



可変長かつ複数セグメントの出力メッセージを処理するアプリケーションの作成

目次

可変長かつ複数のセグメントの IMS トランザクション用の J2C アプリケーションの作成	1
可変長かつ複数セグメントの出力メッセージを処理するアプリケーションの作成.	1
演習 1.1: リソース・アダプターを選択	3
演習 1.2: Web プロジェクトおよび Java インターフェースと実装のセットアップ	5

演習 1.3: メッセージ・バッファークラスの作成	6
演習 1.4: アプリケーションをテストするための Java テスト・クラス作成.	12
可変長および複数のセグメントが収められた IMS トランザクション用の J2C アプリケーションを作成する：要約	15

可変長かつ複数のセグメントの IMS トランザクション用の J2C アプリケーションの作成

このチュートリアルでは、「J2C Java™ Bean」ウィザードを使用して、可変長かつ複数セグメント入出力データを持つ IMS™ トランザクションの処理を行う、シンプルな Web アプリケーションを作成する方法を説明します。

学習目標

このチュートリアルでは以下のことを説明します。

- 「J2C Java Bean」ウィザードを使用して、IMS トランザクションを実行する J2C Java Bean を作成する。
- メッセージ・バッファ・クラスを作成し、ドックレット注釈を使用してこのクラスを編集する。
- J2C Java Bean のメソッドを作成し、IMS トランザクションの実行およびメソッドへの入出力データ型タイプの提供を行う。
- 出力メッセージのセグメント用 Java データ・バインディングを作成する。
- アプリケーションをテストするためのテスト用 Java クラスを作成する。

所要時間

30 分

関連情報



PDF バージョンを表示する

複数セグメント出力のサンプル

可変長かつ複数セグメントの出力メッセージを処理するアプリケーションの作成

このチュートリアルでは、可変長かつ複数セグメントの IMS トランザクション出力メッセージの処理を行う J2C アプリケーションを生成するための、詳細なステップを説明します。

このチュートリアルには、オプションでインストール可能なコンポーネントが必要な場合があります。システム要件を参照して、適切なオプション・コンポーネントがインストールされていることを確認してください。

このチュートリアルはいくつかの練習に分かれています。チュートリアルを正しく機能させるためには、演習を順序どおりに実行する必要があります。このチュートリアルでは、「J2C Java Bean」ウィザードを使用して、IMS でトランザクションを実行する Java Bean を作成する方法を学習します。練習では、次の作業を行います。

- 「J2C Java Bean」ウィザードを使用して、IMS トランザクションを実行する J2C Java Bean を作成する。
- メッセージ・バッファ・クラス CCIBuffer.java を作成し、ドックレット注釈を使用してこのクラスを編集する。

- J2C Java Bean のメソッドを作成し、IMS トランザクションの実行およびメソッドへの入出力データ型タイプの提供を行う。
- 出力メッセージのセグメント用Java データ・バインディングを作成する。
- テスト Java クラス `TestMultiSeg.java` を作成して、IMS トランザクションを実行する J2C Java Bean メソッドを起動し、IMS トランザクションが戻すデータのバッファから出力セグメントを取り込む。

注: テスト Java クラス `TestMultiSeg.java` は、英語地域用に作成されたので、その他の地域では、コードを変更する必要がある場合があります。

所要時間

このチュートリアルを終了するには、約 30 分かかります。このチュートリアルに関連した他の概念を検討する場合、完了するのにさらに長い時間がかかります。

スキル・レベル

上級

対象読者

このチュートリアルは、特にエンタープライズ情報システム (EIS) および IMS に詳しいユーザーを対象にしています。

システム要件

このチュートリアルを完了するには、以下のツールとコンポーネントがインストール済みである必要があります。

- IBM® WebSphere® Application Server バージョン 6.1
- J2EE コネクター (J2C) ツール
- **IMS 環境に関する情報:** このチュートリアルでは、あなたのアプリケーションは、IMS 内の IMS アプリケーション・プログラムと対話します。IMS 接続のホスト名およびポート番号や、トランザクションが実行する IMS データ・ストアの名前などの情報を取得する必要があります。この情報については、IMS システム管理者に連絡してください。具体的には、IMSMultiSegmentOutput IMS プログラムを実行する場合は、IMS でいくつかのセットアップ作業を実行する必要があります。
- **COBOL ファイル MSOut.cb1:** このファイルは、製品のインストール・ディレクトリー `radeclipsepluginscom.ibm.j2c.cheatsheet.content_6.0.0.samplesIMSMultiSegmentOutput` にあります。このファイルをローカルに保管する場合は、4 ページの『MSOut.cb1』からコードをコピーすることができます。
- クリーンなワークスペース

このチュートリアルを使用するには、アプリケーション・サーバーのインストールと構成が行なわれていることが必要です。サーバー・ランタイム環境が使用可能であるかどうかを検証するには、「ウィンドウ」→「設定」をクリックして、「サーバー」を展開し、「インストール済みランタイム」をクリックします。このペインを使用して、インストール済みサーバー・ランタイムの定義を追加、除去、または編集することができます。また、新規サーバーのサポートをダウンロードしてインストールすることもできます。

前提条件

このチュートリアルの内容すべてを実行するためには、次の知識が必要です。

- J2EE および Java プログラミング
- 基本的な IMS Transaction Manager (IMS TM) の概念

2 可変長かつ複数セグメントの出力メッセージを処理するアプリケーションの作成

演習 1.1: リソース・アダプターの選択

この演習では、リソース・アダプターを選択して、IMS サーバーに接続するように構成するための、詳細なステップについて説明します。

このチュートリアルで使用する IMS トランザクションは、IMS インストール検査プログラムではありません。このチュートリアルでは、制御ステートメントの情報に基づいて IMS への呼び出しを発行する IMS アプリケーション・プログラムである、DFSDDLTO が使用されます。このチュートリアル用の DFSDDLTO 制御ステートメントを、以下に示します。ただし、このチュートリアルを実行するためには、ご使用の環境を DFSDDLTO に合わせて構成し、また、必要な JCL を提供する必要があります。このチュートリアルでは、SKS2 を DFSDDLTO アプリケーションのトランザクション・コードとして使用します。

DFSDDLTO 制御ステートメント

```
S11 1 1 1 1 TP 1
L GU
E OK
E Z0017 DATA SKS2 M2 SI1M3 SI1
WTO SEGMENT SI1 RECEIVED
L GN
E QD
WTO END OF INPUT SEGMENTS
L ISRT IW06OUT
L Z0012 DATA *****M1S01
E OK
WTO SEGMENT S01 INSERTTED
L ISRT
L Z0027 DATA *****M1S02*****M2S02
E OK
WTO SEGMENT S02 INSERTTED
L ISRT
L Z0048 DATA *****M1S03*****M2S03*****M3S03
E OK
WTO SEGMENT S03 INSERTTED
WTO CURRENT PROGRAM STLDDLTO2 TERMINATED
L GU
```


このチュートリアルでは、COBOL データ構造を使用して、IMS トランザクションの入力メッセージおよび出力メッセージについて説明します。IMS から戻される出力メッセージは、次の 3 つの固定長セグメントで構成されることに注意してください。

- OUTPUT-SEG1 (16 バイト)
- OUTPUT-SEG2 (31 バイト)
- OUTPUT-SEG3 (52 バイト)

この特定の IMS アプリケーションによって戻される出力メッセージは 99 バイトの固定サイズであり、COBOL 01 構造 OUTPUT-MSG で表記されます。

この複数セグメント・アプリケーションを開発する 1 つの方法としては、COBOL 定義 OUTPUT-MSG を使用してトランザクションの出力を定義する方法が挙げられます。2 番目の方法は、トランザクションの出力のための出力メッセージを作成することです。このチュートリアルで提供するコードでは、2 番目の方法を使用します。これは、この方法が可変長出力メッセージを処理するアプリケーションの作成にも使用できるためです。個々のメッセージ・セグメント用の COBOL 定義は、個々のセグメントのデータへのアクセスを単純化するために、引き続き使用します。

IMS サーバーへの接続

1. J2EE アイコン () がワークスペースの右上部タブに表示されない場合は、J2EE パースペクティブに切り替える必要があります。メニュー・バーから、「ウィンドウ」>「パースペクティブを開く」>「その他」と選択します。「パースペクティブの選択」ウィンドウが開きます。
2. 「J2EE」を選択します。
3. 「OK」をクリックします。J2EE パースペクティブが開きます。
4. J2EE パースペクティブで、「ファイル」>「新規」>「その他」と選択します。
5. 「新規」ページで、「J2C」>「J2C Java Bean」と選択します。「次へ」をクリックします。
6. 「リソース・アダプターの選択」ページで、J2C 1.5 IMS リソース・アダプターを選択します。このチュートリアルでは、「IMS Connector for Java (IBM: 9.1.0.2.3)」を選択します。「次へ」をクリックします。
7. 「接続プロパティ」ページで、「管理接続」をクリアし、「非管理接続」を選択します。(このチュートリアルでは、非管理対象の接続を使用して、IMS に直接アクセスします。)デフォルトの接続クラス名 `com.ibm.connector2.ims.ico.IMSManagedConnectionFactory` を受け入れます。ブランク・フィールドに、必要なすべての接続情報を入力してください。必須フィールドは以下のとおりです。これらはアスタリスク (*) で表されています。
 - a. TCP/IP 接続用:
 - 1) ホスト名: (必須) IMS 接続の IP アドレスまたはホスト名。
 - 2) ポート番号: (必須) ターゲット IMS 接続で使用するポートの番号。
 - b. ローカルのオプション接続用:
 - 1) IMS 接続名: (必須) ターゲット IMS 接続の名前。
 - c. 両方の接続用:
 - 1) データ・ストア名: (必須) ターゲット IMS データ・ストアの名前。
8. これらの接続情報は、担当の IMS システム管理者から得ることができます。必要な接続情報を指定したら、「次へ」をクリックしてください。

MSout.cbl

MSout.cbl のコードを次に示します。

MSout.cbl

```
IDENTIFICATION DIVISION.
program-id. pgml.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
DATA DIVISION.
*
*      IMS TOC Connector for Java, Multi-segment Output Example
*
*****/
*                                                                    */
* (c) Copyright IBM Corp. 1998                                     */
* All Rights Reserved                                              */
* Licensed Materials - Property of IBM                             */
*                                                                    */
* DISCLAIMER OF WARRANTIES.                                       */
*                                                                    */
* The following (enclosed) code is provided to you solely for the */
* purpose of assisting you in the development of your applications.*/
* The code is provided "AS IS." IBM MAKES NO WARRANTIES, EXPRESS OR */
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF */
* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING */
```



```

* THE FUNCTION OR PERFORMANCE OF THIS CODE. */
* IBM shall not be liable for any damages arising out of your use */
* of the generated code, even if they have been advised of the */
* possibility of such damages. */
* */
* DISTRIBUTION. */
* */
* This generated code can be freely distributed, copied, altered, */
* and incorporated into other software, provided that: */
* - It bears the above Copyright notice and DISCLAIMER intact */
* - The software is not for resale */
* */
*****/
*
LINKAGE SECTION.

01 INPUT-MSG.
02 IN-LL          PICTURE S9(3) COMP.
02 IN-ZZ          PICTURE S9(3) COMP.
02 IN-TRCD        PICTURE X(5).
02 IN-DATA1        PICTURE X(6).
02 IN-DATA2        PICTURE X(6).

01 OUTPUT-MSG.
02 OUT-ALLSEGS    PICTURE X(99) VALUE SPACES.

01 OUTPUT-SEG1.
02 OUT-LL          PICTURE S9(3) COMP VALUE +0.
02 OUT-ZZ          PICTURE S9(3) COMP VALUE +0.
02 OUT-DATA1        PICTURE X(12) VALUE SPACES.

01 OUTPUT-SEG2.
02 OUT-LL          PICTURE S9(3) COMP VALUE +0.
02 OUT-ZZ          PICTURE S9(3) COMP VALUE +0.
02 OUT-DATA1        PICTURE X(13) VALUE SPACES.
02 OUT-DATA2        PICTURE X(14) VALUE SPACES.

01 OUTPUT-SEG3.
02 OUT-LL          PICTURE S9(3) COMP VALUE +0.
02 OUT-ZZ          PICTURE S9(3) COMP VALUE +0.
02 OUT-DATA1        PICTURE X(15) VALUE SPACES.
02 OUT-DATA2        PICTURE X(16) VALUE SPACES.
02 OUT-DATA3        PICTURE X(17) VALUE SPACES.

PROCEDURE DIVISION.

```

演習 1.2: Web プロジェクトおよび Java インターフェースと実装のセットアップ

演習 1.2 では、Web プロジェクトおよび Java インターフェースと実装の作成を段階的に学習します。

開始する前に、演習 1.1 を完了しておく必要があります。この演習では、次のことを行います。

- J2C Java Bean を作成する
 - 動的 Web プロジェクトを作成する
1. ワークベンチを使用して行うすべての作業は、プロジェクトに関連付ける必要があります。プロジェクトでは、プロジェクトのタイプに基づいた機能により最適化された、作業ファイルとディレクトリーの系統的なビューが提供されます。ワークベンチでは、すべてのファイルがプロジェクト内になければならないため、J2C Java Bean を作成する前に、ファイルを入れるプロジェクトを作成しておく必要があります。「新規 J2C Java Bean 」ページで、「プロジェクト名」フィールドに、値 MultiSegOutput を入力します。

2. 「プロジェクト名」フィールドの横にある「新規」ボタンをクリックして、新規プロジェクトを作成します。
3. 次に、動的 Web プロジェクトを作成します。「新規ソース・プロジェクトの作成」ページで、「Web プロジェクト」を選択し、「次へ」をクリックします。
4. 「新規動的 Web プロジェクト」ページで、「詳細を表示」をクリックします。
5. 次の値が選択されていることを確認します。
 - a. プロジェクト名: MultiSegOutput
 - b. プロジェクト・コンテンツ: デフォルトの使用
 - c. ターゲット・ランタイム: WebSphere Application Server v6.1
 - d. 構成: デフォルトを受け入れる
 - e. EAR にプロジェクトを追加: チェック
 - f. EAR プロジェクト名: MultiSegOutputEAR
6. 「次へ」をクリックします。
7. 「プロジェクト・ファセット」ページで、デフォルト設定を受け入れます。
8. 「次へ」をクリックします。
9. 「Web モジュール」ページで、デフォルト設定を受け入れます。
10. 「終了」をクリックします。
11. 動的 Web パースペクティブに切り替えるかどうかを尋ねるダイアログ・ボックスが表示されます。「はい」をクリックします。
12. 「J2C Java Bean 出力プロパティ」ページで、以下を実行します。
 - a. 「プロジェクト名」フィールドで、「参照」をクリックして「MultiSegOutput」プロジェクトを選択します。「OK」をクリックします。
 - b. 「パッケージ名」フィールドに sample.ims と入力します。
 - c. 「インターフェース名」フィールドに MS0 と入力します。
 - d. 「実装名」フィールドに MS0Impl と入力します。
 - e. 「Ant スクリプトとしてセッションを保管」を未チェックのままにします。
13. 「終了」をクリックします。

演習 1.3: メッセージ・バッファ・クラスの作成

演習 1.3 では、メッセージ・バッファ・クラスの作成を学習します。

この演習を始める前に、「演習 1.2: Web プロジェクトおよび Java インターフェースと Java 実装のセットアップ」を完了しておく必要があります。この演習では、次のことを行います。

- メッセージ・バッファ・クラスを作成する
 - ドックレット注釈を使用してメッセージ・バッファ・クラスを編集する
 - 入出力バインディング操作を作成する
 - 出力セグメント・データ・マッピングを作成する
1. 最初にメッセージ・バッファ・クラスを作成します。 **MultiSegOutput** プロジェクトを展開し、「Java リソース」を展開してから、「Java ソース」を展開します。
 2. **sample.ims** パッケージを右クリックし、「新規」>「クラス」を選択して、「新規クラス」ウィザードを起動します。

3. クラスの名前として **CCIBuffer** を入力します。すべてのデフォルト設定を受け入れます。
4. 「終了」をクリックします。Java エディターで **CCIBuffer** クラスが開きます。
5. **CCIBuffer** クラスのコメント・セクションに、タグ `@type-descriptor.message-buffer` を入力します。
6. **Ctrl+S** キーを押して、変更を保管します。新規コードは 10 ページの『**CCIBuffer.java**』に自動生成されることに注意してください。
7. 次に、**IMS** トランザクションを実行するメソッドと、入力メッセージ・データ・タイプを作成します。「プロジェクト・エクスプローラー」ビューで、**MSOImpl.java** を右クリックし、「ソース」>「**J2C Java Bean へのメソッドの追加**」を選択します。
8. 「新規 Java メソッド」ページで、「追加」をクリックします。
9. Java メソッド名として **runMultiSegOutput** を入力します。「次へ」をクリックします。
10. 入力タイプを定義するために、「新規」をクリックします。
11. 「**COBOL_to_Java**」マッピングを選択します。「参照」をクリックします。
12. COBOL ファイル **MSOut.cbl** を探して選択します。「開く」をクリックします。
13. 「次へ」をクリックします。
14. 「COBOL インポーター」ページで、「詳細を表示」をクリックします。
 - a. 次のオプションを選択してください。

表 1. COBOL インポーターのパラメーター設定値

パラメーター	値
プラットフォーム名 (Platform Name)	z/OS
コード・ページ	IBM-037
浮動小数点形式	IBM 16 進数
外部 10 進符号	EBCDIC
エンディアン名	Big
リモート整数エンディアン名	Big
引用符名	DOUBLE
TRUNC 名	STD
Nsymbol 名	DBCS

- b. 「照会」をクリックして、データをロードします。
 - c. データ構造のリストが表示されます。「データ構造 (Data structures)」フィールドで **INPUT-MSG** を選択します。
 - d. 「次へ」をクリックします。
 - e. 「**Ant スクリプトとしてセッションを保管**」を未チェックのままにします。
15. 「保管するプロパティ」ページで、デフォルトのクラス名「**INPUTMSG**」を、「**InputMsg**」で上書きします。「終了」をクリックします。
16. 次に、出力メッセージ・データ・タイプを作成します。出力タイプを定義するために、「参照」をクリックします。
17. 「データ・タイプの選択」フィールドに **CC** と入力します。「対応する型」フィールドに「**CCIBuffer**」が表示されます。出力タイプとして「**CCIBuffer**」を選択します。「終了」をクリックします。
18. 「Java メソッド」ページで、「完了」をクリックします。

19. 「Java メソッド」ページで、**InteractionVerb** が **SYNC_SEND_RECEIVE(1)** に設定されていて、送信に続いて受信を実行するよう、IMS との対話が指定されていることを確認します。
20. 「終了」をクリックします。
21. 次に、出力セグメント・データ・マッピングを作成します。まず、**OutputSeg1.java** クラスを作成します。このステップを実行するには、データ・マッピング・ファイルのみを作成できるよう、独立したデータ・マッピング・ウィザードを使用する必要があります。
22. 「ファイル > 新規 > その他 > CICS/IMS Java データ・バインディング」を選択して、「データ・バインディング」ウィザードを起動します。
23. 「次へ」をクリックします。
24. 「マッピングの選択」リストで「**COBOL から Java**」を選択します。「参照」をクリックして、COBOL コピーブック **MSOut.cbl** 見つけます。
25. 「次へ」をクリックします。
26. 「COBOL インポーター」ページで、「詳細を表示」をクリックします。
 - a. 次のオプションを選択してください。

表 2. COBOL インポーターのパラメーター設定値

パラメーター	値
プラットフォーム名 (Platform Name)	z/OS
コード・ページ	IBM-037
浮動小数点形式	IBM 16 進数
外部 10 進符号	EBCDIC
エンディアン名	Big
リモート整数エンディアン名	Big
引用符名	DOUBLE
TRUNC 名	STD
Nsymbol 名	DBCS

- b. 「照会」をクリックして、データをロードします。
 - c. データ構造のリストが表示されます。「データ構造」フィールドで **OUTPUT-SEG1** を選択します。
 - d. 「次へ」をクリックします。
27. 「保管するプロパティ」ウィザードで、「参照」をクリックして、先ほど作成した **MultiSegOutput** プロジェクトを選択します。
28. 「参照」をクリックして、パッケージ名 **sample.ims.data** を選択します。
29. 「Java クラス名」を **OUTPUTSEG1** から **OutputSeg1** に変更します。
30. 「終了」をクリックします。
31. それでは、**OutputSeg2.java** クラスを作成します。「ファイル > 新規 > その他 > CICS/IMS Java データ・バインディング」を選択して、「データ・バインディング」ウィザードを起動します。
32. 「次へ」をクリックします。
33. 「マッピングの選択」リストで「**COBOL から Java**」を選択します。「参照」をクリックして、COBOL コピーブック **MSOut.cbl** 見つけます。
34. 「COBOL インポーター」ページで、「詳細を表示」をクリックします。

- a. 次のオプションを選択してください。

表 3. COBOL インポーターのパラメーター設定値

パラメーター	値
プラットフォーム名 (Platform Name)	z/OS
コード・ページ	IBM-037
浮動小数点形式	IBM 16 進数
外部 10 進符号	EBCDIC
エンディアン名	Big
リモート整数エンディアン名	Big
引用符名	DOUBLE
TRUNC 名	STD
Nsymbol 名	DBCS

- b. 「照会」をクリックして、データをロードします。
- c. データ構造のリストが表示されます。「データ構造」フィールドで **OUTPUT-SEG2** を選択します。
- d. 「次へ」をクリックします。
35. 「保管するプロパティ」ウィザードで、「ブラウズ」をクリックして、先ほど作成した **MultiSegOutput** プロジェクトを選択します。
36. 「参照」をクリックして、パッケージ名 **sample.ims.data** を選択します。
37. Java クラス名を **OUTPUTSEG2** から **OutputSeg2** に変更します。
38. 「終了」をクリックします。
39. それでは、**OutputSeg3.java** クラスを作成しましょう。「ファイル > 新規 > その他 > CICS/IMS Java データ・バインディング」を選択して、「データ・バインディング」ウィザードを起動します。
40. 「次へ」をクリックします。
41. 「マッピングの選択」リストで「**COBOL から Java**」を選択します。「参照」をクリックして、COBOL コピーブック **MSOut.cbl** 見つけます。
42. 「COBOL インポーター」ページで、「詳細を表示」をクリックします。
- a. 次のオプションを選択してください。

表 4. COBOL インポーターのパラメーター設定値

パラメーター	値
プラットフォーム名 (Platform Name)	z/OS
コード・ページ	037
浮動小数点形式	IBM 390 16 進数
外部 10 進符号	EBCDIC
エンディアン名	Big
リモート整数エンディアン名	Big
引用符名	DOUBLE
TRUNC 名	STD
Nsymbol 名	DBCS

- b. 「照会」をクリックして、データをロードします。

- c. データ構造のリストが表示されます。「データ構造」フィールドで **OUTPUT-SEG3** を選択します。
 - d. 「次へ」をクリックします。
43. 「保管するプロパティ (Saving properties)」ウィザードで、「ブラウズ」をクリックして、先ほど作成した **MultiSegOutput** プロジェクトを選択します。
 44. 「参照」をクリックして、パッケージ名 **sample.ims.data** を選択します。
 45. 「Java クラス名」を **OUTPUTSEG3** から **OutputSeg3** に変更します。
 46. 「終了」をクリックします。

CCIBuffer.java

CCIBuffer.java クラス内の生成済みコードを次に示します。

CCIBuffer.java

```
/*
 * Created on Oct 13, 2004
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package sample.ims;

/**
 * @author ivyho
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 * @type-descriptor.message-buffer
 */
public class CCIBuffer implements javax.resource.cci.Record,
    javax.resource.cci.Streamable, com.ibm.etools.marshall.RecordBytes {

    private byte[] buffer_ = null;

    /**
     * @generated
     */
    public CCIBuffer() {
        return;
    }

    /**
     * @generated
     * @see javax.resource.cci.Record#getRecordShortDescription()
     */
    public String getRecordShortDescription() {
        return (this.getClass().getName());
    }

    /**
     * @generated
     * @see javax.resource.cci.Record#hashCode()
     */
    public int hashCode() {
        return (super.hashCode());
    }

    /**
     * @generated
     * @see javax.resource.cci.Streamable#write(OutputStream)
     */
}
```

```

public void write(java.io.OutputStream outputStream)
    throws java.io.IOException {
    outputStream.write(buffer_);
}

/**
 * @generated
 * @see javax.resource.cci.Record#setRecordShortDescription(String)
 */
public void setRecordShortDescription(String shortDescription) {
    return;
}

/**
 * @generated
 */
public int getSize() {
    if (buffer_ != null)
        return (buffer_.length);
    else
        return (0);
}

/**
 * @generated
 * @see java.lang.Object#toString
 */
public String toString() {
    StringBuffer sb = new StringBuffer(super.toString());
    sb.append("\n");
    com.ibm.etools.marshall.util.ConversionUtils.dumpBytes(sb, buffer_);
    return (sb.toString());
}

/**
 * @generated
 * @see javax.resource.cci.Record#getRecordName()
 */
public String getRecordName() {
    return (this.getClass().getName());
}

/**
 * @generated
 */
public byte[] getBytes() {
    return (buffer_);
}

/**
 * @generated
 * @see javax.resource.cci.Record#clone()
 */
public Object clone() throws CloneNotSupportedException {
    return (super.clone());
}

/**
 * @generated
 * @see javax.resource.cci.Record#setRecordName(String)
 */
public void setRecordName(String recordName) {
    return;
}

/**
 * @generated

```

```

    * @see javax.resource.cci.Record#equals()
    */
    public boolean equals(Object object) {
        return (super.equals(object));
    }

    /**
     * @generated
     * @see javax.resource.cci.Streamable#read(InputStream)
     */
    public void read(java.io.InputStream inputStream)
        throws java.io.IOException {
        byte[] input = new byte[inputStream.available()];
        inputStream.read(input);
        buffer_ = input;
    }


    /**
     * @generated
     */
    public void setBytes(byte[] bytes) {
        buffer_ = bytes;
    }
}

```

演習 1.4: アプリケーションをテストするための Java テスト・クラスの実成

演習 1.4 では、アプリケーションのテストを行うための Java テスト・クラスの実成を学習します。

この演習を始める前に、「演習 1.3: メッセージ・バッファ・クラスの実成」を完了しておく必要があります。この演習では、次のことを行います。

- Java テスト・クラスを実成する。
 - 以下に示したコードを使用してクラスを編集する。
 - テスト・クラスを実行してアプリケーションをテストする。
1. まず、**Java テスト・クラスを実成します**。MultiSegOutput プロジェクトを展開し、「**Java リソース**」を展開してから、**sample.ims** パッケージを選択します。
 2. 右マウス・ボタンをクリックして、「**新規**」を選択します。  クラス・オプションを選択して、新規 Java クラスを実成します。
 3. 「Java クラス名」フィールドに、TestMultiSeg と入力します。TestMultiSeg.java クラスは、例示用としてのみ提供されている点に注意してください。ご使用の IMS マシンの仕様に合わせて、トランザクション・コードを変更する必要がある場合があります。トランザクション・コードについては、IMS 管理者にお尋ねください。TestMultiSeg.java クラス内にあるステートメント `input.setIn_trcd("SKS6 ")` を探して変更を加えることができます。
 4. 「ソース・フォルダー」に **MultiSegOutput/JavaSource** が、「パッケージ名」に **sample.ims** が含まれていることを確認してください。
 5. 「終了」をクリックします。
 6. Java クラス・エディターで、**TestMultiSeg.java** を開きます。
 7. エディター内のコードをすべて、次のものに置き換えてください。

```

/*****
 * Licensed Materials - Property of IBM
 *

```



```

* com.ibm.j2c.cheatsheet.content
*
*Copyright IBM Corporation 2004. All Rights Reserved.    *
* Note to U.S. Government Users Restricted Rights: Use,
* duplication or disclosure restricted by GSA ADP Schedule
* Contract with IBM Corp.
*****/
package sample.ims;

import com.ibm.etoools.marshall.util.MarshallIntegerUtils;
import sample.ims.data.*;

public class TestMultiSeg
{
    public static void main (String[] args)
    {
        byte[] segBytes = null;
        int srcPos = 0;
        int dstPos = 0;
        int totalLen = 0;
        int remainLen = 0;
        byte[] buff;
        short LL = 0;
        short ZZ = 0;

        try
        {
            // -----
            // Populate the IMS transaction input message with
            // data. Use the input message format handler method
            // getSize() to set the LL field of the input message.
            // -----
            InputMsg input = new InputMsg();
            input.setIn_ll((short) input.getSize());
            input.setIn_zz((short) 0);
            //-----
            // find out the transaction code from your IMS
            // administrator
            //-----
            input.setIn_trcd("SKS6 ");
            input.setIn_data1("M2 S11");
            input.setIn_data2("M3 S11");

            // -----
            // Run the IMS transaction. The multi-segment output
            // message is returned.
            // -----
            MSOImpl proxy = new MSOImpl();

            sample.ims.CCIBuffer output = proxy.runMultiSegOutput(input);

            // -----
            // Retrieve the multi-segment output message as a
            // byte array using the output message format
            // handler method getBytes().
            // -----
            System.out.println(
                "\nSize of output message is: " + output.getSize());
            segBytes = output.getBytes();

            srcPos = 0;
            dstPos = 0;
            totalLen = segBytes.length;
            remainLen = totalLen;

            // -----
            // Populate first segment object from buffer.

```

```

// -----
buff = null;
// Get length of segment.
LL =
    MarshallIntegerUtils.unmarshallTwoByteIntegerFromBuffer(
        segBytes,
        srcPos,
        true,
        MarshallIntegerUtils.SIGN_CODING_TWOS_COMPLEMENT);

// Put segment in byte array.
buff = new byte[LL];
System.arraycopy(segBytes, srcPos, buff, dstPos, LL);
remainLen -= LL;

// Create and populate segment object from byte array.
OutputSeg1 S1 = new OutputSeg1();
S1.setBytes(buff);
System.out.println(
    "\nOutSeg1 LL is:    "
    + S1.getOut__ll()
    + "\nOutSeg1 ZZ is:    "
    + S1.getOut__zz()
    + "\nOutSeg1_DATA1 is: "
    + S1.getOut__data1());

// -----
// Populate second segment object from buffer.
// -----
srcPos += LL;
buff = null;
// Get length of segment.
LL =
    MarshallIntegerUtils.unmarshallTwoByteIntegerFromBuffer(
        segBytes,
        srcPos,
        true,
        MarshallIntegerUtils.SIGN_CODING_TWOS_COMPLEMENT);

// Put segment in byte array.
buff = new byte[LL];
System.arraycopy(segBytes, srcPos, buff, dstPos, LL);
remainLen -= LL;

// Create and populate segment object from byte array.

OutputSeg2 S2 = new OutputSeg2();
S2.setBytes(buff);
System.out.println(
    "\nOutSeg2 LL is:    "
    + S2.getOut__ll()
    + "\nOutSeg2 ZZ is:    "
    + S2.getOut__zz()
    + "\nOutSeg2_DATA1 is: "
    + S2.getOut__data1()
    + "\nOutSeg2_DATA2 is: "
    + S2.getOut__data2());
// -----
// Populate third segment object from buffer.
// -----
srcPos += LL;
buff = null;
// Get length of segment.
LL =
    MarshallIntegerUtils.unmarshallTwoByteIntegerFromBuffer(
        segBytes,
        srcPos,

```

```

        true,
        MarshallIntegerUtils.SIGN_CODING_TWOS_COMPLEMENT);

// Put segment in byte array.
buff = new byte[LL];
System.arraycopy(segBytes, srcPos, buff, dstPos, LL);
remainLen -= LL;

// Create and populate segment object from byte array.
OutputSeg3 S3 = new OutputSeg3();
S3.setBytes(buff);
System.out.println(
    "\nOutSeg3 LL is:      "
    + S3.getOut_ll()
    + "\nOutSeg3 ZZ is:      "
    + S3.getOut_zz()
    + "\nOutSeg3_DATA1 is: "
    + S3.getOut_data1()
    + "\nOutSeg3_DATA2 is: "
    + S3.getOut_data2()
    + "\nOutSeg3_DATA3 is: "
    + S3.getOut_data3());
}
catch (Exception e)
{
    System.out.println("\nCaught exception is: " + e);
}
}
}

```

8. それではアプリケーションをテストします。 **MultiSegOutput** プロジェクトと **sample.ims** パッケージを展開します。
9. TestMultiSeg.java クラスを右クリックし、「実行」を選択します。「実行」>「Java アプリケーション」を選択します。
10. コンソールに以下のような出力が表示されれば、テストは成功です。

```

Size of output message is:      99
OutSeg1 LL is:                  16
OutSeg1 ZZ is:                  768
OutSeg1_DATA1 is:               *****M1S01

OutSeg2 LL is:                  31
OutSeg2 ZZ is:                  768
OutSeg2_DATA1 is:               *****M1S02
OutSeg2_DATA2 is:               *****M2S02

OutSeg3 LL is:                  52
OutSeg3 ZZ is:                  768
OutSeg3_DATA1 is:               *****M1S03
OutSeg3_DATA2 is:               *****M2S03
OutSeg3_DATA3 is:               *****M3S03

```

11. おつかれさまでした。これで複数セグメント出力のチュートリアルは完了です。

可変長および複数のセグメントが収められた IMS トランザクション用の J2C アプリケーションを作成する：要約

このチュートリアルでは、可変長および複数セグメントの IMS トランザクション出力メッセージの処理を行う J2C アプリケーションを生成するための、詳細なステップを説明しました。

演習での学習事項

このチュートリアルでは、以下の内容を学習しました。

- 「J2C Java Bean」ウィザードを使用して、IMS トランザクションを実行する J2C Java Bean を作成する。
- メッセージ・バッファークラス `CCIBuffer.java` を作成し、ドックレット注釈を使用してこのクラスを編集する。
- J2C Java Bean のメソッドを作成し、IMS トランザクションの実行およびメソッドへの入出力データ型タイプの提供を行う。
- 出力メッセージのセグメント用Java データ・バインディングを作成する。
- テスト Java クラス `TestMultiSeg.java` を作成して、IMS トランザクションを実行する J2C Java Bean メソッドを起動し、IMS トランザクションが戻すデータのバッファークラスから出力セグメントを取り込む。