# Micro Focus Visual COBOL チュートリアル

# COBOL 開発: Eclipse - JVM COBOL プログラムの単体テスト

#### 1. 目的

本チュートリアルでは、JVM COBOL プログラムに対するテスト作成、実行方法、および、テスト結果を表示させる方法の習得を目的として います。

Visual COBOL 製品に付属している MFUnit は、xUnit 系の単体テストフレームワークです。xUnit はオブジェクト指向型の単体テスト フレームワーク SUnit に起源を持つ JUnit や RUnit等の単体テストフレームワークの総称です。MFUnit は xUnit の設計アーキテクチ ャーや仕組みは取り入れつつも COBOL 開発者にとって扱いやすい手続き型の COBOL を対象とした単体テストフレームワークという設計 思想の下、開発されました。

MFUnit は COBOL 開発作業に以下の利点を提供します。

- テストを繰返し実行させることができるため、修正作業時などのテスト工数の削減が見込める
- Jenkins などの継続的インテグレーション (Continuous Integration) ツールと連携によりテストの自動化が行え、DevOps サイクルの導入が足がかりを作れる

MFUnit は JVM COBOL に対するテストを記述することもできますが、JVM COBOL は 他の Java 言語と同様、Java バイトコードを 生成します。このため、Java プログラムのように JUnit 単体テストフレームワークを使用することで、より統一性のあるテスト運用が可能です。 本チュートリアルでは、JUnit 単体テストフレームワークを利用したテスト実装方法を学びます。

#### 2. 前提

- 本チュートリアルで使用したマシン OS : Windows 10
- Micro Focus Visual COBOL 7.0 for Eclipse がインストール済みであること

本資料は、JVM COBOL に対する単体テストフレームワークの利用方法を記載したチュートリアルです。ネイティブ COBOL の単体テ スト実現方法については、別チュートリアルを参照ください。

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。 サンプルプログラムのダウンロード

# **MICRO**<sup>®</sup> FOCUS

# 内容

- 1. 目的
- 2. 前提
- 3. チュートリアル手順の概要
- 3.1. IDE からの実行
  - 3.1.1. Eclipse の起動
  - 3.1.2. チュートリアルプロジェクトの作成
  - 3.1.3. JVM COBOL プログラムの作成
  - 3.1.4. JUnit プロジェクトの作成
  - 3.1.5. JUnit テストプログラムの実行
- 3.2. コマンドラインからの実行



### 3. チュートリアル手順の概要

### 3.1. IDE からの実行

#### 3.1.1. Eclipse の起動

1) スタートメニューより、Visual COBOL for Eclipse を起動します。



2) ワークスペースを指定し、[起動(L)] ボタンをクリックします。

Eclipse ランチャー		×
ディレクトリーをワークスペースとして選択		
	+	
LCIIpsela、アンスマスパレントのとに用して、境況設定と開発成素物を体行しる	9 .	
ワークスペース( <u>W)</u> : C:¥workspace_tut_jvm_junit	✓ 参照( <u>B</u> )	
□この選択をデフォルトとして使用し、今後この質問を表示しない(U)		
▶ 最近のワークスペース( <u>R</u> )		
	+2540 +2640	
	起動(L) キャンセル	



### 3.1.2. チュートリアルプロジェクトの作成

1) Eclipse IDE メニューより、[ファイル(F)] > [新規(N)] > [COBOL JVM プロジェクト] を選択してください。

🥯 w	🕑 workspace_tut_jvm_junit - Eclipse								
ファイ	Ίル(F)	編集(E)	リファクタリング	ナビゲート(N)	検索	プロジ	፤//	ト(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 プロジェクト	
	ファイル	を開く(.)					📸	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					۲ġ	🛊 リモート COBOL プロジェクト		
	最近Ø	ロファイル				>	<b>(</b>	リモート COBOL コピーファイル プロジェクト	
	閉じる	(C)			Ctrl	+W	<b>€</b> Û	COBOL ユニット テスト プロジェクト	
	すべて	、-/ 閉じる(L)		Ct	rl+シフト	+W	얟	COBOL JVM プロジェクト	
							4	リモート COBOL JVM プロジェクト	

- 2) 以下の入力を行い、[終了(F)] ボタンをクリックします。
  - プロジェクト名: AirportDemoJVMJUnit

JRE: デフォルト JRE の使用

#### 補足)

JRE については、各環境にあわせて異なる選択をして頂いても問題ありません。

■ COBOL JVM プロジェクトの新規作成					×
<b>COBOL JVM プロジェクト</b> ワークスペースまたは外部の場所に COBOL JVM プロジェクトを作成します。					\$
ブロジェクト名(P): AirportDemoJVMJUnit					] ^
ロケーション(L): C¥70workspace-tut¥AirportDemoJVMJUnit			参照	R(R)	
プロジェクトテンプレートを選択		1	L @ 50.5	= ≠ +# -₽	
□ テンプレートの参照		<u>7971</u>	ノートの設定	<u> Eを構成</u>	
場所; ファイルシステムを選択: default ~			参照		
JRE					
○ 実行環境 JRE の使用(V):	JavaSE-11			$\sim$	
〇 プロジェクト固有の JRE を使用(S):	AdoptOpenJDK			$\sim$	
Use default JRE 'AdoptOpenJDK' and workspace compiler preferences			JRE ?	<u>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(\	N)	
ワーキング・セット(O):		$\sim$	選択(	E)	Ľ.
					*
? < 戻る(8)	次へ(N) >	終了(F)		キャンセノ	ŀ



プロジェクトが作成されたことを確認します。



3) AirportDemoJVMJUnit プロジェクト配下の「src」フォルダを選択した上で、マウスの右クリックにてコンテクストメニューを



4) [一般] > [ファイル・システム] を選択し、[次へ(N) >] ボタンをクリックします。

<ul> <li>インボート</li> </ul>		×
選択 ローカル・ファイル・システムから既存のプロジェクトへリソースをインボートします。	2	1
インポート・ウィザードの選択(S):		
Jィルタ入力		
<ul> <li>◇ ② 一般</li> <li>③ アーカイブ・ファイル</li> <li>○ ファイル・システム</li> <li>③ フォルターまたはアーカイブ由来のプロジェクト</li> <li>② 既存プロジェクトをワークスペースへ</li> <li>□ 設定</li> <li>&gt; ② AWS</li> <li>&gt; ② EB</li> <li>&gt; ② Gradle</li> <li>&gt; ② ZEE</li> <li>&gt; ② Micro Focus</li> <li>&gt; ③ Micro Focus インターフェイスマッパー</li> <li>&lt; ③ Opende</li> </ul>		~
(?)          次へ(N) >         終了(F)	キャンセ	μ

COBOL 開発: Eclipse - JVM COBOL プログラムの単体テスト



5) [参照(R)] ボタンをクリックし、サンプルファイルを展開したフォルダ内の ProjectData¥src フォルダを選択し、src フォルダに チェックした上で、[終了(F)] ボタンをクリックします。

インポート	—		(
<b>ファイル・システム</b> ローカル・ファイル・システムからリソースをインポートします。			7
次のディレクトリーから(Y): C:¥VC50TutorialJVMUnit¥ProjectData¥src	~	参照(R)	]
	Image: space of the system     Image: space of the system       Image: space of the system		
タイフをフィルター(T) すべて選択(S) 選択をすべて解除(D)		±070 AAD	
<ul> <li>1 ノルート元 ノオルダ(L): AirportDemoJVMJUnit/src</li> <li>オブション         <ul> <li>第告を出さずに既存リソースを上書き(O)</li> <li>トップ・レベルのフォルダーを作成(C)</li> <li>拡張 &gt;&gt;(A)</li> </ul> </li> </ul>		参照(W)	]
?	< 戻る(B) 次へ(N) > 終了(F)	キャンセル	

COBOL エクスプローラー画面より、インポートが完了したことを確認します。





- 6) AirportDemoJVMJUnit プロジェクト配下を選択した状態で、再度、上記インポート手順を以下の方法にて実施します。
  - ・ [参照(R)] ボタンクリック後のフォルダ選択にて、サンプルファイルを展開したフォルダ内の ProjectDataを選択
  - ・ airports.dat をインポート

インポート後、以下のようになります。



## 3.1.3. JVM COBOL プログラムの作成

前手順でインポートした aircode.cbl はレガシーなネイティブ COBOL プログラムです。本手順では、aircode.cbl に一切の変 更を加えることなく、JVM COBOL として機能するよう、Wrapper クラスを作成します。

AirportDemoJVMJUnit プロジェクト名を選択した状態で、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、[新規作成(N)] > [COBOL JVM クラス]を選択します。

😤 C 🔀 陆 ナビ 😤 A	p	📕 サ− 🖳 🗖	🧧 aircode.cbl 🔀		
	<b>~</b>	= 🔁 🙀 ∽	aircode.cbl 🕨		
airports.dat		新規作成(N)	>	曾	COBOL JVM プロジェクト
> 🛋 COBOL JVM 実行時	寺 a	型階層を開く	Alt+シフト+H	<b>않</b>	COBOL JVM ユニット テスト プロジェクト
v ∰ src	D	<u>ີ</u> ∎ ⊐ピ−(C)	Ctrl+C	18 18	COBOL Jビーファイル ノロジェクト COBOL プロジェクト
<ul> <li>◆ 冊 (デフォルト・パッケー</li> <li>◆ @ aircode.cbl</li> <li>&gt; @ aircode</li> <li>imparams.cpy</li> <li>imparams.cpy</li> <li>aircode.cpy</li> <li>★部ワイブラリー</li> </ul>		修飾名のコピー(Y) 貼り付け(P) 削除(D) コンテキストから除去 終動(V)	<b>Ctrl+V</b> 削除 Ctrl+Alt+シフト+下	ないです。	COBOL ユニット テスト プロジェクト リモート COBOL JVM プロジェクト リモート COBOL コピーファイル プロジェクト リモート COBOL プロジェクト
_,		名前を変更(M)	F2		プロジェクト(R)
		ビルド アクション タスクのスキャン	>		COBOL コピーファイル COBOL プログラム
	4	インポート(i) エクスポート(O)	>		スタントアロン ファイル リモート スタンドアロン ファイル
ד <b>די די ד</b>	8	更新(F) プロジェクトを閉じる(S) Close Unrelated Proj	F5		COBOL JVM Delegate COBOL JVM Enum COBOL JVM Value Type COBOL JVM インターフェイス
<ul> <li>P aircode</li> <li>▶ P Data Division</li> </ul>		リモートシステムビューで 検証(V)	表示	C) T	COBOL JVM クラス COBOL JVM ソースフォルタ



- 2) 以下の入力を行い、[終了(F)] ボタンをクリックします。
  - パッケージ: "com.sample"
  - 名前: "AircodeWrapper"

COBOL JVM	クラスの新規作成			×
COBOL JVM ク COBOL JVM クラ	<b>ラス</b> スを新規作成します。			
ソース・フォルダ( <u>D</u> ):	AirportDemoJVMJUnit/src		参照(	<u>o</u> )
パッケージ( <u>K</u> ):	com.sample		参照(	<u>W</u> )
名前( <u>M</u> ): 修飾子:	AircodeWrapper            public(P) Ointernal(N)         abstract(I) final(L) static(C)		]	
スーパークラス( <u>S</u> ):			参照(	( <u>E</u> )
1 79-7I-X(I):			追加( 除去	<u>A</u> )
?		終了( <u>F</u> )	キャン	セル

AircodeWrapper プログラムが作成されたことを確認します。





3) サンプルファイルを展開したフォルダ内の AircodeWrapper.cbl でプログラムを上書き保存してください。

AircodeWrapper.cbl 🔀
AircodeWrapper.cbl
•••••*A·1·B·••···2···•3···••4····5
⊖ class-id com.sample.AircodeWrapper public.
working-storage section. 01 pp procedure-pointer.
*> Program linkage data 01 lnk-function PIC X.
88 get-matches value '1'.
88 get-distance value '2'.
88 get-details value '3'.
88 open-file value 4.
88 display-record value '6'
01 lnk-airport1 PIC X(3).
01 lnk-airport2 PIC X(3).
01 lnk-prefix-text PIC X(3).
01 lnk-distance-result.
03 distance-km PIC ZZ,ZZ9.
03 distance-miles PIC 22,229.
01 lnk-matched-codes PIC X(30) occurs 10
01 lnk-rec.
03 ap-code PIC X(4).
03 ap-name PIC X(30).
03 ap-city PIC X(30).
03 ap-country PIC X(20).
03 <del>ap-geo</del> .
07 ap-lat-sign PIC X
07 ap-lat-sign PIC X.
<

この Wrapper クラスは、com.sample.AircodeWrapper クラスとして、Java 言語から レガシーの COBOL を呼び 出せるよう、各種メソッドを定義しています。

#### 3.1.4. JUnit プロジェクトの作成

1) Eclipse IDE メニューより、[ファイル(F)] > [新規(N)] > [その他(O)] を選択します。



COBOL 開発: Eclipse - JVM COBOL プログラムの単体テスト



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

● 新規		
ウィザードを選択		-
Java プロジェクトの作成		
ウィザード <u>(W)</u> :		
7ብሥል አካ		
<ul> <li>✓ ⇒ Java</li> <li></li></ul>		^
●3 ソース・フォルタ ● パッケージ ※ 既存 Ant ビルド・ファイルからの Java プロジェクト ◎ 注釈 ●3 列挙型		
> lə Java の実行/テバック > lə JUnit		<b>v</b>
(P) < 戻る(B) 次へ(N) > 終了(E)	)	キャンセル

3) 以下の入力を行い、[終了(F)] ボタンをクリックします。

プロジェクト名: AirportDemoJUnit

IRE :	デフォルト	1RF	の使用
	1 2/1/21		UXIV

補足)

JRE については、各環境にあわせて異なる選択をして頂いても問題ありません。



● 新規 Java プロジェクト		– 🗆 X
J <b>ava プロジェクトの作成</b> Java プロジェクトをワークスペースまたは外部ロケーションに作成します。		
プロジェクト名(P): AirportDemoJUnit		
ロケーション(L): C:¥70workspace-tut¥AirportDemoJUnit		参照(R)
JRE		
○ 実行環境 JRE の使用(V):	JavaSE-11	$\sim$
〇 プロジェクト固有の JRE を使用(S):	AdoptOpenJDK	$\sim$
Use default JRE 'AdoptOpenJDK' and workspace compiler preferences		<u>JRE を構成</u>
プロジェクト・レイアウト		
○ プロジェクト・フォルダをソースおよびクラス・ファイルのルートとして使用(U)		
● ソースおよびクラス・ファイルのフォルダーを個別に作成(C)		<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(0):		選択(E)
<ul><li>(?)</li><li>&lt; 戻る(B)</li></ul>	次へ(N) > 終了(F)	キャンセル
· · · · · · · · · · · · · · · · · · ·		

以下のようなダイアログが表示された場合、[パースペクティブを開く(O)] ボタンをクリックします。

💿 関連	付けられたパースペクティブを開きますか?		×				
2	この種のプロジェクトは Java パースペクティ	びに関連付けられます。					
	<sup>ッ</sup> 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されています。パッケージ・エクスプロー ラー、型階層、および Java 固有のナビゲーション・アクションを提供します。						
	このパースペクティブを開きますか?						
□ 常に	この設定を使用する(R)	パースペクティブを開く(O)	いいえ(N)				

プロジェクトが作成されたことを確認します。





 4) AirportDemoJUnit プロジェクト名を選択した状態で、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、[ビルド・パス (B)] > [ビルド・パスの構成(C)] を選択します。

パッケージ・エクスプローラ-	- 83		E 😌 😜 🗸 🗆 🗆	T	
<ul> <li>&gt; AirportDemoJUnit</li> <li>&gt; AirportDemoJVMJU</li> <li>&gt; InternalTDProject</li> </ul>		新規(W) 次ヘジャンプ(I)	>		
		新規ウィンドウで開く(N) 型階層を開く(N) 表示方法(W)	F4 Alt+シフト+W >		
		コピー(C) 修飾名のコピー(Y) 貼り付け(P) 削除(D)	Ctrl+C Ctrl+V 削除		
	<u>.</u>	コンテキストから除去	Ctrl+Alt+シフト+下	L	
		ビルド・パス(B) ソース(S)	> Alt+シフト+S >	₩ #2	ソースのリンク(S) 新規ソース・フォルダー(N)
1		リファクタリング(T)	Alt+シフト+T >	<del>()</del>	ソース・フォルダとして使用(U)
	èn Zi	インポート(I) エクスポート(O)			外部アーカイブの追加(V) ライブラリーの追加(L)
	Ś	更新(F)	F5	8:	ビルド・パスの構成(C)

5) 「ライブラリー(L)」タブを選択し、[クラスパス]を選択したうえで [ライブラリーを追加(i)] ボタンをクリックします。

フィルタ入力 Java のビルド・パス	$\Diamond$	• =>	- 8
<ul> <li>&gt; リソース Coverage Javadoc Dブーション</li> <li>&gt; レバ・バス上の JAR およびクラス・フォルダー(T):</li> <li>&gt; Java ゴイウ</li> <li>&gt; Java ゴイウ</li> <li>&gt; Java コード・スタイル</li> <li>&gt; Java のビルド・バス Project Facets Task Tags</li> <li>&gt; Validation WikiText</li> <li>&gt; ダスク・リボジトリー ビルター</li> </ul>	JAR の追加(J 部 JAR の追加( 変数の追加(\ イブラリーを追か ス・フォルダの追 ラス・フォルダーを	1) ロ(X) マ/) ロ(i) むロ(C) を追加(C	)

6) 「JUnit」を選択し、[次へ(N) >] ボタンをクリックします。

💿 ライブラリーの追加			_		$\times$
<b>ライブラリーの追</b> 追加するライブラリー・	<b>加</b> タイプを選択します	ŧ.		8	3
COBOL JVM 実行時 CXF ランタイム EAR ライブラリー JRE システム・ライブラリ	システム -				
JUnit Maven Managed De Web App ライブラリー サーパー・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 接続可能性ドライパー	pendencies 定義				
?	< 戻る( <u>B</u> )	次へ( <u>N</u> ) >	終了( <u>F</u> )	キャンセノ	k

COBOL 開発: Eclipse - JVM COBOL プログラムの単体テスト



7) そのまま、[終了(F)] ボタンをクリックします。



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

🧿 プロパティ: AirportDemoJUnit		— C	]	$\times$
7ィルタ入力	Java のビルド・パス	< ◄	\$	<b>₩</b> 00
> リソース Coverage Javadoc ロケーション	(5) ジース(S) ジョブェクト(P) よ ライブラリー(L) 小 順序およびエクスポート(O) ジェント(R):			
> Javaエディタ > Javaコード・スタイル	<u>♣ ∓ジュールパ</u> ス ♠ クラスパス	追加(D)		
> Java コンパイラー Java のビルド・パス Project Facets Task Tags		編集(E) 除去(M)		



9) AirportDemoJVMJUnit プロジェクトにチェックを行い、[OK] ボタンをクリックします。

🧿 必要なプロジェクトのネ	<b>昱</b> 択	_		×
追加するプロジェクトを選	択してください:			
AirportDemoJ	VMJUnit			
	すべて選択(S)	選択を	すべて解除	£(D)
		_	h - h - le le	
	OK		キャンセノ	

10) AirportDemoJVMJUnit プロジェクトが追加されたことを確認し、[適用して閉じる] ボタンをクリックします。

🥏 プロパテ₁: AirportDemoJUnit		— 🗆 X
7ィルタ入力	Java のビルド・パス	<> ▼ ⇒ ₹ 8
<ul> <li>リソース</li> <li>Coverage</li> <li>JavaGC ロケーション</li> <li>Java コード・スタイル</li> <li>Java コード・スタイル</li> <li>Java コード・スタイル</li> <li>Java ロビルド・パ(ス)</li> <li>Project Facets</li> <li>Task Tags</li> <li>Validation</li> <li>WikiText</li> <li>タスク・リボジトリー</li> <li>ビルダー</li> <li>プロジェクト・参照</li> <li>裏行/デバッグ設定</li> </ul>	(些 ソース(S) ご プロジェクト(P) ▲ ライブラリー(L) ◇ 順序およびエクスボート(O) ④ モジュール依存関係(M) ビルド・パス上に必要なプロジェクト(R): ◇ セジュール(ス ◇ ☆ クラスパ(ス ◇ ご AirportDemoJVMJUnit	追加(D) 編集(E) 除去(M)
		適用(L)
?	適用し	て閉じる キャンセル

11) AirportDemoJUnit プロジェクト配下の src 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、[新規(W)] > [クラス]を選択します。



パッケージ・エクスプローラ	5- X		E 🕏 🗊	▽ □ □		
✓  AirportDemoJUni	t イゴニI	– [AdoptOpen IDK]				
⇒ and be set	1775	- [AdoptOpenDK]				
> 🛋 JUnit 5		新規(W)		>	鬯	Java プロジェクト
> 📂 AirportDemoJVM		新規ウィンドウで開く(N)			Û	AWS Java Project
> 🥃 InternalTDProject		型階層を開く(N)		F4	P	AWS Lambda Java Project
		表示方法(W)	Alt+	·シフト+W >	•	AWS Serverless Java Project
		71 <sup>2</sup> (0)		CHILC.		AWS Java Web Project
				Ctri+C	2	Aspect Project
	喧	19mi石のJビー(Y)				ノロジェクト(R)
	F	貼り付け(P)		Ctrl+V	ŧŶ	パッケージ
	×	削除(D)		削除	0	<b>クラス</b>
	<u>S.</u>	コンテキストから除去	Ctrl+Alt+	シフト+下	œ	インターフェース

# 12) 以下の入力を行い、[終了(F)] ボタンをクリックします。

パッケージ: com.sample

名前: Aircode	eWrapperTest		
🧿 新規 Java クラス			
<b>Java クラス</b> 新規 Java クラスを作成し	ます。		0
ソース・フォルダ( <u>D</u> ):	AirportDemoJUnit/src		参照( <u>o</u> )
<u>パッケージ(K</u> ):	com.sample		参照( <u>W</u> )
エンクロージング型(Y):			参照( <u>W</u> )
名前( <u>M</u> ): 修飾子:	AircodeWrapperTest         ● public(P)       ○ package       ○ private(\vee)       ○ pro         □ abstract(I)       □ final(L)       □ static(○	otected( <u>T</u> )	
スーパークラス( <u>S</u> ):	java.lang.Object		参照( <u>E</u> )
インターフェース( <u>i</u> ):			追加( <u>A</u> )
			除去( <u>R</u> )
作成するメソッド・スタブの道			
	<ul> <li>□ public static void main(String[] args)</li> <li>□ スーパークラスからのコンストラクター(C)</li> <li>○ 継承された抽象メソッド(H)</li> </ul>		
コメントを追加しますか? (テ	ンフレートの構成およびデフォルト値については <u>ここ</u> を参照) □ コメントの生成( <u>G</u> )		
?	終	了( <u>F</u> )	キャンセル



AircodeWrapperTest.java が作成されることを確認します。



13) サンプルファイルを展開したフォルダ内の AircodeWrapperTest.java でコードを上書き保存します。



一般的な JUnit テストが記述されており、そのテストプログラム内で、さきほど作成した AircodeWrapper クラスが通常の Java クラスとして利用されていることが分かります。このように、JVM COBOL を利用することで、COBOL を意識させること なく、Java プログラム内で利用することができます。



### 3.1.5. JUnit テストプログラムの実行

- 1) AirportDemoJUnit プロジェクトの配下の AircodeWrapperTest.java 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、[実行(R)] > [実行の構成(N)] ボタンをクリックします。

⊯ パッケージ・エクスプローラー 🛛		<b>□ \$ \$ ~ □</b>	AircodeWrapperTest.j	ava 🔀
<ul> <li>✓ (ご) AirportDemoJUnit</li> <li>&gt; ▲) JRE システム・ライブラリー [AdoptOpenJDK</li> <li>✓ ● src</li> <li>✓ ● com.sample</li> <li>&gt; ② AircodeWrapperTest.java</li> <li>&gt; ▲) JUnit 5</li> <li>&gt; ② AirportDemoJVMJUnit</li> </ul>		新規(W) 開く(O) アブリケーションから開く(H) 型階層を開く(N) 表示方法(W)	F3 F3 ト F4 Alt+シフト+W >	<pre>ple; rg.junit.Assert.assertEquals; t.Test; le.AircodeWrapper;</pre>
> 🔁 InternalTDProject		コピー(C) 修飾名のコピー(Y) 貼り付け(P) 削除(D)	Ctrl+C Ctrl+V 削除	<pre>rcodeWrapperTest {   testGetDistance() {   wrapper wrapper = new AircodeWrapper(;   initialize();   openFile();   tance = wrapper.getDistance("HND", "LH</pre>
	<u>\$</u> _	コンテキストから除去 ビルド・パス(B) ソース(S) リファクタリング(T)	Ctrl+Alt+シフト+下 > Alt+シフト+S> Alt+シフト+T>	<pre>guals(9591, distance); .closeFile();</pre>
	2	インボート(I) エクスポート(O) 参照(F) 宮言(I)	>	
	Ŷ	更新(F) ワーキング・セットの割り当て(A)	F5	
	Q.	Coverage As 実行(R)	>	四 1 サーバーで実行 Alt+シノフト+X R
	**	デバッグ(D) プロファイル(P) 検証(V) ローカル履歴から復元(Y)	>	2 AWS SAM Local Ju 3 JUnit テスト Alt+シフト+X,T 実行の構成(N)

2) 画面左側より「JUnit」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、[New Configuration] を選択します。





3) 「環境」タブを選択し、[追加(A)] ボタンをクリックします。

名前(N): AircodeW	rapperTest						
国 テスト (M= 引数 🕋 JRE (🍫 依存関係 💱 ソース 🔽 環境 🔃 共通(C)							
設定する環境変数(S	)						
変数	値	;追加(A)					
		選択(L)					
		編集(D)					
		削除(O)					

- 4) 以下の入力を行い、[OK] ボタンをクリックします。
  - 名前: "dd\_airports"
  - 值: "..¥AirportDemoJVMJUnit¥airports.dat"

🧐 環境変数の編集						
名前( <u>N</u> ):	dd_airports					
值( <u>V</u> ):	¥AirportDemoJVMJUnit¥airports.dat	変数( <u>B</u> )				
	OK	キャンセル				

5) dd\_airports 環境変数が追加されたことを確認して、[実行(R)] ボタンをクリックします。

💿 実行構成			— 🗆 X
構成の作成、管理、および JUnit テストを起動する構成を作成し	実 <del>行</del> <sup>ます。</sup>		
ご ● ◎ ● × ■	名前(N): AircodeWrapp	perTest 」JRE (%) クラスパス (5g/ ソース (電) 環境 (二) 共通(C)	
<ul> <li>Eclipse デノリリーンヨン</li> <li>Eclipse データ・ツール</li> <li>Gradle Project</li> <li>Grunt</li> <li>Gulp</li> <li>HTTP ブレビュー</li> <li>J2E ブレビュー</li> <li>J2E ブレビュー</li> <li>Java アブリケーション</li> <li>Java アブリット</li> <li>JUnit</li> <li>JUnit ブライン・テスト</li> <li>Mode js Application</li> <li>OSGi フレームワーク</li> <li>XSL</li> <li>Jug クスク・コンテキスト・テスト</li> <li>スBサーバー</li> <li>汎用サーバー(外部からの起調)</li> </ul>	設定する環境変数(S) 変数 ● dd_airports ● ネイティブ環境への環 ○ ネイティブ環境を指定	値 ¥AirportDemoJVMJUnit¥airports.dat 	新規(E) 選択(L) 編集(D) 育]除(O)
< > 30 項目のうち 30 項目がフィルターに-		前回保管した状態に戻	す(V) 適用(Y)
?		実行	(R) 閉じる

JUnit ビューが表示され、全てのテストが成功したことを示す緑色で表示されていることを確認します。

COBOL 開発: Eclipse - JVM COBOL プログラムの単体テスト



🛱 パッケージ・エク 🚽 JUnit 🛛 🗖 🗖	
00	
0.831 秒後に完了	
実行: 2/2 🛛 エラー: 0 🖾 失敗: 0	
✔ 🔚 AircodeWrapperTest [ランナー: JUnit 5] (0	
https://www.commonsciences Constructionasciencesciencesciencesciencesciencesciencesciencesciencesciencesciencesciencesciencesciencesciences	
॑ testGetDistance2 (0.004 秒)	

続いて、エラーケースを確認します。

6) JUnit ビューより testGetDistance1 をダブルクリックし、9591 という数値を 9592 に修正した上で保存します。



Junit ビューの「com.sample.AircodeWrapperTest」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、「実行(R)」を選択します。



COBOL 開発: Eclipse - JVM COBOL プログラムの単体テスト



JUnit ビューが赤色となり、さきほど数値を修正した testGetDistance1 が失敗していることが分かります。



#### 3.2. コマンドラインからの実行

JUnit テストプログラムはもちろん、JVM COBOL の開発・テストについても、コマンドラインから実施できます。従来のスタイルでの テスト作業の効率化を図ることができ、Jenkins などの CI ツールと連携する事で、テストの自動実行を行なえるため、品質担保 や作業工数の削減が見込めます。

ここでは、3.1 で作成したテストプログラムをコマンドラインから実行する方法について学びます。

1) Windows メニューより、Micro Focus Visual COBOL 配下の Visual COBOL コマンドプロンプト をクリックします。



2) 作業フォルダを作成し、作成したフォルダに移動します。

C:¥>mkdir VCCommandTutorial && cd VCCommandTutorial C:¥VCCommandTutorial>



- 3) 以下のコマンドを実行し、JVM COBOL, JUnit テストプログラムをコンパイルします。
  - 注意)

以下のコマンドで設定する環境変数は、お客様の環境に合わせて修正してください。 VC\_INSTALL\_PATH: Visual COBOL 製品のインストールフォルダ JUNIT\_JAR\_PATH: JUnit jar ファイルへの絶対パス ECLIPSE\_WORKSPACE: 3.1 で指定したワークスペースフォルダへの絶対パス

- set VC\_INSTALL\_PATH="c:¥Program Files (x86)¥Micro Focus¥Visual COBOL"
- set JUNIT\_JAR\_PATH="C:¥path-to-junit"
- set ECLIPSE\_WORKSPACE=C:\u00e4workspace\_tut\_jvm\_junit
- set

CLASSPATH=%VC\_INSTALL\_PATH%¥bin¥mfcobol.jar;%VC\_INSTALL\_PATH%¥bin¥mfcobolrts.j ar;%JUNIT\_JAR\_PATH%

- mkdir bin
- cobol %ECLIPSE\_WORKSPACE%¥AirportDemoJVMJUnit¥src¥aircode.cbl jvmgen(sub) iloutput(bin);
- cobol %ECLIPSE\_WORKSPACE%¥AirportDemoJVMJUnit¥src¥com¥sample¥AircodeWrapper.cbl
   jvmgen(sub) iloutput(bin);
- cd bin && jar cvf aircodejvm.jar \* && move aircodejvm.jar .. && cd ..
- rmdir /S /Q bin
- javac -d . -

cp %CLASSPATH%;aircodejvm.jar %ECLIPSE\_WORKSPACE%¥AirportDemoJUnit¥src¥com¥sa mple¥AircodeWrapperTest.java

C:¥VCCommandTutorial>set VC\_INSTALL\_PATH="c:¥Program Files (x86)¥Micro Focus¥Visual COBOL"

C:¥VCCommandTutorial>set JUNIT\_JAR\_PATH="D:¥jarlib¥poi-4.1.2¥lib¥junit-4.12.jar"

C:¥VCCommandTutorial>set

CLASSPATH=%VC\_INSTALL\_PATH%¥bin¥mfcobol.jar;%VC\_INSTALL\_PATH%¥bin¥mfcobolrt s.jar;%JUNIT\_JAR\_PATH%

C:¥VCCommandTutorial>mkdir bin

C:¥VCCommandTutorial>cobol %ECLIPSE\_WORKSPACE%¥AirportDemoJVMJUnit¥src¥aircode .cbl jvmgen(sub) iloutput(bin);

(出力内容省略)

C:¥VCCommandTutorial>cobol %ECLIPSE\_WORKSPACE%¥AirportDemoJVMJUnit¥src¥com¥sa mple¥AircodeWrapper.cbl jvmgen(sub) iloutput(bin);

(出力内容省略)

C:¥VCCommandTutorial>cd bin && jar cvf aircodejvm.jar \* && move aircodejvm.jar .. && cd ..

(出力内容省略)



C:¥VCCommandTutorial>rmdir /S /Q bin

C:¥VCCommandTutorial>javac -d . -

cp %CLASSPATH%;aircodejvm.jar %ECLIPSE\_WORKSPACE%¥AirportDemoJUnit¥

src¥com¥sample¥AircodeWrapperTest.java

C:¥VCCommandTutorial>

- 4) 以下のコマンドを実行し、JUnit テストを実行します。
  - set dd\_airports=%ECLIPSE\_WORKSPACE%¥AirportDemoJVMJUnit¥airports.dat
  - set JUNIT\_DEPENDENCY="C:¥Users¥Public¥Micro Focus¥Visual

COBOL¥eclipse¥plugins¥org.hamcrest.core\_1.3.0.v20180420-1519.jar"

 java -cp %CLASSPATH%;aircodejvm.jar;%JUNIT\_DEPENDENCY%;. org.junit.runner.JUnitCore com.sample.AircodeWrapperTest

#### C:¥VCCommandTutorial>set

dd\_airports=%ECLIPSE\_WORKSPACE%¥AirportDemoJVMJUnit¥airports.dat

C:¥VCCommandTutorial>set JUNIT\_DEPENDENCY="C:¥Users¥Public¥Micro Focus¥Visual COBOL¥eclipse¥plugins¥org.hamcrest.core\_1.3.0.v20180420-1519.jar"

C:¥VCCommandTutorial>java -cp %CLASSPATH%;aircodejvm.jar;%JUNIT\_DEPENDENCY%;. org.junit.runner.JUnitCore com.sample.AircodeWrapperTest

JUnit version 4.12

.HND Tokyo Intl	
Japan	Lat:+035.552258 Lon:+139.077969
LHR Heathrow	
United Kingdom	Lat:+051.004775 Lon:-000.461389
.NRT Narita Intl	
Japan	Lat:+035.764722 Lon:+140.038638
LHR Heathrow	
United Kingdom	Lat:+051.004775 Lon:-000.461389
Time: 0.187	
OK (2 tests)	

#### 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。