CS266 Software Reverse Engineering (SRE)

Reengineering and Reuse of Legacy Software

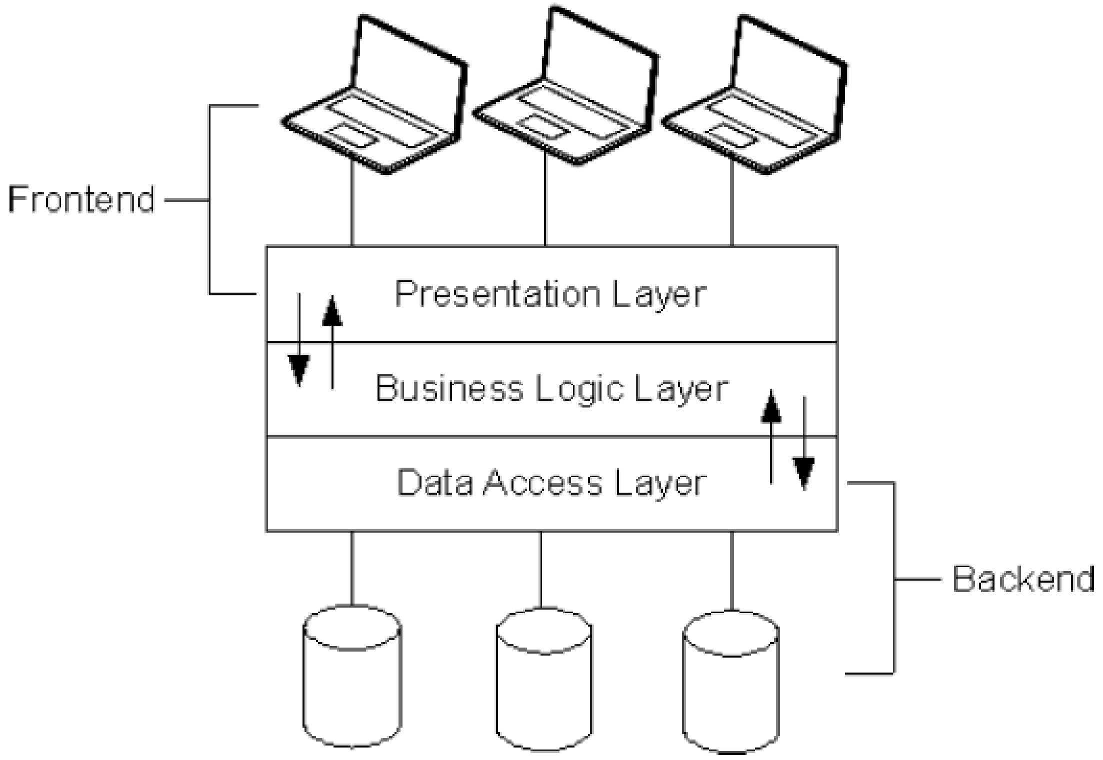
Teodoro (Ted) Cipresso, teodoro.cipresso@sjsu.edu
Department of Computer Science
San José State University
Spring 2015

The information in this presentation is taken from the thesis "Software reverse engineering education" available at http://scholarworks.sjsu.edu/etd_theses/3734/ where all citations can be found.

Reengineering and Reuse of Legacy Software

Introduction, Motivation, and Considerations

- If good development practices were followed, legacy software is typically composed of three layers [5]:



Layers of a well-structured legacy software application.

Reengineering and Reuse of Legacy Software

Introduction, Motivation, and Considerations

- Legacy applications that are not sufficiently componentized, such that their general organization resembles the three layers, are not good candidates for reengineering and reuse.

- The most widely accepted technique to reuse legacy application components is that of *Wrappering* [5], where a new piece of code provides an interface to a legacy application component or layer without requiring code changes to it.

- Typically, candidate applications should be well-structured such that the business logic can be isolated, encapsulated, and made into reusable components.

Reengineering and Reuse of Legacy Software

Introduction, Motivation, and Considerations

- Unless enough of an application's source code remains such that it's possible to identify the names of reusable entry points (procedures) and their I/O data structures, attempting to reuse the application may be difficult.

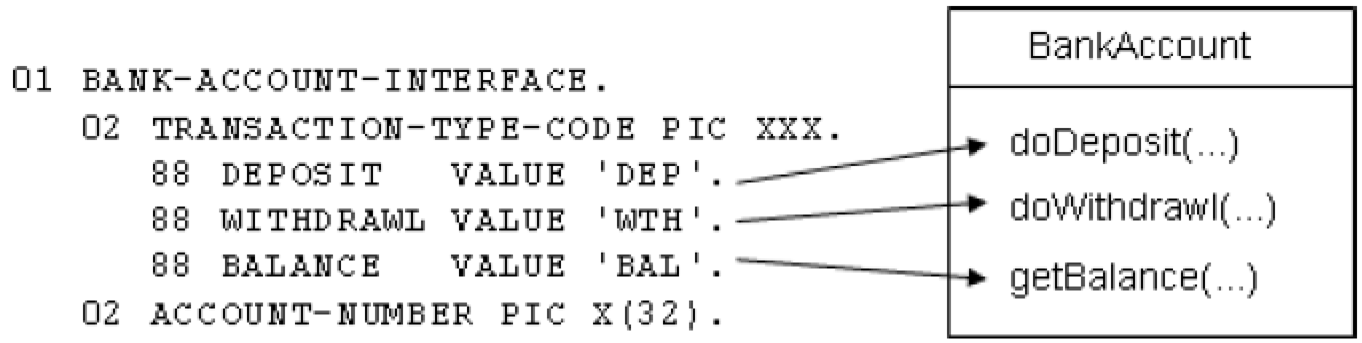
- While it is possible to learn the names of entry points that have been explicitly exported by an application in the case of a DLL, the names don't indicate the layout of the expected I/O data structures.

- One way to discover the entry points and I/O data structures in legacy machine code is to read the source code of other applications which depend on it.

Reengineering and Reuse of Legacy Software

COBOL and “Legacy” Languages

- The COBOL programming language is most often associated with legacy software applications.
- Normally, COBOL programs have a single entry point; additional “alternate” entry points are rare.
- Legacy COBOL programs often include functional discriminators in their I/O data structures.



Mapping legacy functional discriminators to an object-oriented design.

Reengineering and Reuse of Legacy Software

Reuse of Code in the Business Logic Layer

- In a real-world situation, we would be looking to reuse legacy components whose machine code is the result of thousands of lines of high-level language statements (COBOL) that implement a particular business process.
- Since our focus is more on reuse and reengineering of legacy code at a basic level, it's not necessary to encumber ourselves with a very large program in order to learn strategies for reuse and reengineering.
- Therefore we consider a small COBOL “calculator” that we wish to make reusable from Java. This program is assumed to be something from the business logic layer.

Reengineering and Reuse of Legacy Software

Sample COBOL Business Logic Component

```
******************************************************************
** Simple COBOL program that performs integer arithmetic        **
******************************************************************
 IDENTIFICATION DIVISION.
  PROGRAM-ID. 'SMPLCALC'.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 77 MSG-NUMERIC-OVERFLOW PIC X(25)
    VALUE 'Numeric overflow occurred'.
 77 MSG-SUCCESSFUL PIC X(22)
    VALUE 'Completed successfully'.
 LINKAGE SECTION.
* Input/Output data structure
 01 SMPLCALC-INTERFACE.
    02 SI-OPERAND-1 PIC S9(9) COMP-5.
    02 SI-OPERAND-2 PIC S9(9) COMP-5.
    02 SI-OPERATION PIC X.
       88 DO-ADD VALUE '+'.
       88 DO-SUB VALUE '-'.
       88 DO-MUL VALUE '*'.
    02 SI-RESULT PIC S9(18) COMP-3.
    02 SI-RESULT-MESSAGE PIC X(128).
 PROCEDURE DIVISION USING
   BY REFERENCE SMPLCALC-INTERFACE.
 MAINLINE SECTION.
* Perform requested arithmetic
```

Reengineering and Reuse of Legacy Software

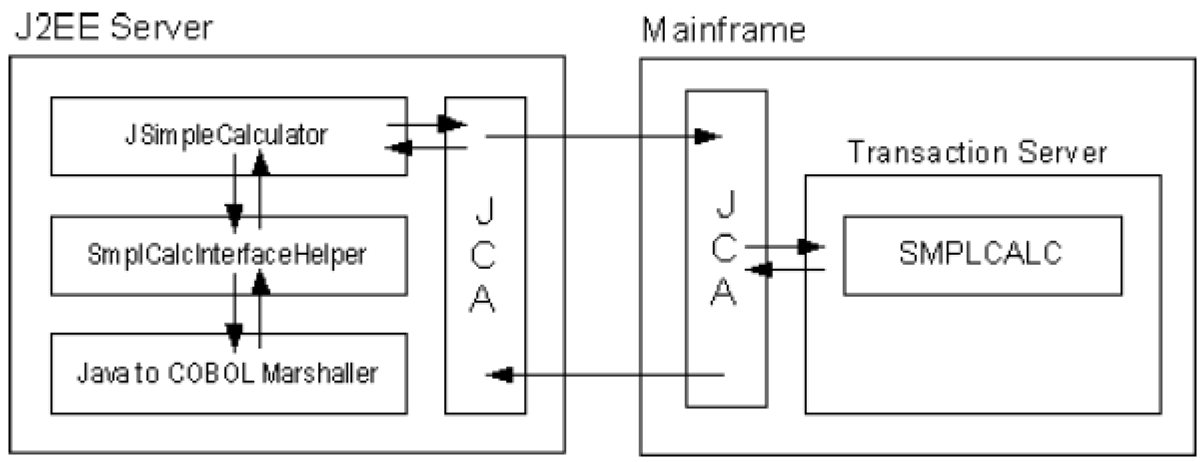
Sample COBOL Business Logic Component

```
    INITIALIZE SI-RESULT SI-RESULT-MESSAGE
    EVALUATE TRUE
      WHEN DO-ADD
        COMPUTE SI-RESULT = SI-OPERAND-1 + SI-OPERAND-2
          ON SIZE ERROR
            PERFORM HANDLE-SIZE-ERROR
        END-COMPUTE
      WHEN DO-SUB
        COMPUTE SI-RESULT = SI-OPERAND-1 - SI-OPERAND-2
          ON SIZE ERROR
            PERFORM HANDLE-SIZE-ERROR
        END-COMPUTE
      WHEN DO-MUL
        COMPUTE SI-RESULT = SI-OPERAND-1 * SI-OPERAND-2
          ON SIZE ERROR
            PERFORM HANDLE-SIZE-ERROR
        END-COMPUTE
    END-EVALUATE
 * Successful return
    MOVE MSG-SUCCESSFUL TO SI-RESULT-MESSAGE
    MOVE 2 TO RETURN-CODE
    GOBACK
    .
```

Reengineering and Reuse of Legacy Software

Modernizing Business Logic Components

- Many commercial tools support importing a COBOL data structure and generating Java marshalling classes.
- These marshalling classes are intended to be used with the J2EE Connector Architecture (JCA) where a Java application wrappers a legacy software application.



Example JCA implementation for accessing a legacy application.

Reengineering and Reuse of Legacy Software

Modernizing Business Logic Components (cont'd)

- A popular alternative to using the JCA architecture to reengineer and reuse legacy applications is to implement a Service Oriented Architecture (SOA).
- SOA components become capable of communicating without the tight and fragile coupling of traditional binary interfaces because they are wrappered with a platform-neutral interface such as XML and Web services.
- When XML is used as envisioned, all data, both of type character and numeric are represented as printable text—completely divorced from any platform specific representation or encoding.

Reengineering and Reuse of Legacy Software

Modernizing Business Logic Components (cont'd)

- The net effect of this is that two entities or programs can interact without having to know the data structures that comprise each other's binary interface.
- Of course, the XML that is exchanged cannot be arbitrary, so industry standards such as XML Schema (XSD), and Web Services Definition Language (WSDL) fill this gap.
- A Web service is considered to be WS-I compliant, or generally interoperable, if it meets many criteria, one of which is the use of XML for the input and output of each operation exposed by service.

Reengineering and Reuse of Legacy Software

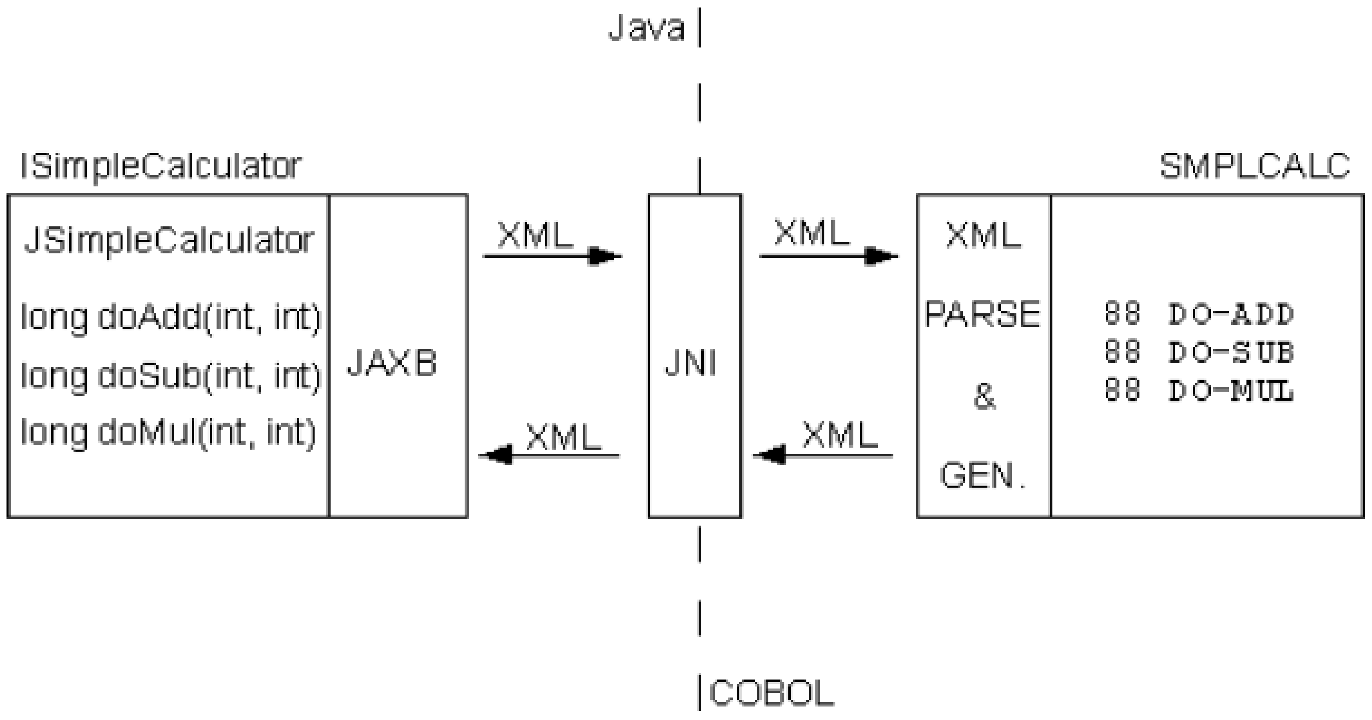
Software Reengineering/Reuse Exercise

- This particular requirement of WS-I where XML is *the* interoperable interface of choice, sets the stage for a meaningful exercise.
- A Legacy Software Reengineering and Reuse Exercise was developed to demonstrate wrapping a COBOL program so that is reusable from Java using XML in a local environment.
- In the exercise, one is asked to create a language neutral XML interface to the COBOL calculator program and invoke it from a Java program, which incidentally makes it reusable from other Java programs.

Reengineering and Reuse of Legacy Software

Software Reengineering/Reuse Exercise (cont'd)

- Overview of the architecture for the exercise:



Architecture for legacy application reengineering and reuse from Java.

Reengineering and Reuse of Legacy Software

Software Reengineering/Reuse Exercise (cont'd)

- Steps in the reengineering and reuse exercise:
  - Create an XML Schema which represents all of the data in the SMPLCALC-INTERFACE COBOL data structure.
  - Write a Java interface ISimpleCalculator.java for three computation types supported by SMPLCALC.cbl.
  - Write a Java class JSimpleCalculator.java that implements the interface defined in ISimpleCalculator.java and provides a user interface.
  - Use the Java command-line utility xjc, in combination with the XML Schema, generate Java to XML marshalling code (JAXB).

Reengineering and Reuse of Legacy Software

Software Reengineering/Reuse Exercise (cont'd)

- Steps in the reengineering and reuse exercise (cont'd):
  - Write a small C/C++ JNI program Java2CblXmlBridge.cpp which exports a method Java2SmplCalc that:
    - Invokes XML2CALC.cbl, passing the XML document received from JSimpleCalculator.java.
    - Returns the XML generated by XML2CALC.cbl to JSimpleCalculator.java.

Reengineering and Reuse of Legacy Software

Software Reengineering/Reuse Exercise (cont'd)

- Steps in the reengineering and reuse exercise (cont'd):
  - Write a COBOL program XML2CALC.cbl:
    - Marshalls XML from Java2CblXmlBridge.cpp into SMPLCALC-INTERFACE.
    - Invokes SMPLCALC.cbl, passing SMPLCALC-INTERFACE by reference.
    - Marshalls SMPLCALC-INTERFACE back to XML before returning to Java2CblXmlBridge.cpp.
  - Compile XML2CALC.cbl and link it with the object code for SMPLCALC.cbl (SMPLCALC.obj).

Reengineering and Reuse of Legacy Software

Software Reengineering/Reuse Exercise (cont'd)

- Steps in the reengineering and reuse exercise (cont'd):
  - Create a DLL to be loaded by JSimpleCalculator.java by compiling and linking Java2CblXmlBridge.cpp with the object code for XML2CALC.cbl.
  - Update JSimpleCalculator.java to use the JAXB marshalling code to send/receive XML through the JNI layer and display the results.

Reengineering and Reuse of Legacy Software

Software Reengineering/Reuse Exercise (cont’d)

- Highlights of the solution code:
  - SimpleCalculator.xsd

```
* Input/Output data structure
 01 SMPLCALC-INTERFACE.
    02 SI-OPERAND-1 PIC S9(9) COMP-5.
    02 SI-OPERAND-2 PIC S9(9) COMP-5.
    02 SI-OPERATION PIC X.
       88 DO-ADD VALUE '+'.
       88 DO-SUB VALUE '-'.
       88 DO-MUL VALUE '*'.
    02 SI-RESULT PIC S9(18) COMP-5.
    02 SI-RESULT-MESSAGE PIC X(128).
```

```
<element name="SI-OPERAND-1">
    <simpleType>
     <restriction base="integer">
      <totalDigits value="9" />
     </restriction>
    </simpleType>
   </element>
 . . .
<element name="SI-OPERATION">
    <simpleType>
     <restriction base="string">
      <enumeration value="+" />
      <enumeration value="-" />
      <enumeration value="*" />
     </restriction>
    </simpleType>
   </element>
```

Reengineering and Reuse of Legacy Software

Software Reengineering/Reuse Exercise (cont’d)

- Highlights of the solution code (cont’d):
  - ISimpleCalculator.java

```
package info.reversingproject.jsimplecalculator;

public interface ISimpleCalculator {

    long doAdd(int _1stOp, int _2ndOp) throws ArithmeticException;

    long doSubtract(int fist_operand, int _2ndOp) throws ArithmeticException;

    long doMultiply(int _1stOp, int _2ndOp) throws ArithmeticException;
}
```

Reengineering and Reuse of Legacy Software

Software Reengineering/Reuse Exercise (cont'd)

- Highlights of the solution code (cont'd):
  - JSimpleCalculator.java

```
public class JSimpleCalculator implements ISimpleCalculator {
    native String smplCalcXmlInterface(String xmldoc);

    static {
    System.loadLibrary("Java2CblXmlBridge");
    }

    /**
     */
    @Override
    public long doAdd(int _1stOp, int _2ndOp) throws ArithmeticException {
    if (JSimpleCalculatorUI._DEBUG_)
        System.out.println(EOL + "[D] JSimpleCalculator.doAdd(" + _1stOp
            + ", " + _2ndOp + ")");
    SMPLCALCINTERFACE addResult = invokeXmlInterface("+", _1stOp, _2ndOp);
    return addResult.getSIRESULT().longValue();
    }
```

Reengineering and Reuse of Legacy Software

Software Reengineering/Reuse Exercise (cont'd)

- Highlights of the solution code (cont'd):
  - JSimpleCalculator.java (cont'd)

```
public SMPLCALCINTERFACE invokeXmlInterface(String calcType, int _1stOp,
    int _2ndOp) {
if (JSimpleCalculatorUI._DEBUG_)
    System.out.println(EOL
        + "[D] JSimpleCalculator.invokeXmlInterface(" + calcType
        + ", " + _1stOp + ", " + _2ndOp + ")");
SMPLCALCINTERFACE inputData = new SmplCalcJaxbFactory()
    .createSMPLCALCINTERFACE();
inputData.setSIOPERATION(calcType);
inputData.setSIOPERAND1(BigInteger.valueOf(_1stOp));
inputData.setSIOPERAND2(BigInteger.valueOf(_2ndOp));
inputData.setSIRESULTMESSAGE("");
inputData.setSIRESULT(BigInteger.valueOf(0));
String inputXml = SmplCalcJaxbMarshaller.serializeXML(inputData);
SMPLCALCINTERFACE outputData = SmplCalcJaxbMarshaller
    .loadXML(outputXml);
return outputData;
}
```

Reengineering and Reuse of Legacy Software

Software Reengineering/Reuse Exercise (cont'd)

- Highlights of the solution code (cont'd):
  - Java2CblXmlBridge.c

```
jstring JNICALL Java_info_reversingproject_jsimplecalculator_JSimpleCalculator_smplCalcXmlInterface
  (JNIEnv *env, jobject parent_obect, jstring xml_doc)
{
  // Get input XML document passed from Java
  // omitted...
  const char *xml_input = (*env)->GetStringUTFChars(env, xml_doc, &iscopy);
  int xml_len = strlen(xml_input);
  // Allocate XML I/O buffer and copy input XML
  xml_buffer = (char*)malloc(32767);
  memset(xml_buffer, 0x00, 32767); // Initialize
  memcpy(xml_buffer, xml_input, xml_len);
  // Free JNI memory used for MBCS to SBCS conversion
  (*env)->ReleaseStringUTFChars(env, xml_doc, &iscopy);
  // call COBOL to XML marshalling layer, passing XML I/O buffer
  cobinit(); // Initialize Micro Focus COBOL runtime
  XML2CALC(&xml_len, xml_buffer); // Call COBOL
  // Null terminate XML returned from COBOL
  // omitted...
  // Allocate UTF version of XML to return to Java
  output_xml = (*env)->NewStringUTF(env, xml_buffer);
  // Free XML I/O buffer
  free(xml_buffer);
  // Return XML generated by COBOL as Java String
  return output_xml;
}
```

Reengineering and Reuse of Legacy Software

Software Reengineering/Reuse Exercise (cont'd)

- Highlights of the solution code (cont'd):
  - XML2CALC.cbl

```
 LINKAGE SECTION.
 01 XML-DOC-LEN  PIC S9(9) COMP-5.
 01 XML-DOC-TXT  PIC X(32767).
 PROCEDURE DIVISION USING XML-DOC-LEN XML-DOC-TXT.
 MAINLINE SECTION.
* Parse XML into SMPLCALC-INTERFACE
     XML PARSE XML-DOC-TXT(1:XML-DOC-LEN)
         PROCESSING PROCEDURE XML-HANDLER
     END-XML
* Invoke legacy COBOL application SMPLCALC
     CALL 'SMPLCALC' USING SMPLCALC-INTERFACE
* Generate XML from SMPLCALC-INTERFACE
     XML GENERATE XML-DOC-TXT FROM SMPLCALC-INTERFACE
         COUNT IN XML-DOC-LEN
     END-XML
* Return to client program
     GOBACK
     .
```

Results (cont'd)

Reengineering and Reuse of Legacy Software (cont'd)

- Sample run of solution code:

C:\WINDOWS\system32\cmd.exe

```
****************************************************
** Program: Java Front-end to COBOL Calculator    **
** Purpose: Demonstrate reengineering and reuse   **
**          of a COBOL program from Java by       **
**          establishing an XML bridge leveraging **
**          JAXB, JNI, and COBOL XML support.     **
** Author:  Teodoro Cipresso                      **
**          tcipress@hotmail.com                  **
****************************************************

Select a task from the following menu:

(1) Addition
(2) Subtraction
(3) Multiplication
(4) Toggle Debug OFF
(5) Quit Program

Specify selection: 1

Specify integer operand #1: 512

Specify integer operand #2: 512_
```

Figure 55. Reuse of COBOL from Java using JAXB, JNI, and COBOL XML Support.

Reengineering and Reuse of Legacy Software

Software Reengineering/Reuse Exercise (cont'd)

- Sample run of solution code:

```
C:\WINDOWS\system32\cmd.exe
(3) Multiplication
(4) Toggle Debug OFF
(5) Quit Program

Specify selection: 1

Specify integer operand #1: 512

Specify integer operand #2: 512

[D] JSimpleCalculator.doAdd(512, 512)

[D] JSimpleCalculator.invokeXmlInterface(+, 512, 512)

[D] SmplCalcJaxbMarshaller.serializeXML()

[D] SmplCalcJaxbMarshaller.serializeXML().xmlDoc[<?xml version="1.0" encoding="U
TF-8" standalone="yes"?><SMPLCALC-INTERFACE><SI-OPERAND-1>512</SI-OPERAND-1><SI-
OPERAND-2>512</SI-OPERAND-2><SI-OPERATION>+</SI-OPERATION><SI-RESULT>0</SI-RESUL
T><SI-RESULT-MESSAGE></SI-RESULT-MESSAGE></SMPLCALC-INTERFACE>]

[D] JSimpleCalculator.invokeXmlInterface(): Before call to Java2CblXmlBridge

[D] JSimpleCalculator.invokeXmlInterface(): After call to Java2CblXmlBridge

[D] SmplCalcJaxbMarshaller.loadXML().xmlDoc[<SMPLCALC-INTERFACE><SI-OPERAND-1>51
2</SI-OPERAND-1><SI-OPERAND-2>512</SI-OPERAND-2><SI-OPERATION>+</SI-OPERATION><S
I-RESULT>1024</SI-RESULT><SI-RESULT-MESSAGE>Completed successfully</SI-RESULT-ME
SSAGE></SMPLCALC-INTERFACE>]

[***] COBOL addition result: 1024
```

End