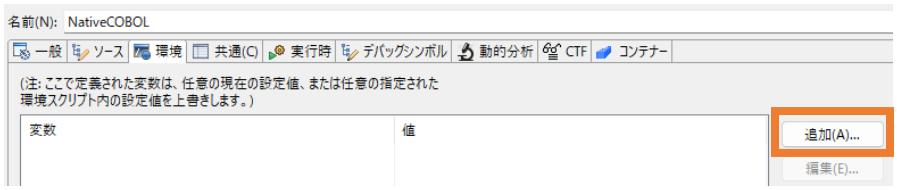
2) COBOL アプリケーションを選択し、マウスの右クリックにてコンテクストメニューを表示した上で、[新規構成(W)] をクリックします。



3) 「名前」に “NativeCOBOL” を入力します。



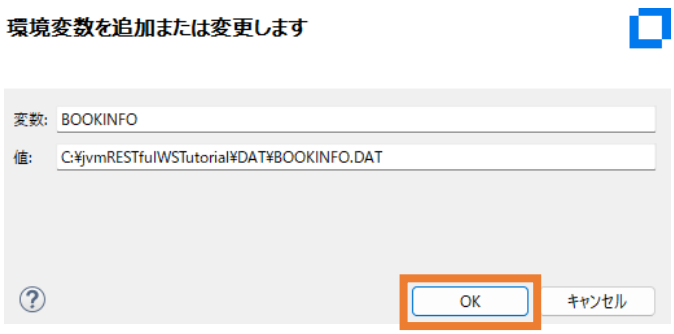
4) [環境]タブを選択し、[追加(A)] をクリックします。



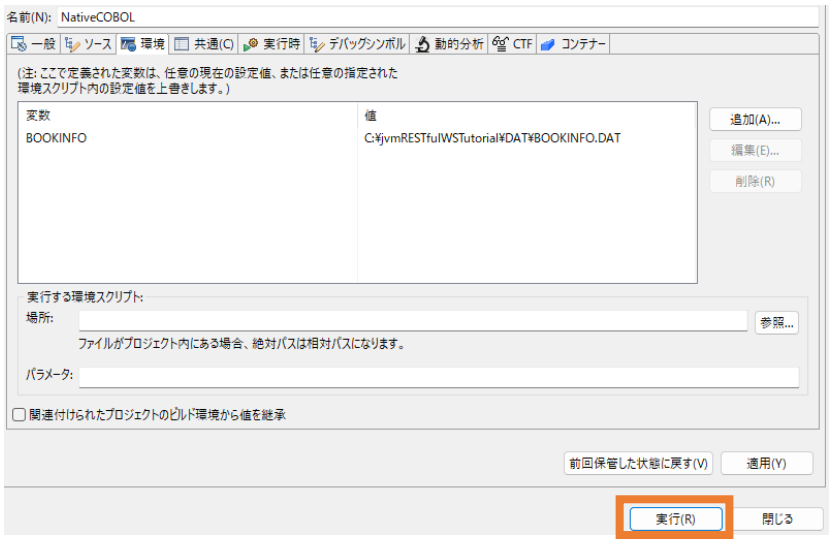
5) 以下の入力を行い、[OK] をクリックします。

変数： “BOOKINFO”

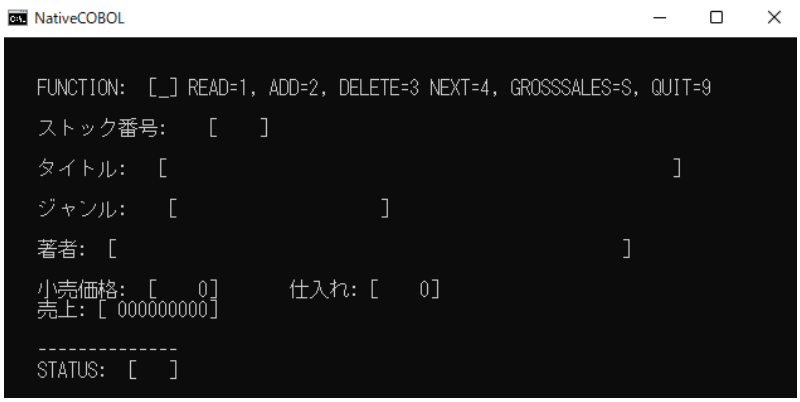
値： “C:¥jvmRESTfulWSTutorial¥DAT¥BOOKINFO.DAT”



6) BOOKINFO 変数が追加されたことを確認して、[実行(R)] をクリックします。



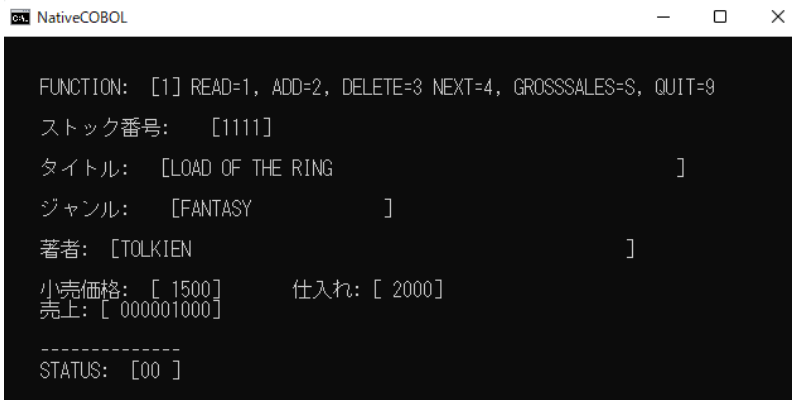
以下のような画面が表示されます。



以下の入力を行った後、Enter キーを押すと、該当書籍情報が表示されます。

FUNCTION : “1”

ストック番号 : “1111”



現在、ストック番号：4444 は未登録のため、以下の入力を行い、Enter キーをクリックします。

FUNCTION: “2”

ストック番号： “4444”

タイトル： “鏡の国のアリス”

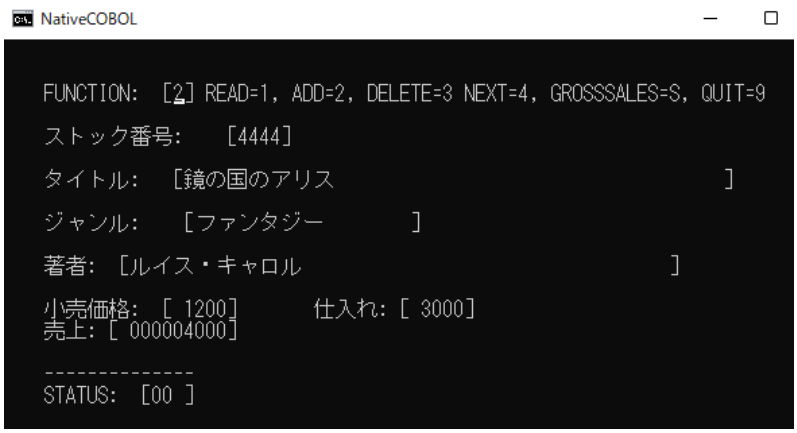
ジャンル： “ファンタジー”

著者： “ルイス・キャロル”

小売価格： “1200”

仕入れ： “3000”

売上： “4000”



実行後、以下の入力を行い、書籍情報が登録されていることを確認してください。

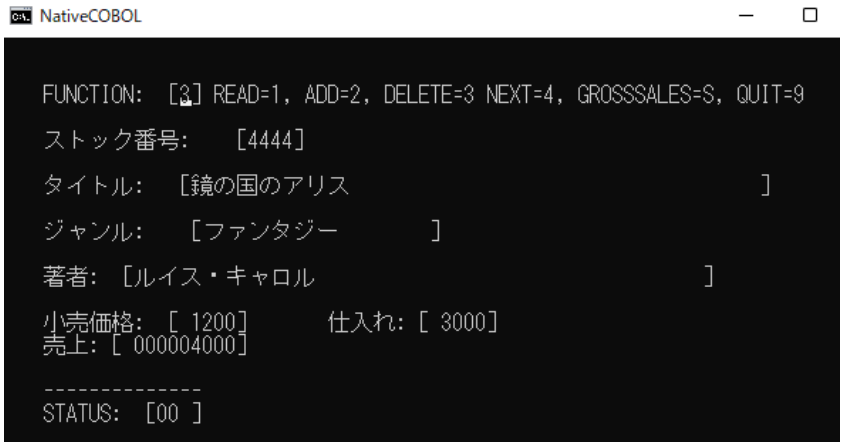
FUNCTION: “1”

ストック番号： “4444”

確認後、以下の入力を行い、Enter キーを押すことで登録した書籍情報が削除されます。

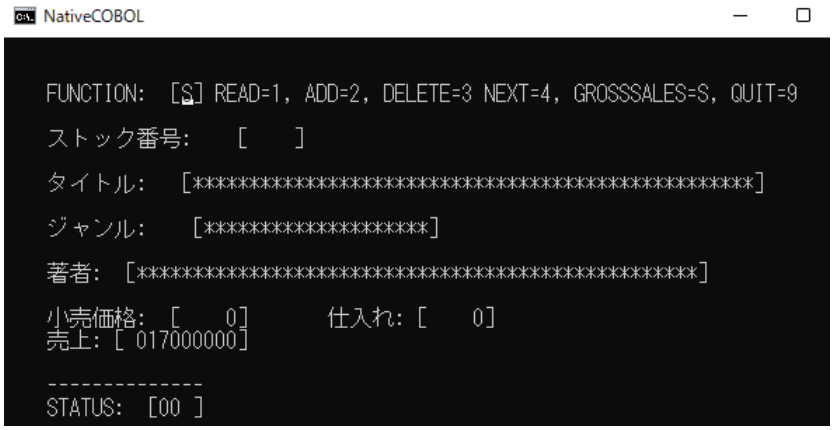
FUNCTION: “3”

ストック番号： “4444”



実行後、READ 機能でストック番号： 4444 が削除されていることを確認してください。

FUNCTION=S の GROSSSALES 機能は登録された全書籍情報の売上額を集計します。

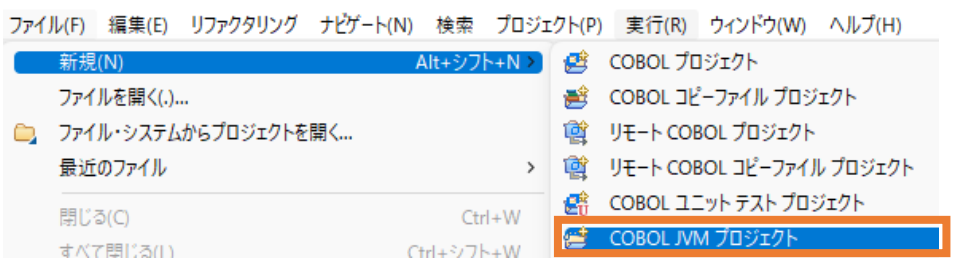


確認後、「FUNCTION」に “9” を入力し、Enter キーを押してプログラムを終了します。

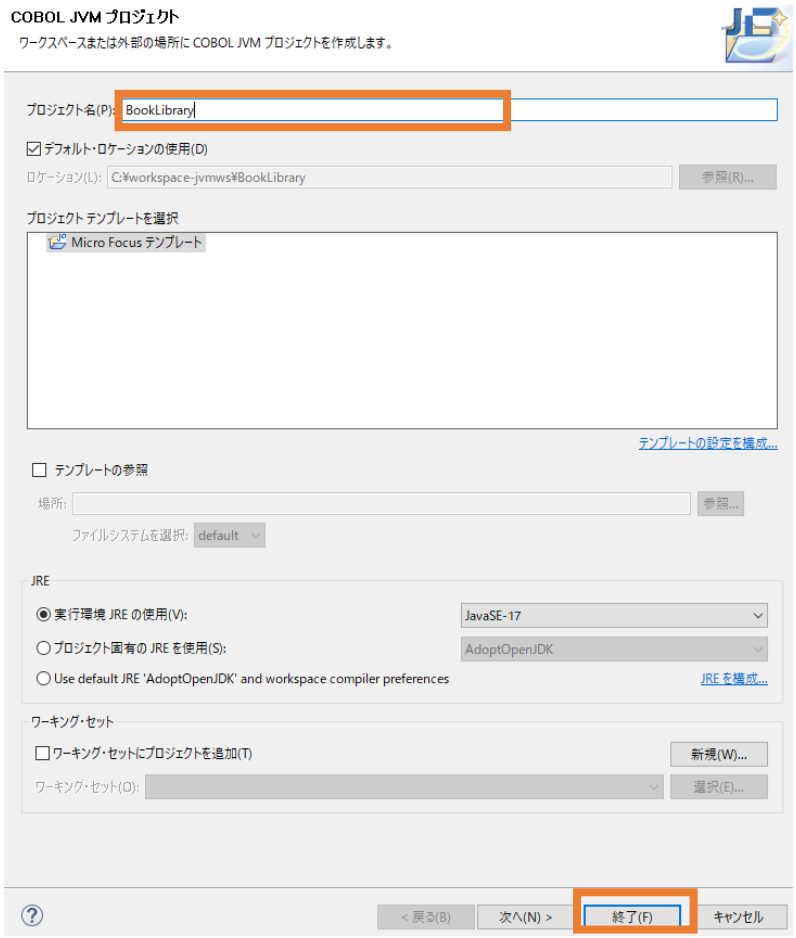
3.2 JVM COBOL プロジェクトの作成

本節では、さきほど確認した従来の COBOL プログラムを JVM COBOL としてコンパイルを行います。

1) Visual COBOL for Eclipse メニューより、[ファイル(F)] > [新規(N)] > [COBOL JVM プロジェクト] を選択します。

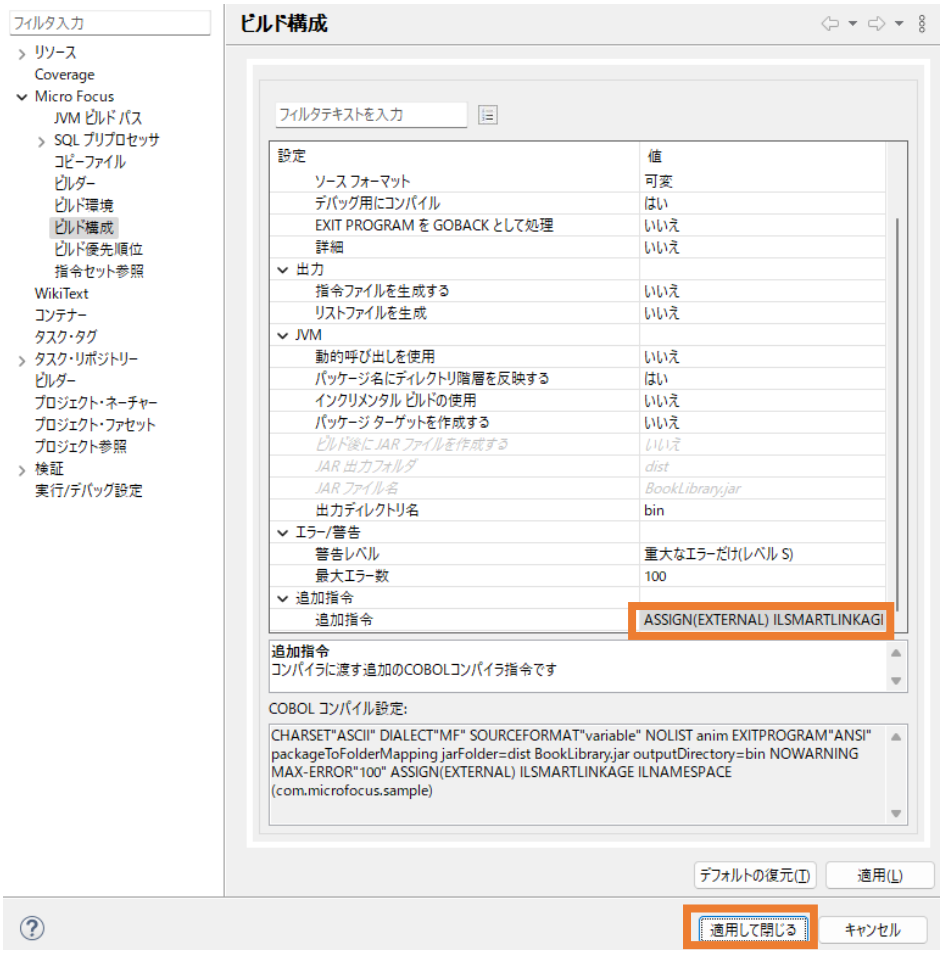


2) 「プロジェクト名」に “BookLibrary” を入力して、[終了(F)] をクリックします。



3) BookLibrary プロジェクトを選択し、マウスの右クリックにてコンテクストメニューを表示したうえで、[プロパティ(R)] を選択します。

4) [Micro Focus] > [ビルド構成] を選択し、「追加指令」に “ASSIGN(EXTERNAL) ILSMARTLINKAGE ILNAMESPACE(com.microfocus.sample)” を設定し、[適用して閉じる] をクリックします。

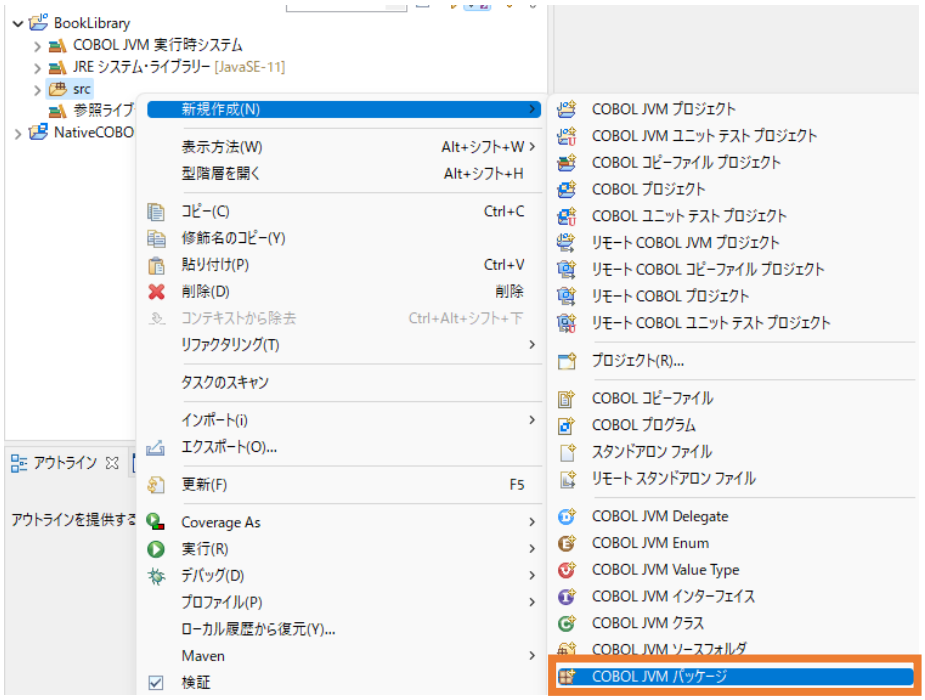


補足）

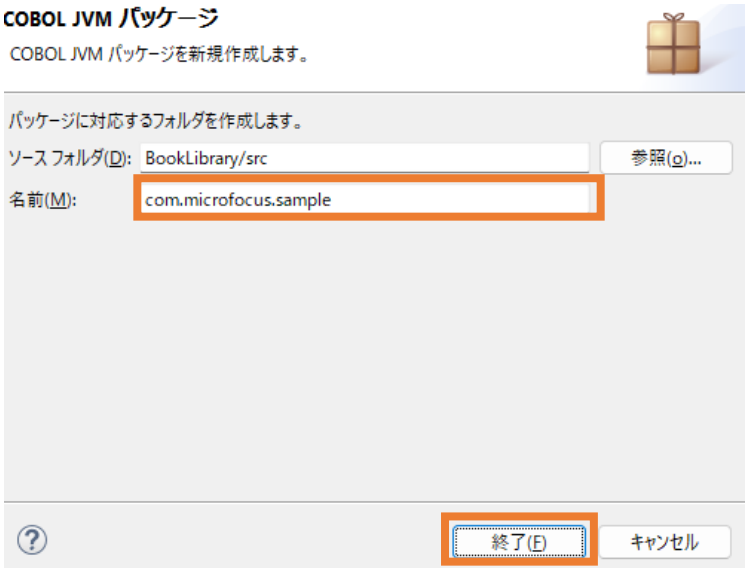
ILSMARTLINKAGE コンパイラ指令により、COBOL と Java 言語でのデータ型の差異を吸収するラッパークラスがコンパイル時に自動生成されます。本指令の詳細については、製品マニュアルトップより、[リファレンス] > [コンパイラ指令] > [コンパイラ指令 - アルファベット順一覧] > [ILSMARTLINKAGE] を参照してください。

ILNAMESACE コンパイラ指令は、JVM COBOL の出力先のパッケージ階層を指定するもので、今回は、com.microfocus.sample パッケージ配下に出力します。本指令の詳細については、製品マニュアルトップより、[リファレンス] > [コンパイラ指令] > [コンパイラ指令 - アルファベット順一覧] > [ILNAMESPACE] を参照してください。

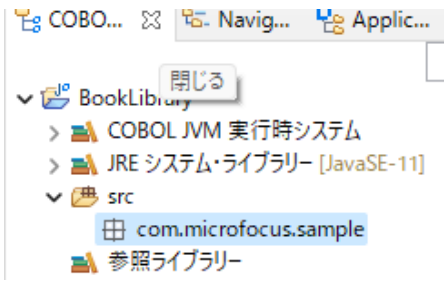
5) BookLibrary プロジェクト配下の src フォルダを選択し、マウスの右クリックにてコンテクストメニューを表示した上で、[新規作成(N)] > [COBOL JVM パッケージ] を選択します。



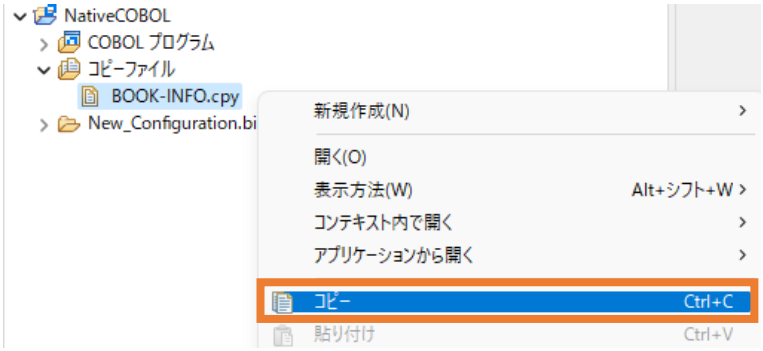
6) 「名前」に “com.microfocus.sample” を入力して、[終了(F)] をクリックします。



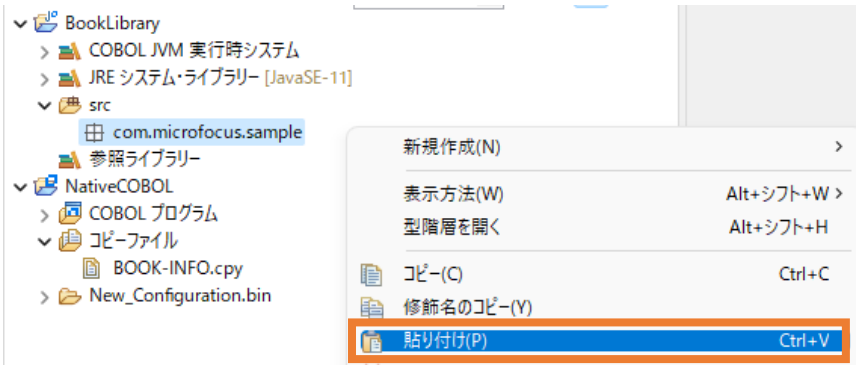
src フォルダ配下に、空のパッケージが作成されます。



7) さきほどの NativeCOBOL プロジェクトの「コピーファイル」配下の “BOOK-INFO.cpy” を選択し、マウスの右クリックにてコンテクストメニューを表示したうえで、[コピー] を選択します。

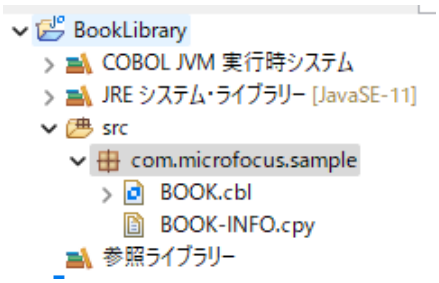


8) BookLibrary プロジェクト配下の src¥com.microfocus.sample フォルダを選択し、マウスの右クリックにてコンテクストメニューを表示したうえで、[貼り付け(P)] を選択します。

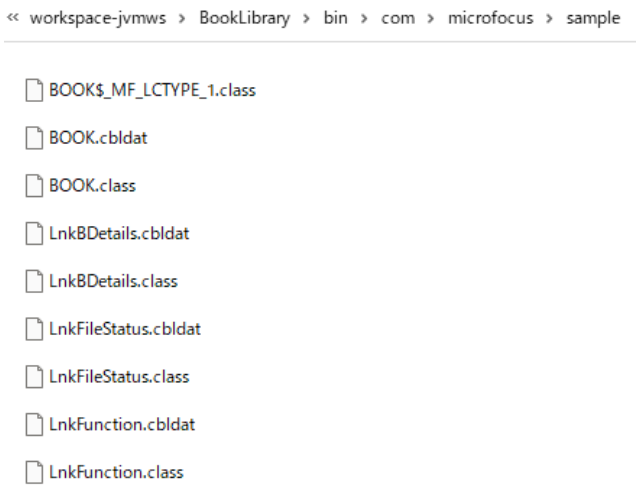


この操作により、BOOK-INFO.cpy が com.microfocus.sample フォルダ配下にコピーされます。

9) さきほどと同様の手順で、NativeCOBOL プロジェクトの「COBOL プログラム」配下の “BOOK.cbl” を BookLibrary プロジェクト配下の src¥com.microfocus.sample フォルダにコピーしてください。コピー後は以下のように表示されます。



デフォルトで自動的にビルドが有効になっている場合、BOOK.cbl をコピーした段階でコンパイルが行われ、プロジェクトが保存されたフォルダ配下の bin¥com¥sample フォルダに、以下の .class ファイル、すなわち、Java バイトコードが生成されます。

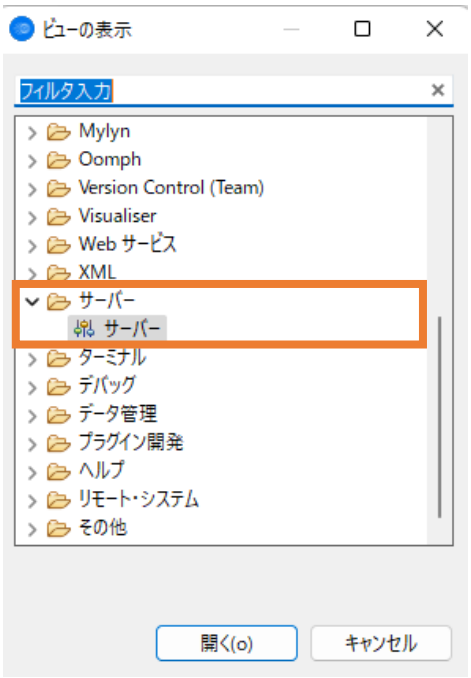


補足）

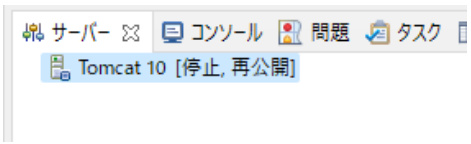
自動的にビルドする設定は、[プロジェクト(P)] ＞ [自動的にビルド(M)] で確認できます。チェックされている場合は自動的にビルドが行われます。

3.3 Apache Tomcat 上で動作する RESTful Web サービスの作成

本節を実行するにあたり、Visual COBOL for Eclipse 上で、予めサーバー設定を実施してください。この設定は、以降の手順で使用する J2EE パースペクティブのデフォルトで表示されるサーバービューで行うことができます。また、異なるパースペクティブを使用している際でも、Eclipse のメニューより、[ウィンドウ(W)] > [ビューの表示(V)] > [その他(O)] を選択した上で表示されるダイアログ上で、以下のビューを選択することで行えます。

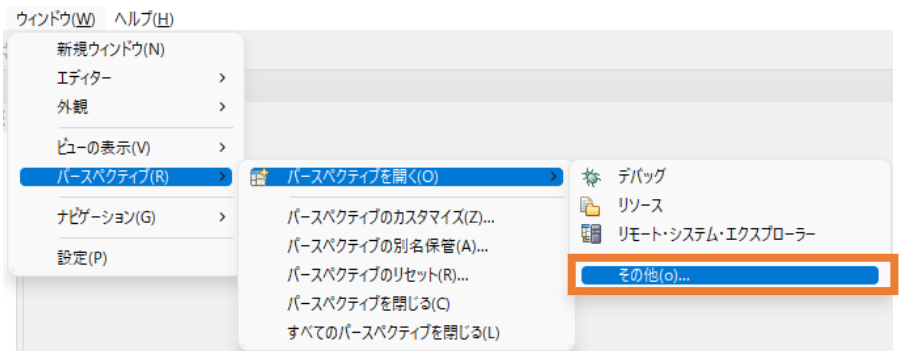


本手順では 以下のように Tomcat 10 のサーバー構成を行っています。

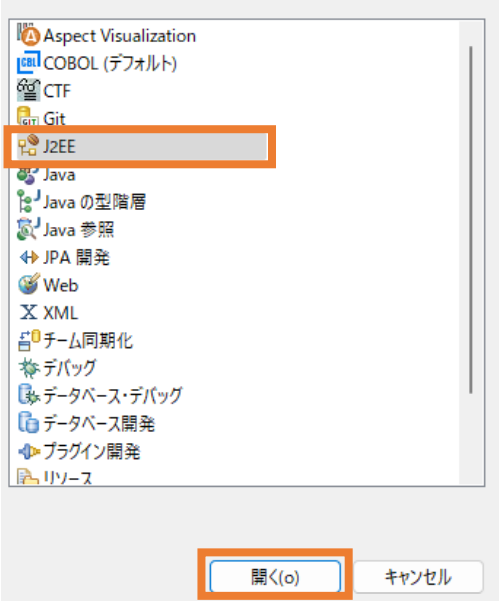


3.3.1 Java Web アプリケーションプロジェクトの作成

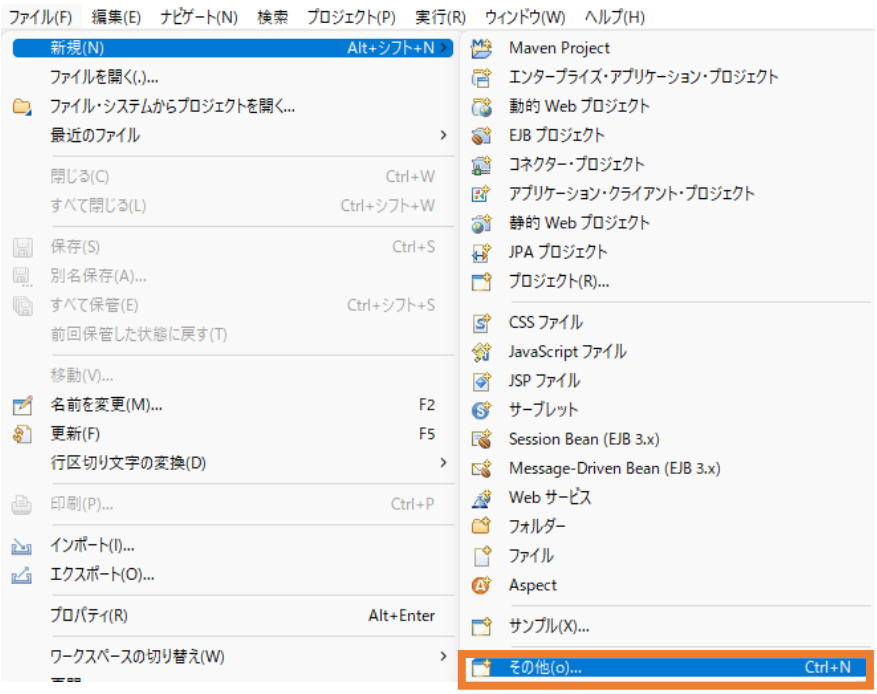
1) [ウィンドウ(W)] > [パースペクティブ(R)] > [パースペクティブを開く(O)] > [その他(o)] を選択します。



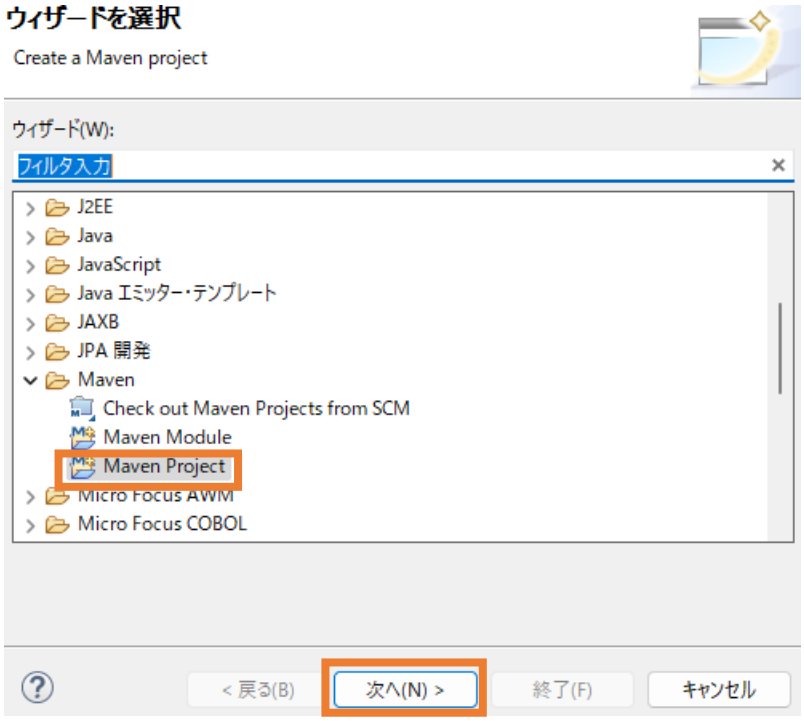
2) 「J2EE」を選択して、[開く(o)] をクリックします。



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



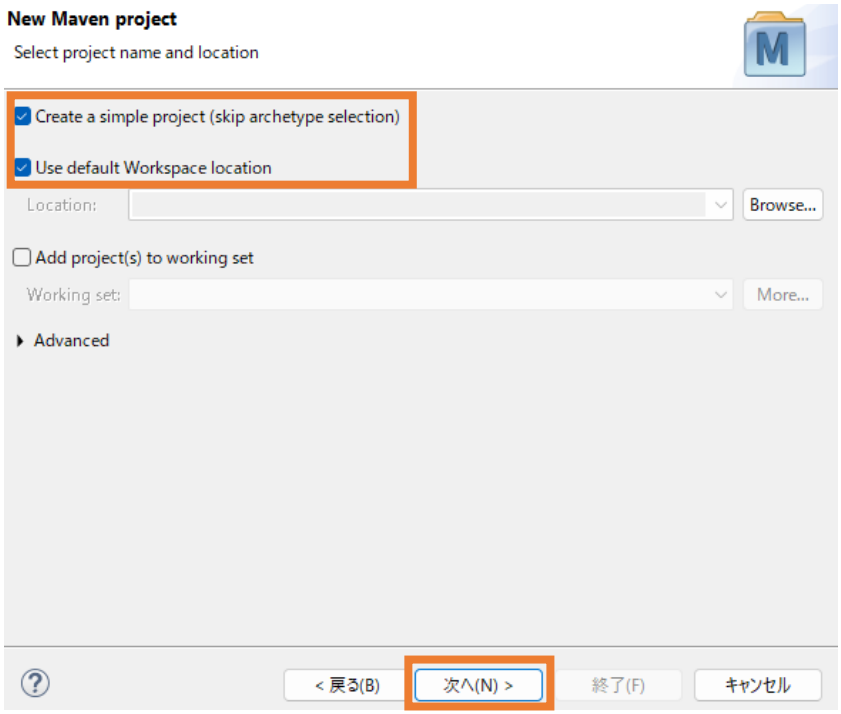
4) [Maven] > [Maven Project] を選択し、[次へ(N)]をクリックします。



5) 以下の設定を行い、[次へ(N)] をクリックします。

Create a simple project (skip archetype selection) にチェック

Use default Workspace location にチェック



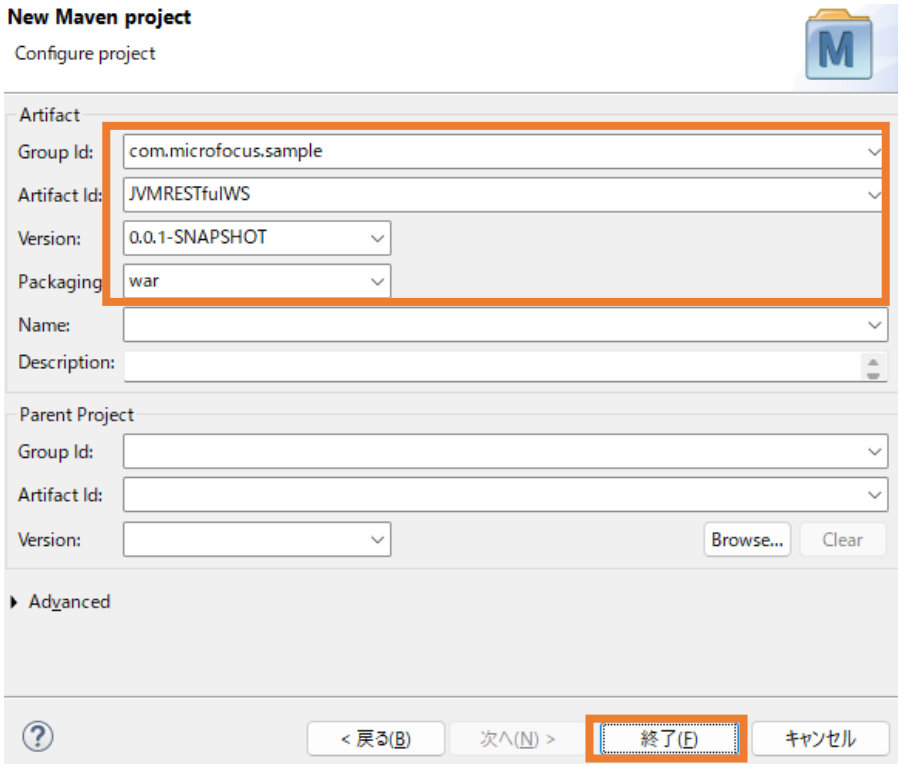
以下の入力を行い、[終了(F)] をクリックします。

Group Id: “com.microfocus.sample”

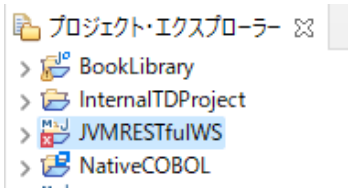
Artifact Id: “JVMRESTfulWS”

Version: “0.0.1-SNAPSHOT”

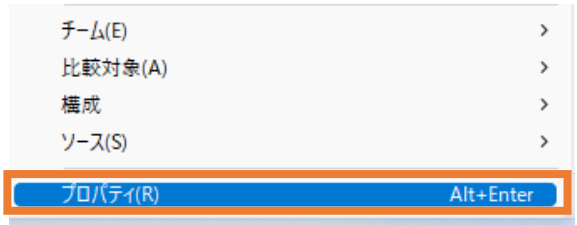
Packaging: “war”



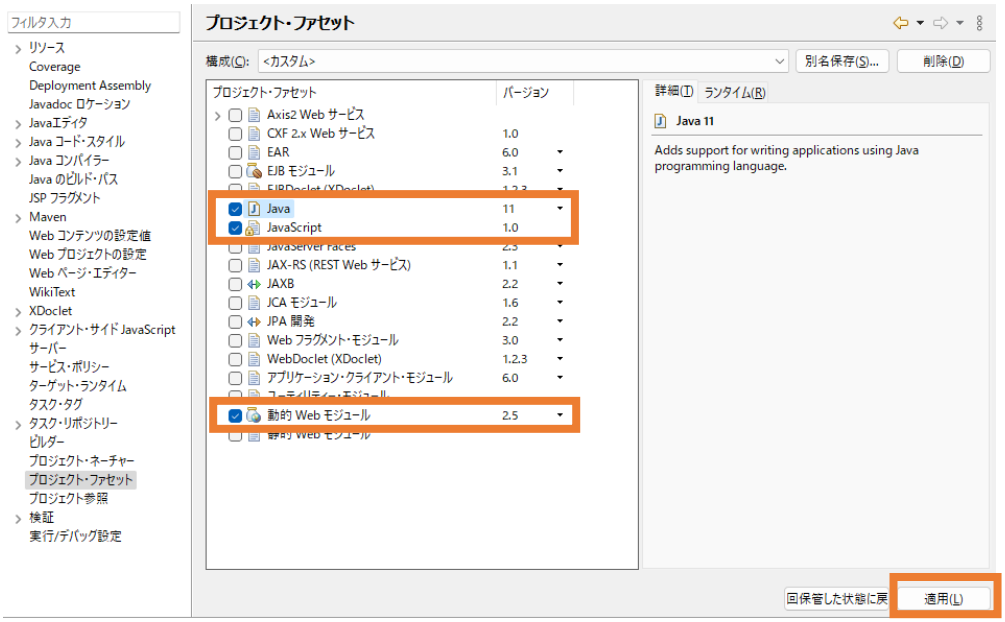
JVMRESTfulWS プロジェクトが作成されます。



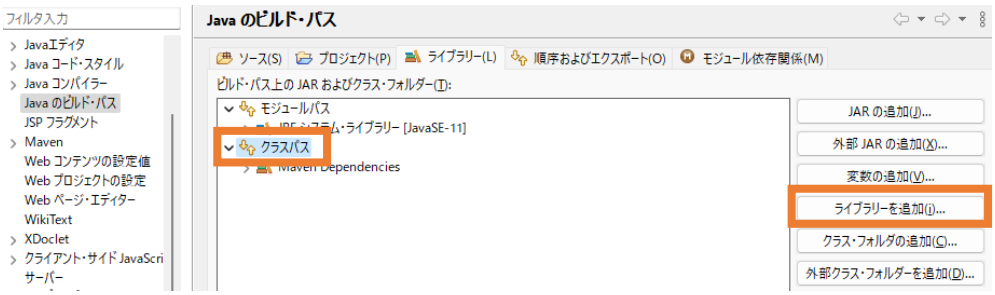
6) JVMRESTfulWS プロジェクトを選択し、マウスの右クリックにてコンテクストメニューを表示したうえで、[プロパティ(R)] を選択します。



7) [プロジェクト・ファセット]を選択し、「Java」に“11”を選択し、「JavaScript」と「動的 Web モジュール」にチェックを入れて[適用(L)]をクリックします。※[プロジェクト・ファセット]を選択した際、「Convert to faceted from…」リンクが表示される場合は、リンクをクリックしてください。

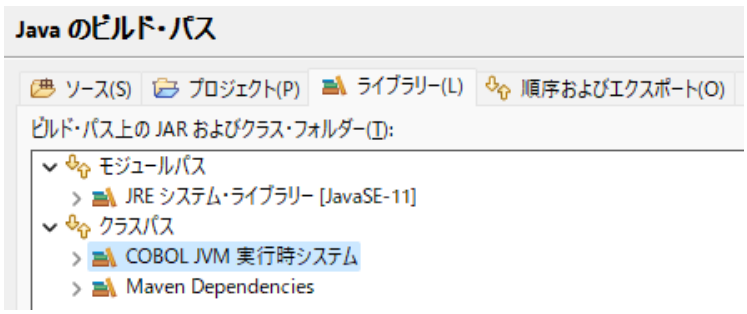


8) [Java のビルド・パス]から[ライブラリー]タブを選択します。「クラスパス」を選択したうえで、[ライブラリーの追加(i)]をクリックします。

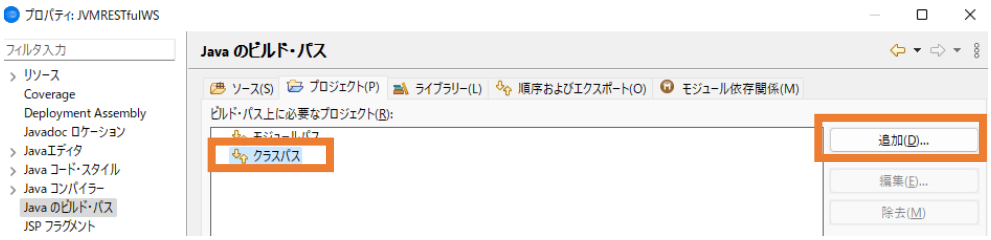


9) [COBOL JVM 実行時システム]を選択したうえで、[次へ(N)]をクリックします。

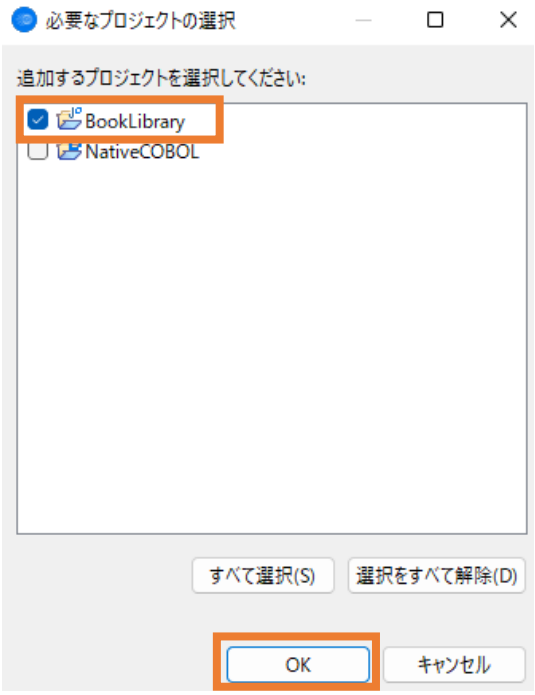
次の画面では、そのまま [終了(F)] をクリックすると、「COBOLJVM 実行時システム」が追加されます。



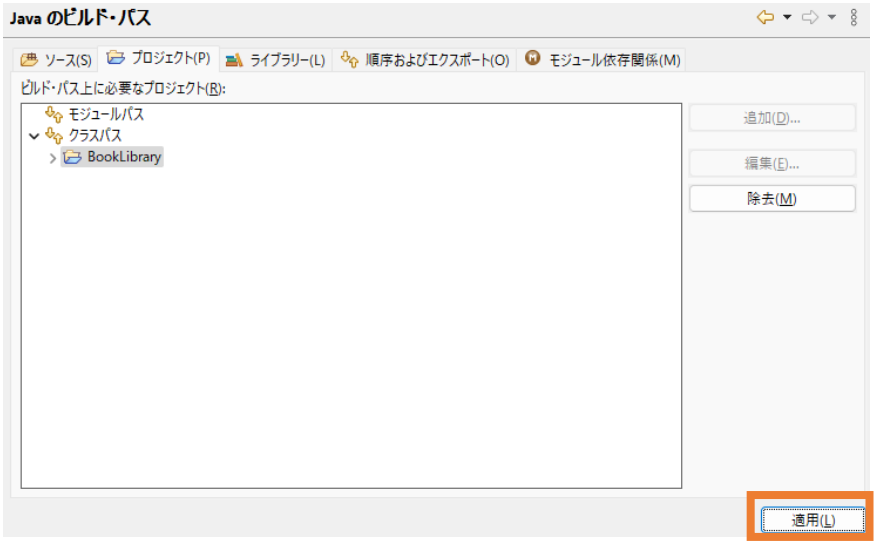
10) [プロジェクト]タブを選択し、「クラスパス」を選択したうえで、[追加(D)] をクリックします。



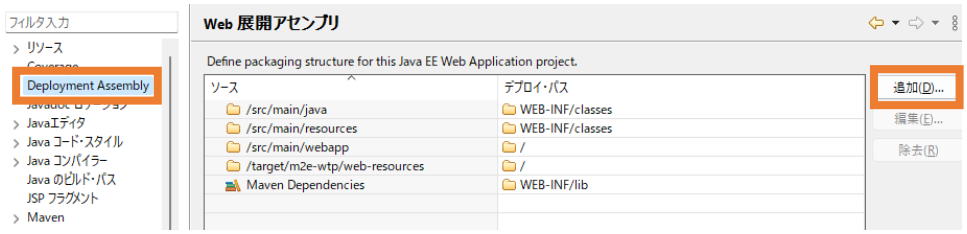
11) 「BookLibrary」プロジェクトにチェックを入れたうえで、[OK] をクリックします。



BookLibrary プロジェクトが追加されたことを確認したうえで、[確認(L)] をクリックします。



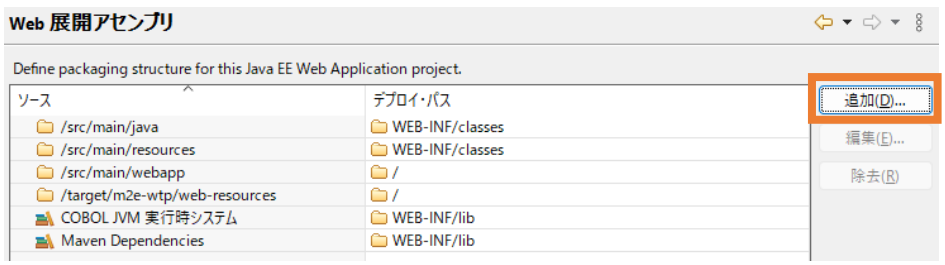
12) [Deployment Assembly] を選択し、[追加(D)] をクリックします。



13) [Java ビルド・パス・エントリー] を選択し、[次へ(N)] をクリックします

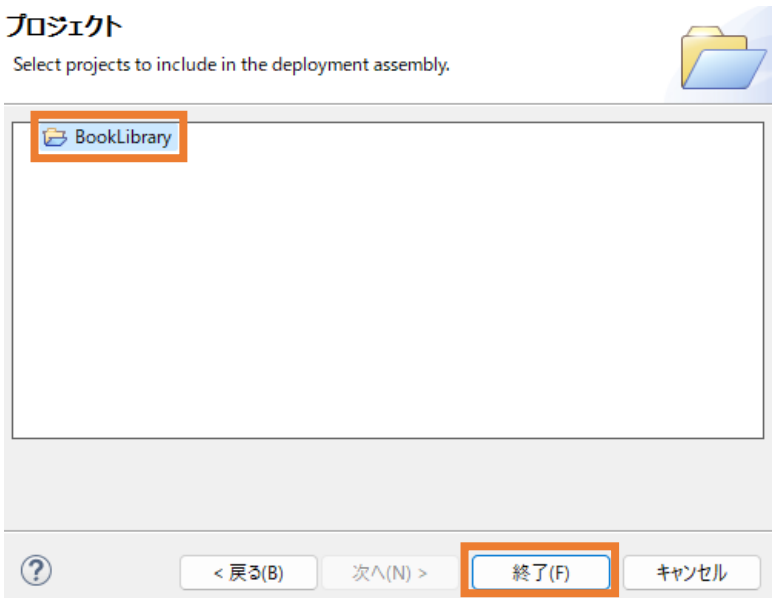
[COBOL JVM 実行時システム] を選択し、[終了(F)] をクリックします。

14) さきほど同様、[追加(D)] をクリックします。

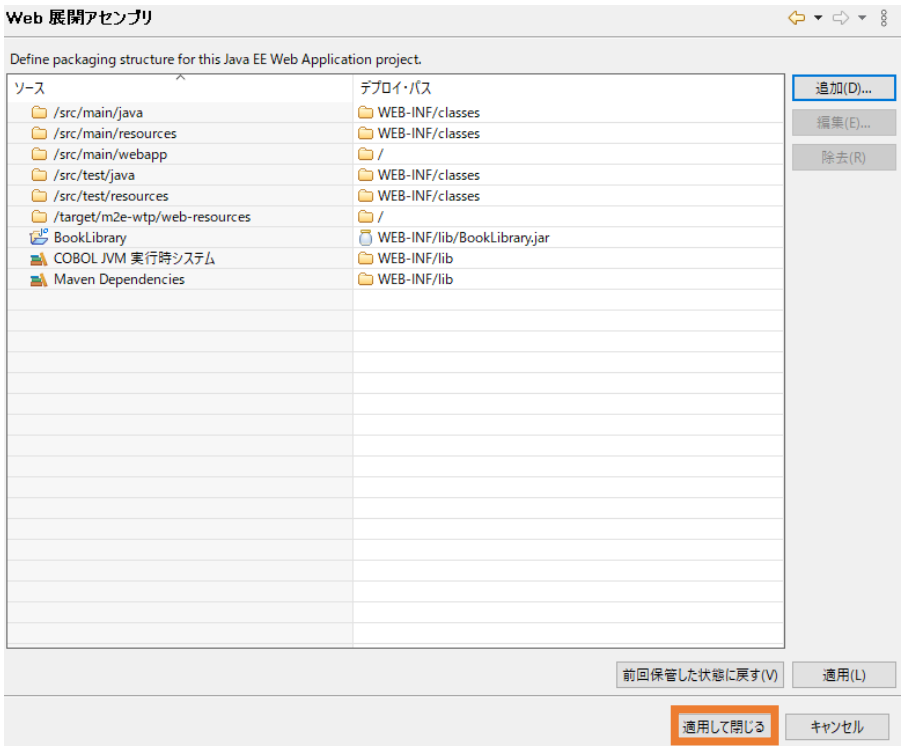


15)「プロジェクト」を選択し、[次へ(N)] をクリックします。

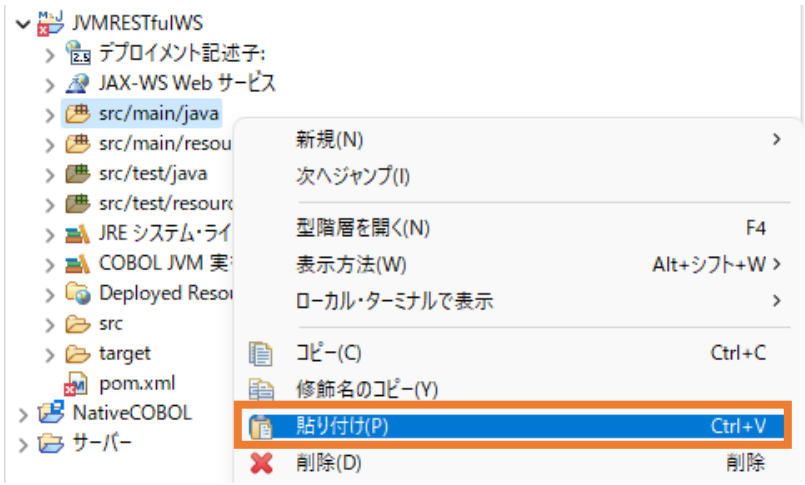
「BookLibrary」を選択し、[終了(F)] をクリックします。



以下のように追加した項目が表示されていることを確認して、[適用して閉じる] をクリックします。



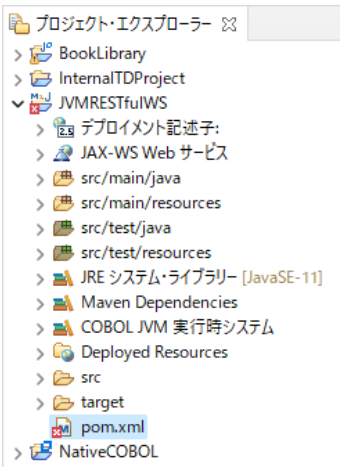
16) C:¥jvmRESTfulWSTutorial¥Java¥src 配下の com フォルダをクリップボードにコピーしたうえで、JVMREstfulWS プロジェクト配下の src/main/java フォルダを選択した状態でマウスの右クリックにてコンテクストメニューを表示して、[貼り付け(P)] を選択します。



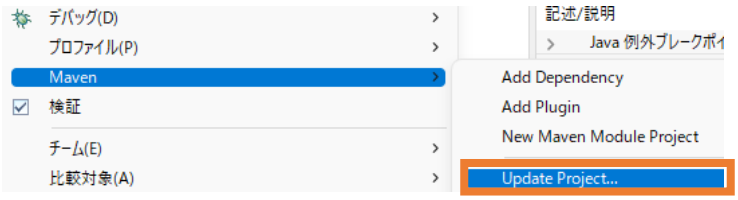
この時点ではエラーになりますが、ここでは無視します。

17) C:¥jvmRESTfulWSTutorial¥Java¥webapps 配下の WEB-INF フォルダをクリップボードにコピーしたうえで、JVMRESTfulWS プロジェクト配下の src¥main¥webapps フォルダを選択し、マウスの右クリックにてコンテクストメニューを表示したうえで、[貼り付け(P)] を選択します。

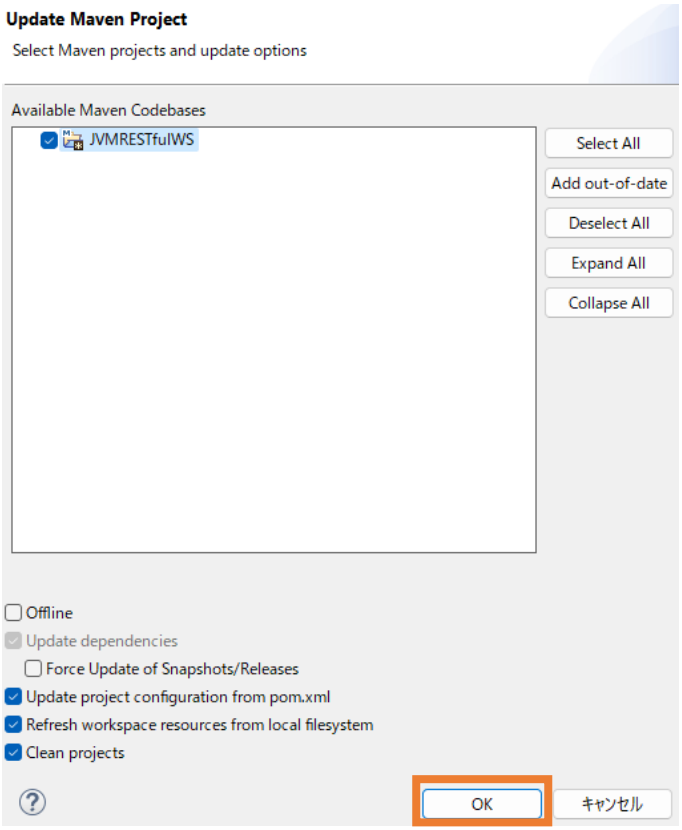
18) JVM RESTfulWS プロジェクト配下の pom.xml の中身を以下のファイルで上書き保存します。
C:¥jvmRESTfulWSTutorial¥Java¥pom.xml



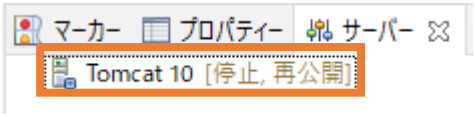
19) pom.xml を選択し、マウスの右クリックにてコンテクストメニューを表示したうえで、[Maven] > [Update Project] を選択します。



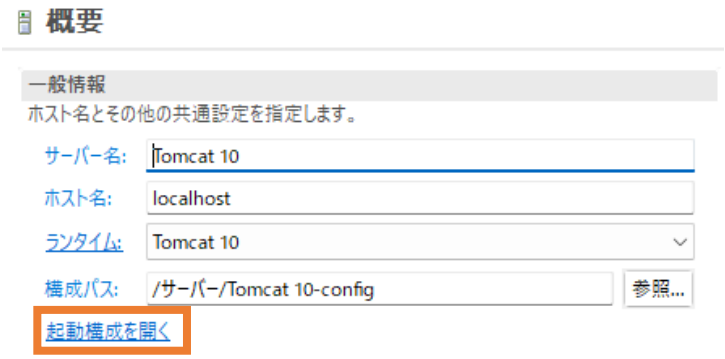
JVMRESTfulWS にチェックがついていることを確認したうえで [OK] をクリックしてください。



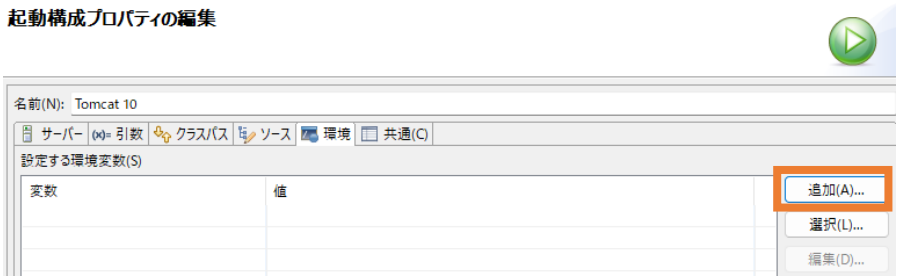
20) サーバービューを開き、「Tomcat 10」サーバーをダブルクリックします。



21) [起動構成を開く] リンクをクリックします。



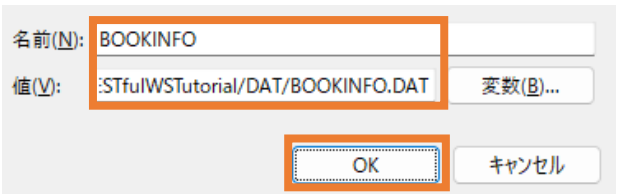
22) [環境]タブを開き、[追加(A)] をクリックします。



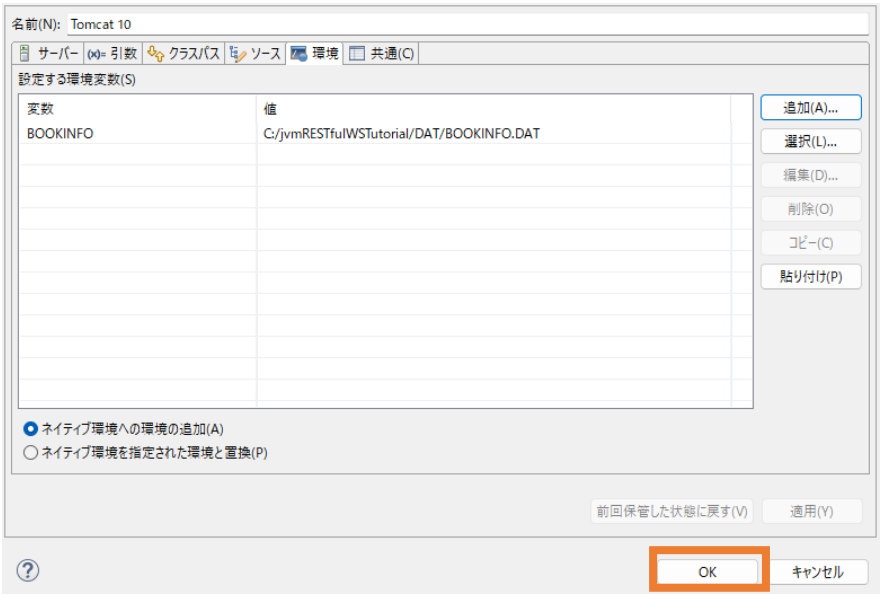
23) 以下の入力を行い、[OK] をクリックします。

名前： “BOOKINFO”

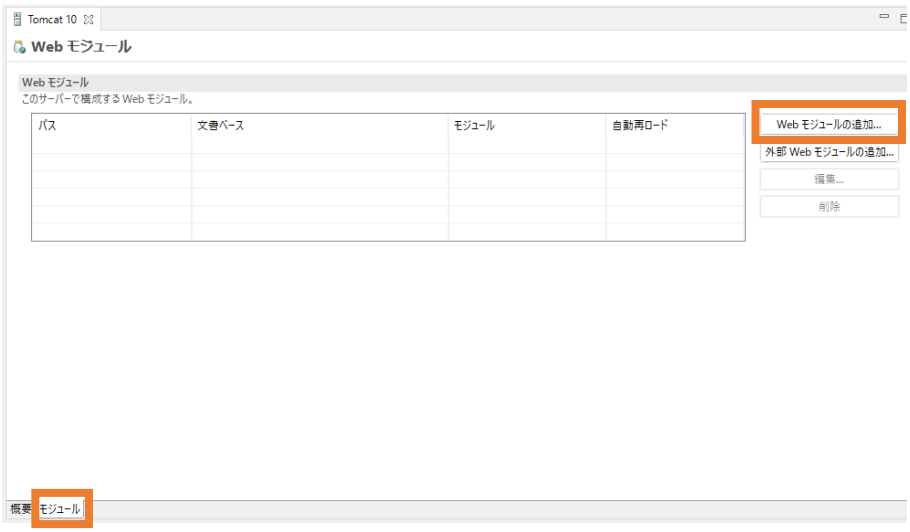
値： “C:/jvmRESTfulWSTutorial/DAT/BOOKINFO.DAT”



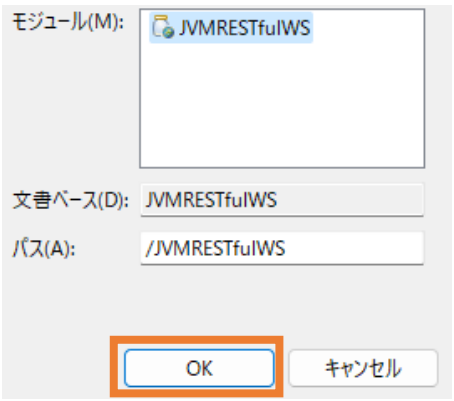
BOOKINFO 変数が追加されたことを確認して、[OK] をクリックします。



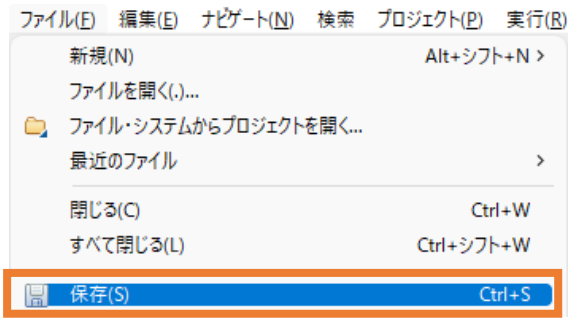
24) [モジュール]タブを選択し、[Web モジュールの追加] をクリックします。



JVMRESTfulWS を選択し、[OK] をクリックします。



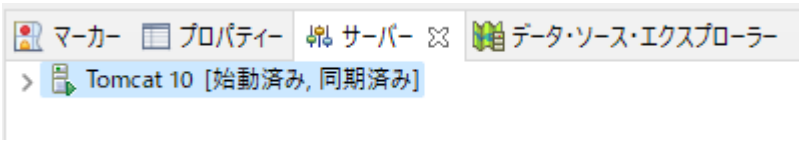
Eclipse IDE のメニューより [ファイル(F)] > [保存(S)] を選択し、変更を保存してください。



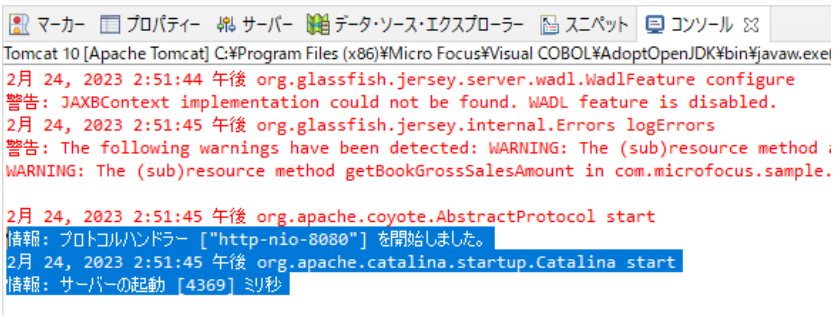
3.3.2 RESTful Web サービスの確認

1) 「Tomcat 10」サーバーを選択し、マウスの右クリックにてコンテクストメニューを表示したうえで、[起動(S)] を選択します。

起動すると、ステータスが “始動済み” に変更されます。



また、コンソールビューでもサーバーの起動を確認できます。

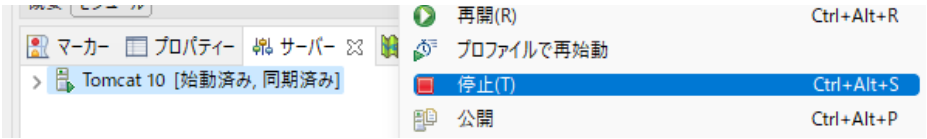


2) C:¥jvmRESTfulWSTutorial¥WebClient 配下の Search.html を Web ブラウザーで開きます。

3) ストック番号に “1111” を入力し、[検索] をクリックすると、先に確認したように結果が画面上に戻されます。

このように、Web 画面から Java で作成した RESTful Web サービスを介して COBOL ロジックを呼び出すことが確認できます。このインターフェースは、一般的な RESTful Web サービス仕様に基づいているため、COBOL 資産を活用した他システムとのデータ連携が容易に行えるようになります。

Eclipse IDE のサーバービューに戻り、[Tomcat 10] を選択し、マウスの右クリックにてコンテクストメニューを表示したうえで、[停止(T)] をクリックしてサーバーを停止します。



補足）

今回は、Eclipse IDE 上から Tomcat アプリケーションサーバーを起動し、RESTful Web サービスを稼働させていましたが、Eclipse IDE を使用せず、Tomcat をはじめとしたアプリケーションサーバー上でも今回のサービスを稼働させることができます。

WHAT'S NEXT

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

免責事項

ここで紹介したソースコードは、機能説明のためのサンプルであり、製品の一部ではございません。ソースコードが実際に動作するか、御社業務に適合するかなどに関しまして、一切の保証はございません。 ソースコード、説明、その他すべてについて、無謬性は保障されません。

ここで紹介するソースコードの一部、もしくは全部について、弊社に断りなく、御社の内部に組み込み、そのままご利用頂いても構いません。

本ソースコードの一部もしくは全部を二次的著作物に対して引用する場合、著作権法の精神に基づき、適切な扱いを行ってください。