



Crear una aplicación para procesar mensajes de salida de longitud variable y de varios segmentos



---

## Contenido

### **Crear una aplicación J2C para una transacción IMS que contenga longitud variable y varios segmentos . . . . . 1**

Crear una aplicación para procesar mensajes de salida de longitud variable y de varios segmentos . . . . .	1
Lección 1.1: Seleccionar el adaptador de recurso . . . . .	2
Lección 1.2: Configurar un proyecto Web y la interfaz e implementaciones Java . . . . .	5

Lección 1.3: Crear una clase de almacenamiento intermedio de mensaje . . . . .	6
Lección 1.4: Crear una clase proxy Java para probar la aplicación . . . . .	11
Resumen de la creación de una aplicación J2C para una transacción IMS que contenga longitud variable y varios segmentos . . . . .	15



---

## Crear una aplicación J2C para una transacción IMS que contenga longitud variable y varios segmentos

En esta guía de aprendizaje se describe cómo utilizar el Asistente J2C Java Bean para construir una aplicación Web sencilla que procese una transacción IMS con datos de entrada y salida de longitud variable y con múltiples segmentos.

### Objetivos educativos

Esta guía de aprendizaje le permite:

- Utilizar el asistente Bean Java J2C para crear un bean Java J2C que ejecute una transacción IMS.
- Crear una clase de almacenamiento intermedio de mensaje y editar esta clase utilizando anotaciones doclet.
- Crear un método para que el bean Java J2C ejecute la transacción IMS y proporcionar tipos de datos de entrada y salida para el método.
- Crear enlaces de datos Java para los segmentos del mensaje de salida.
- Crear una clase de Java de prueba para probar la aplicación

### Tiempo necesario

30 minutos

#### Información relacionada



Ver la versión PDF

Ejemplo de salida de varios segmentos

---

## Crear una aplicación para procesar mensajes de salida de longitud variable y de varios segmentos

En esta guía de aprendizaje recorrerá todos los pasos detallados para generar una aplicación J2C que procesa mensajes de salida de transacción IMS de longitud variable y con múltiples segmentos.

Esta guía de aprendizaje puede necesitar algunos componentes instalables opcionales. Consulte la lista de requisitos del sistema para asegurarse de haber instalado los componentes opcionales adecuados.

Esta guía de aprendizaje se divide en varios ejercicios que deben realizarse siguiendo el orden indicado para cumplir los objetivos marcados. En esta guía de aprendizaje aprenderá a utilizar el asistente Bean Java J2C para crear un bean Java que ejecute una transacción en IMS. Mientras realiza los ejercicios podrá:

- Utilizar el asistente Bean Java J2C para crear un bean Java J2C que ejecute una transacción IMS.
- Crear una clase de almacenamiento intermedio de mensaje, CCIBuffer.java, y editar esta clase utilizando anotaciones doclet.
- Crear un método para que el bean Java J2C ejecute la transacción IMS y proporcionar tipos de datos de entrada y salida para el método.
- Crear enlaces de datos Java para los segmentos del mensaje de salida.
- Crear una clase de prueba Java, TestMultiSeg.java, para invocar el método de bean Java J2C que ejecuta la transacción IMS y poblar los segmentos de salida desde el almacenamiento intermedio de los datos devueltos por la transacción IMS.

**Nota:** La clase Java de prueba *TestMultiSeg.java*, se creó para un entorno local inglés; deberá modificar el código para otros entornos locales.

## Tiempo necesario

La consecución de esta guía de aprendizaje debe tardar aproximadamente 30 minutos. Si explora otros conceptos relacionados con esta guía de aprendizaje, puede tardar más en completarla.

## Nivel de habilidad

Experimentado

## Público

Esta guía de aprendizaje está pensada para los usuarios familiarizados con sistemas de información de empresa (EIS) y con IMS en concreto.

## Requisitos del sistema

Para completar esta guía de aprendizaje, necesita tener instalados los componentes y las herramientas siguientes:

- IBM WebSphere Application Server, versión 6.1
- Herramientas J2EE Connector (J2C)
- **Información acerca del entorno IMS:** en esta guía de aprendizaje, la aplicación interactúa con un programa de aplicación IMS en IMS. Debe obtener información como por ejemplo el nombre de sistema principal y el número de puerto de IMS Connect y el nombre del almacén de datos de IMS en el que se ejecutará la transacción. Póngase en contacto con el administrador del sistema IMS para obtener esta información. Concretamente, deberá realizar ciertas tareas de configuración en IMS si desea ejecutar el programa IMS IMS\MultiSegmentOutput.
- **Una copia del archivo COBOL MSOut.cbl:** encontrará este archivo en el directorio de instalación del producto \rad\eclipse\plugins\com.ibm.j2c.cheatsheet.content\_7.0.0\samples\IMS\MultiSegmentOutput. Si desea almacenarlo localmente, puede copiar el código de aquí: “MSOut.cbl” en la página 4
- **Un espacio de trabajo limpio.**

Para utilizar esta guía de aprendizaje, debe tener un servidor de aplicaciones instalado y configurado. Para verificar que tiene disponible un entorno de tiempo de ejecución de servidor, pulse **Ventana → Preferencias**, expanda **Servidor** y pulse **Entornos de tiempo de ejecución instalados**. Puede utilizar este panel para añadir, eliminar o editar definiciones de tiempo de ejecución de servidor instaladas. También puede descargar e instalar soporte para un servidor nuevo.

## Prerrequisitos

Para poder realizar esta guía de aprendizaje de principio a fin, debe estar familiarizado con:

- Programación J2EE y Java
- Conceptos básicos de IMS Transaction Manager (IMS TM)

---

## Lección 1.1: Seleccionar el adaptador de recurso

En esta lección seguirá los pasos detallados para seleccionar el adaptador de recursos y configurarlo para la conexión con el servidor IMS.

La transacción IMS que se utiliza en esta guía de aprendizaje no es uno de los programas de verificación de instalación de IMS. Esta guía de aprendizaje utiliza DFSDDL0, un programa de aplicación IMS que

emite llamadas a IMS basándose en información de sentencias de control. A continuación encontrará las sentencias de control DFSDDLTO de esta guía de aprendizaje. No obstante, debe configurar su entorno para DFSDDLTO y proporcionar el JCL necesario. En esta guía se utiliza SKS2 como código de transacción para la aplicación DFSDDLTO.

### Sentencias de control DFSDDLTO

```
S11 1 1 1 1 TP 1
L GU
E OK
E Z0017 DATA SKS2 M2 SI1M3 SI1
WTO SEGMENT S11 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 STLDDL2 TERMINATED
L GU
```


En esta guía de aprendizaje se utilizan estructuras de datos COBOL para describir los mensajes de entrada y salida de transacción IMS. Tenga en cuenta que los mensajes de salida devueltos por IMS están formados por tres segmentos de longitud fija:

- OUTPUT-SEG1 (16 bytes)
- OUTPUT-SEG2 (31 bytes)
- OUTPUT-SEG3 (52 bytes)

El mensaje de salida devuelto por esta aplicación IMS particular tiene un tamaño fijo de 99 bytes y se representa mediante la estructura COBOL 01 OUTPUT-MSG.

Un modo de desarrollar esta aplicación multisegmento consiste en utilizar la definición de COBOL OUTPUT-MSG para definir la salida de la transacción. Un segundo modo consiste en crear un mensaje de salida para la salida de la transacción. El código que proporciona esta guía de aprendizaje utiliza el segundo método, ya que también se puede utilizar para crear una aplicación que procese un mensaje de salida de longitud variable. Se seguirán utilizando las definiciones COBOL de los distintos segmentos de mensaje para simplificar el acceso a los datos de cada uno de los segmentos

### Conexión con el servidor IMS

1. Si el icono J2EE, , no aparece en la pestaña superior derecha del espacio de trabajo, deberá pasar a la perspectiva J2EE. En la barra de menús, seleccione **Ventana > Abrir perspectiva > Otras**. Se abre la ventana Seleccionar perspectiva.
2. Seleccione **J2EE**.
3. Pulse **Aceptar**. Se abre la perspectiva J2EE.
4. En la perspectiva J2EE, seleccione **Archivo > Nuevo > Otro**.
5. En la página Nuevo, seleccione **J2C > Bean Java J2C**. Pulse **Siguiente**
6. En la página Selección de adaptadores de recurso, seleccione el adaptador de recursos IMS J2C 1.5. Para esta guía de aprendizaje, seleccione **IMS Connector para Java (IBM: 9.1.0.2.3)**. Pulse **Siguiente**.

7. En la página Propiedades de conexión, deseleccione **Conexión gestionada** y seleccione **Conexión no gestionada**. (En esta guía de aprendizaje utilizará una conexión no gestionada para acceder directamente a IMS.) Acepte el nombre de clase de conexión por omisión de `com.ibm.connector2.ims.ico.IMSManagedConnectionFactory`. En los campos en blanco, entre toda la información de conexión necesaria. Los campos obligatorios, indicados con un asterisco (\*), son los siguientes:
  - a. **Para conexión TCP/IP:**
    - 1) **Nombre de sistema principal:** (Obligatorio) Dirección IP o nombre de sistema principal de IMS Connect.
    - 2) **Número de puerto:** (Obligatorio) Número de puerto que utiliza la conexión IMS destino.
  - b. **Para la conexión de opción local:**
    - 1) **Nombre de IMS Connect:** (Obligatorio) Nombre de la conexión IMS destino.
    - 2)
  - c. **Para ambos:**
    - 1) **Nombre de almacén de datos:** (Obligatorio) Nombre del almacén de datos IMS destino.
8. Puede obtener la información de conexión solicitándola a su administrador de sistema IMS. Cuando haya proporcionado la información de conexión necesaria, pulse **Siguiente**.

## MSout.cbl

Este es el código de MSout.cbl:

### 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.

---

## Lección 1.2: Configurar un proyecto Web y la interfaz e implementaciones Java

En la lección 1.2 seguirá los pasos necesarios para crear un proyecto Web y una interfaz y unas implementaciones Java

Antes de empezar, debe completar la Lección 1.1. En esta lección realizará las siguientes tareas:

- Crear un bean Java J2C
- Crear un proyecto Web dinámico
  1. Todo el trabajo que se realiza en el entorno de trabajo debe asociarse a un proyecto. Los proyectos proporcionan una vista organizada de los archivos y directorios de trabajo, optimizada con funciones basadas en el tipo de proyecto. En el entorno de trabajo, todos los archivos deben residir en un proyecto, por lo que antes de crear el bean Java J2C, deberá crear un proyecto donde colocarlo. En la página Nuevo bean Java J2C, escriba el valor MultiSegOutput en el campo **Nombre de proyecto**.
  2. Pulse el botón **Nuevo** situado junto al campo **Nombre de proyecto** para crear el nuevo proyecto
  3. Ahora creará un proyecto Web dinámico Web. En la página Creación de un nuevo proyecto fuente, seleccione **Proyecto Web** y pulse **Siguiente**.
  4. En la página Nuevo proyecto Web dinámico, pulse **Mostrar valores avanzados**.
  5. Asegúrese de que se seleccionan los siguientes valores:
    - a. **Nombre de proyecto**: MultiSegOutput
    - b. **Contenido del proyecto**: acepte el valor predeterminado
    - c. **Tiempo de ejecución destino**: WebSphere Application Server v6.1
    - d. **Configuraciones**: acepte el valor predeterminado
    - e. **Añadir proyecto a un EAR**: marcado
    - f. **Nombre de proyecto EAR**: MultiSegOutputEAR
  6. Pulse **Siguiente**.

7. En la página Facetas de proyecto, acepte los valores predeterminados.
8. Pulse **Siguiente**.
9. En la página Módulos Web, acepte los valores predeterminados.
10. Pulse **Terminar**.
11. Aparecerá un recuadro de diálogo solicitando si desea cambiar a la perspectiva Web dinámica. Pulse **Sí**.
12. En la página Propiedades de salida del bean Java J2C:
  - a. En el campo **Nombre de paquete**, pulse **Examinar** y seleccione el proyecto MultiSegOutput. Pulse **Aceptar**.
  - b. Escriba `sample.ims` en el campo **Nombre de paquete**.
  - c. Escriba `MSO` en el campo **Nombre de interfaz**.
  - d. Escriba `MSOImpl` en el campo **Nombre de implementación**.
  - e. Deje sin marcar **Guardar sesión como script ant**.
13. Pulse **Terminar**.

---

## Lección 1.3: Crear una clase de almacenamiento intermedio de mensaje

En la lección 1.3 se describe la creación de una clase de almacenamiento intermedio de mensaje.

Antes de empezar, debe completar la Lección 1.2: configurar el proyecto Web y la interfaz e implementaciones Java. En esta lección realizará las siguientes tareas:

- Crear una clase de almacenamiento intermedio de mensaje
  - Editar la clase de almacenamiento intermedio de mensaje utilizando anotaciones doclet
  - Crear las operaciones de enlace de entrada y salida
  - Crear las correlaciones de datos de segmento de salida
1. **Primero creará una clase de almacenamiento intermedio de mensajes:** Expanda el proyecto **MultiSegOutput**, expanda **Recursos Java** y expanda **JavaSource**.
  2. Pulse con el botón derecho del ratón en el paquete **sample.ims** y seleccione **Nuevo> Clase** para lanzar el asistente Nueva clase.
  3. Especifique **CCIBuffer** como el nombre de la clase. Acepte todos los valores por omisión.
  4. Pulse **Terminar**. La clase **CCIBuffer** se abre en el editor Java.
  5. En la sección de comentarios de la clase **CCIBuffer**, teclee el código `@type-descriptor.message-buffer`.
  6. Pulse **CTRL-S** para guardar los cambios. Verá que se genera nuevo código automáticamente en "CCIBuffer.java" en la página 9.
  7. **A continuación creará un método para ejecutar la transacción IMS y el tipo de datos de mensaje de entrada:** en la vista Explorador de proyectos, pulse con el botón derecho sobre **MSOImpl.java** y seleccione **Fuente > Añadir método a bean Java J2C**.
  8. En la página Método Java pulse **Añadir**.
  9. Especifique `runMultiSegOutput` como el nombre del método Java. Pulse **Siguiente**.
  10. Pulse **Nuevo** para definir el tipo de entrada.
  11. Seleccione la correlación **COBOL\_a\_Java**. Pulse **Examinar**.
  12. Busque el archivo **COBOL MSOut.cbl**. Pulse **Abrir**.
  13. Pulse **Siguiente**.
  14. En la página Importador COBOL, pulse **Mostrar valores avanzados**.
    - a. Seleccione las siguientes opciones:

Tabla 1. Valores de parámetros de importador COBOL

Parámetro	Valor
Nombre de plataforma	Z/OS
Página de códigos	IBM-037
Nombre de formato de coma flotante	IBM Hexadecimal
Signo decimal externo	EBCDIC
Nombre Endian	Big
Nombre endian de entero remoto	Big
Nombre de comillas	DOUBLE
Nombre Trunc	STD
Nombre Nsymbol	DBCS

- b. Pulse **Consulta** para cargar los datos.
  - c. Se muestra una lista de estructuras de datos. Seleccione **INPUT-MSG** en el campo **Estructuras de datos**.
  - d. Pulse **Siguiente**.
  - e. Deje sin marcar **Guardar sesión como script Ant**.
15. En la página Guardar propiedades, el Nombre de clase predeterminado es **INPUTMSG**. Sobrescriba el Nombre de clase con **InputMsg**. Pulse **Terminar**.
16. **A continuación creará el tipo de datos de mensaje de salida:** pulse **Examinar** para definir el tipo de salida.
17. Teclee CC en el campo Seleccionar un tipo de datos y **CCIBuffer** aparecerá en el campo Tipos coincidentes. Seleccione **CCIBuffer** como el tipo de salida. Pulse **Terminar**.
18. En la página Método Java, pulse **Finalizar**.
19. En la página Métodos Java, asegúrese de que **interactionVerb** esté establecido en **SYNC\_SEND\_RECEIVE (1)** para indicar que la interacción con IMS implica una interacción de envío seguida de una de recepción.
20. Pulse **Terminar**.
21. **A continuación creará las correlaciones de datos de segmentos de salida. Primero creará la clase OutputSeg1.java:** Para realizar esta tarea, deberá utilizar un asistente de correlación de datos autónomo para crear solamente los archivos de correlación de datos.
22. Seleccione **Archivo > Nuevo > Otros > Enlace de datos Java CICS/IMS** para invocar el asistente Enlace de datos.
23. Pulse **Siguiente**.
24. Seleccione **COBOL\_a\_Java** en la lista **Elegir correlación**. Pulse **Examinar** para buscar el copy book COBOL MSOut.cbl.
25. Pulse **Siguiente**.
26. En la página Importador COBOL, pulse **Mostrar valores avanzados**.
  - a. Seleccione las siguientes opciones:

Tabla 2. Valores de parámetros de importador COBOL

Parámetro	Valor
Nombre de plataforma	Z/OS
Página de códigos	IBM-037
Nombre de formato de coma flotante	IBM Hexadecimal
Signo decimal externo	EBCDIC

Tabla 2. Valores de parámetros de importador COBOL (continuación)

Parámetro	Valor
Nombre Endian	Big
Nombre endian de entero remoto	Big
Nombre de comillas	DOUBLE
Nombre Trunc	STD
Nombre Nsymbol	DBCS

- b. Pulse **Consulta** para cargar los datos.
- c. Se muestra una lista de estructuras de datos. Seleccione **OUTPUT-SEG1** en el campo **Estructuras de datos**.
- d. Pulse **Siguiente**.
27. En el asistente Guardar propiedades, pulse **Examinar** y seleccione el proyecto **MultiSegOutput** que ha creado antes.
28. Pulse **Examinar** para seleccionar el nombre del paquete: **sample.ims.data**.
29. Cambie el nombre de clase Java de **OUTPUTSEG1** por **OutputSeg1**.
30. Pulse **Terminar**.
31. **Ahora creará la clase OutputSeg2.java:** Seleccione **Archivo > Nuevo > Otros > Enlace de datos Java CICS/IMS** para invocar el asistente Enlace de datos.
32. Pulse **Siguiente**.
33. Seleccione **COBOL\_a\_Java** en la lista **Elegir correlación**. Pulse **Examinar** para buscar el copy book **COBOL MSOut.cbl**.
34. En la página Importador COBOL, pulse **Mostrar valores avanzados**.
  - a. Seleccione las siguientes opciones:

Tabla 3. Valores de parámetros de importador COBOL

Parámetro	Valor
Nombre de plataforma	Z/OS
Página de códigos	IBM-037
Nombre de formato de coma flotante	IBM Hexadecimal
Signo decimal externo	EBCDIC
Nombre Endian	Big
Nombre endian de entero remoto	Big
Nombre de comillas	DOUBLE
Nombre Trunc	STD
Nombre Nsymbol	DBCS

- b. Pulse **Consulta** para cargar los datos.
- c. Se muestra una lista de estructuras de datos. Seleccione **OUTPUT-SEG2** en el campo **Estructuras de datos**.
- d. Pulse **Siguiente**.
35. En el asistente Guardar propiedades, pulse **Examinar** para seleccionar el proyecto **MultiSegOutput** que ha creado antes.
36. Pulse **Examinar** para seleccionar el nombre del paquete: **sample.ims.data**.
37. Cambie el nombre de clase Java de **OUTPUTSEG2** por **OutputSeg2**.
38. Pulse **Terminar**.

39. Ahora creará la clase **OutputSeg3.java**: Seleccione **Archivo > Nuevo > Otros > Enlace de datos Java CICS/IMS** para invocar el asistente Enlace de datos.
40. Pulse **Siguiente**.
41. Seleccione **COBOL\_a\_Java** en la lista **Elegir correlación**. Pulse **Examinar** para buscar el copy book **COBOL MSOut.cbl**.
42. En la página Importador COBOL, pulse **Mostrar valores avanzados**.
  - a. Seleccione las siguientes opciones:

*Tabla 4. Valores de parámetros de importador COBOL*

Parámetro	Valor
Nombre de plataforma	Z/OS
Página de códigos	037
Nombre de formato de coma flotante	IBM 390 Hexadecimal
Signo decimal externo	EBCDIC
Nombre Endian	Big
Nombre endian de entero remoto	Big
Nombre de comillas	DOUBLE
Nombre Trunc	STD
Nombre Nsymbol	DBCS

- b. Pulse **Consulta** para cargar los datos.
  - c. Se muestra una lista de estructuras de datos. Seleccione **OUTPUT-SEG3** en el campo **Estructuras de datos**.
  - d. Pulse **Siguiente**.
43. En el asistente Guardar propiedades, pulse **Examinar** para seleccionar el proyecto **MultiSegOutput** que ha creado antes.
44. Pulse **Examinar** para seleccionar el nombre del paquete: **sample.ims.data**.
45. Cambie el nombre de clase Java de **OUTPUTSEG3** por **OutputSeg3**.
46. Pulse **Terminar**.

## CCIBuffer.java

Este es el código generado en la clase CCIBuffer.java:

### 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;
}
}

```


---

## Lección 1.4: Crear una clase proxy Java para probar la aplicación

En la lección 1.4 se describe la creación de una clase de prueba Java para probar la aplicación.

antes de empezar, debe completar la Lección 1.3: Crear una clase de almacenamiento intermedio de mensaje. En esta lección realizará las siguientes tareas:

- Crear una clase de prueba Java.
- Editar la clase utilizando el código que se ofrece a continuación.
- Ejecutar la clase de prueba para probar la aplicación.

1. **Primero creará una clase de prueba Java:** Expanda el proyecto MultiSegOutput, expanda **Recursos Java** y seleccione el paquete **sample.ims**.
2. Pulse con el botón derecho del ratón y seleccione **Nuevo**. Seleccione la opción de clase  para crear una clase Java nueva.
3. En el campo nombre de clase Java, teclee TestMultiSeg. Tenga en cuenta que la clase TestMultiSeg sólo se ofrece como ejemplo; deberá cambiar el código de transacción para sus especificaciones de máquina IMS. Póngase en contacto con su administrador IMS para conocer el código de transacción. Puede localizar la sentencia `input.setIn_trcd("SKS6 ")` en la clase TestMultiSeg.java y realizar las modificaciones.
4. Asegúrese de que **Carpeta fuente** contiene **MultiSegOutput/JavaSource** y que el **Nombre de paquete** contiene **sample.ims**.
5. Pulse **Terminar**.
6. Abra **TestMultiSeg.java** en el editor de clases Java.
7. Sustituya todo el código del editor por el siguiente:

```

/*****
 * Licensed Materials - Property of IBM
 *
 * com.ibm.j2c.cheatsheet.content
 *
 * Copyright IBM Corporation 2004. Reservados todos los derechos.
 *
 * 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.etools.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. **Ahora probará la aplicación:** expanda el proyecto **MultiSegOutput** y el paquete **sample.ims**.
9. Pulse con el botón derecho del ratón en la clase **TestMultiSeg.java** y seleccione Ejecutar. Seleccione Ejecutar como > Aplicación Java
10. Deberá ver el siguiente mensaje en la consola:

```

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. Enhorabuena. Ha completado la guía de aprendizaje de salida de varios segmentos.

---

## Resumen de la creación de una aplicación J2C para una transacción IMS que contenga longitud variable y varios segmentos

Esta guía de aprendizaje le ha guiado a través de todos los pasos detallados para generar una aplicación J2C que procesa mensajes de salida de transacción IMS de longitud variable y con múltiples segmentos.

### Lecciones aprendidas

En esta guía de aprendizaje ha aprendido a

- Utilizar el asistente Bean Java J2C para crear un bean Java J2C que ejecute una transacción IMS.
- Crear una clase de almacenamiento intermedio de mensaje, CCIBuffer.java, y editar esta clase utilizando anotaciones doclet.
- Crear un método para que el bean Java J2C ejecute la transacción IMS y proporcionar tipos de datos de entrada y salida para el método.
- Crear enlaces de datos Java para los segmentos del mensaje de salida.
- Crear una clase de prueba Java, TestMultiSeg.java, para invocar el método de bean Java J2C que ejecuta la transacción IMS y poblar los segmentos de salida desde el almacenamiento intermedio de los datos devueltos por la transacción IMS.