Construir una aplicación para un copybook COBOL de CICS: varias salidas posibles

Contenido

Crear una aplicación J2C para una transacción CICS que contiene varias salidas posibles

En esta guía de aprendizaje se describe cómo utilizar el Asistente J2C Java Bean para construir una aplicación Web sencilla que procese una transacción que pueda procesar múltiples salidas posibles.

Objetivos educativos

Esta guía de aprendizaje le permite:

- Utilizar el asistente Bean Java J2C para crear una aplicación J2C que interactúa con una transacción CICS utilizando una interfaz de llamadas externas (ECI). En función de la clasificación del cliente (cliente preferido, cliente normal o cliente de poca calidad), el programa devuelve diferente información de salida acerca del cliente. Crear una JSP para desplegar la aplicación en un servidor de aplicaciones de WebSphere.
- Crear un método Java que acepta un número de cliente.
- Crear una JSP para desplegar la aplicación en un servidor de aplicaciones de WebSphere.

Tiempo necesario

30 minutos

Información relacionada

Ver la versión PDF

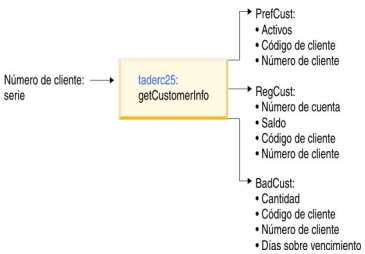
Ejemplo de varias salidas posibles

Introducción a la creación de una aplicación J2C para una transacción CICS que contiene varias salidas posibles

En esta guía de aprendizaje se describe cómo utilizar el asistente de bean Java J2C para crear una aplicación Web sencilla que procese una transacción CICS con varias salidas posibles.

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.

En esta guía de aprendizaje recorrerá todos los pasos detallados para generar una aplicación J2C que interactúa con una transacción CICS que utiliza una interfaz de llamadas externas (ECI). El servicio se crea a partir de una función CICS COBOL, getCustomerInfo, que acepta un número de cliente. En función de la clasificación del cliente (cliente preferido, cliente normal o cliente de poca calidad), el programa devuelve diferente información de salida acerca del cliente.



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 conectarse a un servidor CICS ECI. Mientras realiza los ejercicios podrá:

- Utilizar el asistente Bean Java J2C para crear una aplicación J2C que interactúa con una transacción CICS utilizando una interfaz de llamadas externas (ECI).
- Crear un método Java, *getCustomerInfo*, que acepta un número de cliente. En función de la clasificación del cliente (cliente preferido, cliente normal o cliente de poca calidad), el programa devuelve diferente información de salida acerca del cliente.
- Crear una clase Java TestECIMPO.java para probar su aplicación.

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 CICS ECI 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)
- **Conexión con un servidor CICS**: en esta guía de aprendizaje, la aplicación interactúa con un programa CICS en un servidor. Específicamente, es necesario configurar una pasarela de transacción CICS en un sistema para acceder al servidor. También debe configurar el sistema servidor CICS en el que desea ejecutar CICS. Estos pasos no están cubiertos.
- **Una copia del archivo COBOL taderc25.cbl**. Encontrará este archivo en el directorio de instalación del producto: <dir_instal>\IBM\SDP70Shared\plugins\com.ibm.j2c.cheatsheet.content_7.0.0\Samples\CICS\taderc25. Si desea almacenarlo localmente, puede copiar el código de aquí: “taderc25.cbl” en la página 3
- **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
- Lenguaje de programación COBOL
- Tecnología de servidor CICS ECI

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 ECI CICS.

Conexión con el servidor ECI CICS

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 Adaptadores de recurso, en **Ver por**, seleccione **Versión JCA**. Expanda **1.5** y seleccione **ECIResourceAdapter (IBM:6.0.2)**. Pulse **Siguiente**.
7. En la página Propiedades de conexión, seleccione **Conexión no gestionada**. (En esta guía de aprendizaje utilizará la conexión no gestionada para acceder directamente al servidor CICS, por lo que no se necesita proporcionar el nombre JNDI.) Acepte el **Nombre de clase de conexión** de `com.ibm.connector2.cics.ECIManagedConnectionFactory`. En los campos en blanco, proporcione toda la información de conexión necesaria. Los campos obligatorios, indicados con un asterisco (*), son los siguientes:
   - **Nombre de servidor**: (Opcional) Nombre del servidor CICS Transaction Gateway.
   - **URL de conexión***: (Obligatorio) Dirección del servidor CICS ECI.
   - **Número de puerto**: (Opcional) Número de puerto que se utiliza para comunicarse con CICS Transaction Gateway. El puerto por omisión es 2006.
   - **Nombre de usuario**: (Obligatorio) El nombre de usuario para la conexión.
   - **Contraseña**: (Obligatorio) Contraseña de la conexión.
   - :

   Puede obtener la información de conexión solicitándola a su administrador de sistema del servidor CICS.
8. Pulse **Siguiente**.

taderc25.cbl

Este es el código de taderc25.cbl:

taderc25.cbl

```
identification division.
 program-id. TADERC25.
 environment division.
 data division.
 working-storage section.
 01 tmp pic a(40).
 01  ICOMMAREA.
     02  ICustNo    PIC X(5).
     02  Ifiller    PIC X(11).
 01  GENCUST.
     02  GCUSTCODE PIC X(4).
     02  GFILLER PIC X(40).
 01  PREFCUST.
     02  PCUSTCODE PIC X(4).
     02  PCUSTNO    PIC X(5).
     02  ASSETS  PIC S9(6)V99.
 01  REGCUST.
     02  RCUSTCODE PIC X(4).
```

```
    02  RCUSTNO    PIC X(5).
    02  ACCOUNTNAME PIC A(10).
    02  BALANCE PIC S9(6)V99.
01  BADCUST.
    02  BCUSTCODE PIC X(4).
    02  BCUSTNO    PIC X(5).
    02  DAYSOVERDUE PIC X(4).
    02  AMOUNT PIC S9(6)V99.
LINKAGE SECTION.
01 DFHCOMMAREA.
   02 inputfield pic x(50).
procedure division.
start-para.
    move DFHCOMMAREA to ICOMMAREA.
    IF ICustNo EQUAL '12345'
       move 'PREC' to PCUSTCODE
       move ICustNo to PCUSTNO
       move 43456.33 to ASSETS
       move PREFCUST TO DFHCOMMAREA
    ELSE IF ICustNo EQUAL '34567'
            move 'REGC' to RCUSTCODE
            move ICustNo to RCUSTNO
            move 'SAVINGS' TO ACCOUNTNAME
            move 11456.33 to BALANCE
            move REGCUST TO DFHCOMMAREA
         ELSE
            move 'BADC' to BCUSTCODE
            move ICustNo to BCUSTNO
            move '132' to DAYSOVERDUE
            move -8965.33 to AMOUNT
            move BADCUST TO DFHCOMMAREA
*        END-IF.
    END-IF.
    EXEC CICS RETURN
    END-EXEC.
```

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

Esta lección le lleva a través del proceso de configuración de un proyecto Web y de las implementaciones y la interfaz 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.
2. En la página Nuevo bean Java J2C, escriba el valor `Taderc25Sample` en el campo **Nombre de proyecto**.
3. Pulse **Nuevo** junto al campo **Nombre de proyecto** para crear el proyecto nuevo.
4. En la página Creación de un nuevo proyecto fuente, seleccione **Proyecto Web** y pulse **Siguiente**.
5. En la página Nuevo proyecto Web dinámico, pulse **Mostrar valores avanzados**.
6. Asegúrese de que se seleccionan los siguientes valores:
   a. **Nombre de proyecto**: Taderc25Sample
   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**: Taderc25SampleEAR

7. Pulse **Terminar**.
8. Aparecerá un recuadro de diálogo solicitando si desea cambiar a la perspectiva Web dinámica. Pulse **Sí**.
9. En la página Propiedades de salida del bean Java J2C:
   a. En el campo **Nombre de paquete**, pulse **Examinar** y seleccione el proyecto `Taderc25Sample`. Pulse **Aceptar**.
   b. Escriba `sample.ims` en el campo **Nombre de paquete**.
   c. Escriba `CustomerInfoMO` en el campo **Nombre de interfaz**.
   d. Escriba `CustomerInfoMOImpl` en el campo **Nombre de implementación**.
   e. Deje sin marcar **Guardar sesión como script Ant**.
10. Pulse **Terminar**.

Lección 1.3: Crear un método Java

En la lección 1.3 se describe la creación de un método Java.

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 un método Java
- Crear la correlación de datos de entrada y salida entre COBOL y Java

1. **Primero creará un método Java:** Ahora vamos a crear un método Java que utilizará el importador COBOL para correlacionar los tipos de datos entre el fuente COBOL y los datos del método Java.
2. Abra la vista Fragmentos de código pulsando **Ventana > Mostrar vista > Fragmentos de código**. En la vista Fragmentos de código, pulse **J2C**.
3. Pulse con el botón derecho del ratón en **Añadir método Java a bean Java J2C** y seleccione **Insertar**.
4. En la página Método Java pulse **Añadir**.
5. En el campo de nombre **Método Java**, teclee `getCustomerInfo` para el nombre de la operación. Pulse **Siguiente**.
6. **A continuación creará la correlación de datos de parámetro de entrada**: en este paso importará el archivo "taderc25.cbl" en la página 3 (COBOL) necesario para crear la aplicación. El archivo taderc25.cbl está ubicado en <dir_instal>\IBM\SDP70Shared\plugins\com.ibm.j2c.cheatsheet.content_7.0.0\Samples\CICS\taderc25, donde <dir_instal> es el directorio en el que se ha instalado este producto. El archivo COBOL contiene el programa que se ejecuta en el servidor CICS. Tiene la definición de la estructura que se pasa al servidor CICS a través del área de comunicaciones (COMMAREA). Esta estructura representa los registros de cliente que se devuelven de la aplicación CICS. Antes de poder trabajar con un archivo, debe importarlo del sistema de archivos al entorno de trabajo. En el campo **Especificar el tipo de entrada/salida** de la página Método Java, pulse **Nuevo**.
7. En la página Importar datos, asegúrese de que el campo **Elegir correlación** es **COBOL_TO_JAVA**. Pulse **Examinar** junto al archivo COBOL
8. Busque el archivo taderc25.cbl en el sistema de archivos y pulse **Abrir**.
9. Pulse **Siguiente**.
10. En la página Importador COBOL, **seleccione una estructura de datos de comunicación**:
    a. Seleccione **Win32** para **Nombre de plataforma**.
    b. Seleccione **ISO-8859-1** para **Página de códigos**.
    c. Pulse **Consultar**.

d. Seleccione **ICOMMAREA** para **Estructuras de datos**.

11. Pulse **Siguiente**.
12. En la página Guardar propiedades:
    a. Seleccione **Valor predeterminado** para **Estilo de generación**.
    b. Pulse **Examinar** para elegir el proyecto Web **Taderc25Sample**.
    c. En el campo **Nombre de paquete**, teclee `sample.cics.data`
    d. En el campo **Nombre de clase**, el valor por omisión es **ICOMMAREA**; sustitúyalo por `InputComm`.
13. Pulse **Terminar**.
14. **A continuación creará las distintas salidas posibles para el parámetro de salida:** en el campo **Especificar el tipo de entrada/salida** de la página Método Java, pulse **Nuevo** junto al área Tipo de salida.
15. En la página Importar datos, asegúrese de que el campo Elegir correlación es **COBOL_MPO_TO_JAVA**.
16. Pulse **Nuevo** junto al área de las múltiples salidas posibles.
17. Pulse **Examinar** junto al campo Nombre de archivo cobol y busque el archivo **taderc25.cbl**. Pulse **Abrir**.
18. Pulse **Siguiente**.
19. En la página Importador COBOL, **seleccione una estructura de datos de comunicación**:
    a. Seleccione **Win32** para **Nombre de plataforma**.
    b. Seleccione **ISO-8859-1** para **Página de códigos**.
    c. Pulse **Consultar**.
    d. Seleccione **PREFCUST**, **REGCUST** y **BADCUST** para **Estructuras de datos**.
20. Pulse **Terminar**. En la página Especificar propiedades de configuración de importación de datos, figurarán los tres tipos de datos.
21. Pulse **Siguiente**.
22. **A continuación especificará las propiedades de guardado:** En la página Guardar propiedades verá valores por omisión para cada uno de los registros de tipo personalizado. Asegúrese de que **Taderc25Sample** aparece en el campo **Nombre de proyecto**. Pulse **Examinar** y elija el proyecto Web **Taderc25Sample**.
    a. En la página Especificar propiedades de guardado, resalte **Propiedades de guardado de MPO COBOL a Java**.
       - Teclee `sample.cics.data` en el campo **Nombre de paquete**
       - Teclee `OutputComm` en el campo **Nombre de clase**.
       - Puede seleccionar **Sobrescribir clase existente** .
    b. Expanda **Propiedades de guardado de COBOL MPO a Java**. Deben aparecer los tres elementos de enlace de datos.
    c. Resalte **Propiedades de guardado de COBOL a Java para ″PREFCUST″ en el archivo taderc25.cbl**
       - Para **Estilo de generación**, seleccione **Predeterminado**.
       - Teclee `sample.cics.data` en el campo **Nombre de paquete**
       - Teclee `PrefCust` en el campo **Nombre de clase**.
       - Puede seleccionar **Sobrescribir clase existente** .
    d. Resalte **Propiedades de guardado de COBOL a para ″REGCUST″ en el archivo taderc25.cbl**.
       -
       - Teclee `sample.cics.data` en el campo **Nombre de paquete**
       - Teclee `RegCust` en el campo **Nombre de clase**.

- Puede seleccionar **Sobrescribir clase existente** .

e. Resalte **Propiedades de guardado de COBOL a Java para "BADCUST" en el archivo taderc25.cbl**.
   - Teclee `sample.cics.data` en el campo **Nombre de paquete**
   - Teclee `BadCust` en el campo **Nombre de clase**.
   - Puede seleccionar **Sobrescribir clase existente** .

23. Pulse **Terminar**. Expanda *OutputComm*, y verá que contiene PrefCust, RegCust y BadCust en el campo **Tipo de salida**.
24. En la página Método Java, pulse **Finalizar**.
25. En la página Métodos Java:
    a. Teclee `TADERC25` (el id de programa COBOL) en el campo **functionName**.
    b. Seleccione **Mostrar valores avanzados**.
    c. Seleccione **SYNC_SEND_RECEIVE(1)** en el campo **interactionVerb**.
    d. Teclee `-1` en el campo **replyLength**.
26. Pulse **Terminar**.
27. **Ahora añadirá el código de patrón de reconocimiento para generar el archivo de correlación de datos de salida Java:** Dado que la salida que proviene del programa puede ser de cualquiera de los tipos de datos, el único modo de identificarla consiste en tener algunos patrones predefinidos en la corriente de datos. El método de identificación comprueba el patrón de reconocimiento.

```
@type-descriptor.external-decimal-td
@type-descriptor.float-td
@type-descriptor.initial-value
@type-descriptor.integer-td
@type-descriptor.level88
@type-descriptor.packed-decimal-td
@type-descriptor.recognition-desc
@type-descriptor.restriction
@type-descriptor.simple-instance-td
```

    a. Para añadir el patrón de reconocimiento de PrefCust:
       - Abra el archivo PrefCust.java en un editor Java.
       - Navegue al método **getPcustcode()**. La mejor forma de hacer esto consiste en abrir la vista Esquema y desplazarse hasta encontrar el método deseado.
       - En el área de comentarios del método, añada el código `@type-descriptor.recognition-desc pattern="PREC"` o utilice la asistencia de contenido pulsando Ctrl-Espacio, recorra la lista hasta encontrar el código y especifique `"PREC"` como patrón.
       - Guarde los cambios y se regenerará el código de PrefCust.java.
       - Desplácese hasta el método de identificación para asegurarse de que el cambio se ha efectuado:

```
/**
  * @generated

  */
 public boolean match(Object obj) {
  if (obj == null)
   return (false);
  if (obj.getClass().isArray()) {
   byte[] currBytes = buffer_;
```

```
 try {
  byte[] objByteArray = (byte[]) obj;
  buffer_ = objByteArray;
  if (!("PREC".equals(getPcustcode().toString())))
   return (false);
 } catch (ClassCastException exc) {
  return (false);
 } finally {
  buffer_ = currBytes;
 }
} else
 return (false);
return (true);
}
```

b. Para añadir el patrón de reconocimiento de RegCust:
   - Abra el archivo RegCust.java en un editor Java.
   - Navegue al método **getRcustcode()**. De nuevo, la mejor forma de hacer esto consiste en abrir la vista Esquema y desplazarse hasta encontrar el método deseado.
   - En el área de comentarios del método, añada el código `@type-descriptor.recognition-desc pattern="REGC"` o utilice la asistencia de contenido pulsando Ctrl-Espacio, recorra la lista hasta encontrar el código y especifique `"REGC"` como patrón.
   - Guarde los cambios y se regenerará el código de RegCust.java.
   - Desplácese hasta el método de identificación para asegurarse de que el cambio se ha efectuado:

```
/**
 * @generated

 */
public boolean match(Object obj) {
 if (obj == null)
  return (false);
 if (obj.getClass().isArray()) {
  byte[] currBytes = buffer_;
  try {
   byte[] objByteArray = (byte[]) obj;
   buffer_ = objByteArray;
   if (!("REGC".equals(getRcustcode().toString())))
    return (false);
  } catch (ClassCastException exc) {
   return (false);
  } finally {
   buffer_ = currBytes;
  }
 } else
  return (false);
 return (true);
}
```

c. Para añadir el patrón de reconocimiento de BadCust:
   - Abra el archivo BadCust.java en un editor Java.
   - Navegue al método **getBcustcode()**. De nuevo, la mejor forma de hacer esto consiste en abrir la vista Esquema y desplazarse hasta encontrar el método deseado.
   - En el área de comentarios del método, añada el código `@type-descriptor.recognition-desc pattern="BADC"` o utilice la asistencia de contenido pulsando Ctrl-Espacio, recorra la lista hasta encontrar el código y especifique `"BADC"` como patrón.
   - Guarde los cambios y se regenerará el código de BadCust.java.
   - Desplácese hasta el método de identificación para asegurarse de que el cambio se ha efectuado:

```
/**
 * @generated

 */
```

```
public boolean match(Object obj) {
 if (obj == null)
  return (false);
 if (obj.getClass().isArray()) {
  byte[] currBytes = buffer_;
  try {
   byte[] objByteArray = (byte[]) obj;
   buffer_ = objByteArray;
   if (!("BADC".equals(getBcustcode().toString())))
    return (false);
  } catch (ClassCastException exc) {
   return (false);
  } finally {
   buffer_ = currBytes;
  }
 } else
  return (false);
 return (true);
}
```

Lección 1.4: Desplegar la aplicación

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

Antes de empezar, debe completar la Lección 1.3. En esta lección realizará las siguientes tareas:

- Crear una clase Java para probar su aplicación.
- Ejecutar la clase de prueba.

1. **Primero creará el archivo TestECIMPO:** Expanda el proyecto **CustomerProj**, expanda la sección **Recursos Java** y seleccione el paquete **sample.cics**.
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 `TestECIMPO`
4. Abra **TestECIMPO** en el editor Java.
5. Sustituya todo el código del editor por el siguiente:

   Nota: La clase TestECIMPO.java Java se ha creado para un entorno local inglés; deberá modificar el código para otros entornos locales.

```
/*************************************************************
 * 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 C
 *************************************************************/
package sample.cics;

import sample.cics.data.*;
public class TestECIMPO
{



 public static void process(InputComm input)
 {

  System.out.println("processing....");
     try {
  //CustomerInfoMOImpl proxy = new CustomerInfoMOImpl();
  CustomerInfoMOImpl proxy = new CustomerInfoMOImpl();
```

```
    OutputComm output = proxy.getCustomerInfo (input);

 BadCust badCust = output.getBadCust();
 PrefCust prefCust = output.getPrefCust();
 RegCust regCust = output.getRegCust();

 if (regCust != null)
 {
  System.out.println("Reg Customer");
  System.out.println("account name:  " + regCust.getAccountname());
  System.out.println("balance:  " + regCust.getBalance());
  System.out.println("cust code:  " + regCust.getRcustcode());
  System.out.println("cust no:  " + regCust.getRcustno());
 }
 else if (prefCust != null)
 {
  System.out.println("Pref Customer");
  System.out.println("assets:  " + prefCust.getAssets());
  System.out.println("cust code:  " + prefCust.getPcustcode());
  System.out.println("cust no:  " + prefCust.getPcustno());
 }
 else if (badCust != null)
 {
  System.out.println("Bad Customer");
  System.out.println("amount:  " + badCust.getAmount());
  System.out.println("cust code:  " + badCust.getBcustcode());
  System.out.println("cust no:  " + badCust.getBcustno());
  System.out.println("days overdue:  " + badCust.getDaysoverdue());
 }
 else
  System.out.println("No match");
}
catch (Exception exc)
{
 System.out.println (exc);
 exc.printStackTrace();
}

}

public static void testPrefCust()
{
   System.out.println("===========testPreCust==============");
try {
 InputComm input = new InputComm();
 String prefC = "12345";
 input.setICustNo (prefC);
 process(input);
}
catch (Exception exc)
{
 System.out.println (exc);
 exc.printStackTrace();
}


}

public static void testRegCust()
{
   System.out.println("===========testRegCust==============");
try {
 InputComm input = new InputComm();
 String regC = "34567";
 input.setICustNo (regC);
 process(input);
}
```

```
catch (Exception exc)
{
 System.out.println (exc);
 exc.printStackTrace();
}

}

public static void testBadCust()
{

 System.out.println("==========testBadCust=============");
try {

 InputComm input = new InputComm();
 String badC = "123";
 input.setICustNo (badC);
 process(input);

}
catch (Exception exc)
{
 System.out.println (exc);
 exc.printStackTrace();
}
}


public static void main (String[] args)
{
 testPrefCust();
 testRegCust();
 testBadCust();

}
}
```

6. **A continuación probará la aplicación**
7. Pulse con el botón derecho sobre **TestECIMPO.java** y seleccione **Ejecutar como> Aplicación Java**.
8. La consola debe visualizar la salida siguiente:

```
=========testPreCust=======
processing. . . .
Pref Customer
assets:  43456.33
cust code: PREC
cust no: 12345
=========testRegCust=======
processing. . . .
Reg Customer
account name: SAVINGS
assets:  11456.33
cust code: REGC
cust no: 34567
=========testBadCust=======
processing. . . .
Bad Customer
assets:  -8965.33
cust code: BADC
cust no: 123
days overdue: 132
```

Enhorabuena. Ha completado la guía de aprendizaje CICS Taderc25.

Resumen de la creación de una aplicación J2C para una transacción CICS que contiene varias salidas posibles

Con esta guía de aprendizaje ha aprendido a utilizar el asistente Bean Java J2C para construir una aplicación web simple que procesa una transacción CICS con varias salidas posibles.

Lecciones aprendidas

En esta guía de aprendizaje ha aprendido a

- Utilizar el asistente Bean Java J2C para crear una aplicación J2C que interactúa con una transacción CICS utilizando una interfaz de llamadas externas (ECI).
- Crear un método Java, **getCustomerInfo**, que acepta un número de cliente. En función de la clasificación del cliente (cliente preferido, cliente normal o cliente de poca calidad), el programa devuelve diferente información de salida acerca del cliente.
- Crear una clase Java de prueba para probar su aplicación.
-