Integrating COBOL with Java in the IMS environment

Richard Tran – richtran@us.ibm.com

An IBM Proof of Technology

Powered by IMS Development at Silicon Valley Lab, California

Presentation Agenda

- The benefits of integrating COBOL with Java
- Object-Oriented COBOL: Everything you probably already know
- Extending COBOL to Java
- Using SQL in Java for IMS Database access
- Define system requirements for interoperability

Why does Java matter to me?

- COBOL code invoking Java code
  - Leverage a larger pool of resources and technology!
  - Reduce redundant development
- Java code invoking COBOL code
  - Leverage a larger pool of resources and technology!!
  - COBOL developers can be freed up to focus on high performance applications or new application development

The benefits of integrating COBOL and Java

- Preserving COBOL code makes good business sense
  - Saving $100 per line of code
- No need for “Rip and Replace”
  - COBOL applications can be extended to Java
- Java is well known to new programmers
  - Taught in 87% of universities in 2000, Gartner
  - High School Computer Science Advance Placement exams are in Java
- Makes COBOL application programming more relevant
  - Allows Java developers to bring back value in existing COBOL applications

What is object-oriented COBOL?

- A COBOL syntax that enables COBOL and Java interoperation within an address space. This means that:
  - Java can invoke COBOL class methods
  - COBOL can invoke Java
- Implementation is based on the Java Native Interface (JNI)
  - COBOL INVOKE statement maps onto Java JNI calls
  - COBOL class methods definitions define Java native methods
- Documentation and assistance in mapping Java data types to and from COBOL
- Support for JNI programming in COBOL
  - COBOL COPY file is analogous to jni.h and enables access to JNI callable services

COBOL and Enterprise Java

- Java developers can define enterprise applications through Enterprise Java Beans (EJBs)
  - Persistence
  - Transaction processing
  - Concurrency control
  - Events
  - Security
  - Remote Procedure Calls
- Object-oriented COBOL can access EJBs to leverage these Java enterprise applications

COBOL and Java interoperability: not just IMS

- z/OS Unix
  - Including WebSphere Application Server
- z/OS Batch
- IMS Java dependent regions
  - JMP - Java Message Processing region
  - JBP - Java Batch Processing region
- Windows
  - Windows COBOL component of Rational Developer for z/Series
- AIX
  - IBM COBOL for AIX

What is an Object?

- An *object* (sometimes called a *class*) is a collection of attributes and methods
  - A *attribute* is a characteristic of the object
  - A *method* is the action an Object can perform

Employee Class

Attributes:

Salary

Department

Methods:

Work

Eat lunch

COBOL client-side syntax

- Declare referenced class and full external class name:

```
Configuration section.
Repository paragraph.
Class Employee is 'com.acme.Employee'.
```

- Declare object reference:

```
01 anEmployee usage object reference Employee.
```

- Create instance object:

```
Invoke Employee New using by value id
  returning anEmployee
```

- Invoke instance method:

```
Invoke anEmployee 'payRaise'
  using by value amount
```

- Invoke static method:

```
Invoke Employee 'getNbrEmployees'
  returning totalEmployees
```

Class Inheritance

- A way of forming new classes based on existing classes
- New class inherits attributes and methods of base class
- Example: Manager class based on an Employee class

Employee Class

Attributes:

Salary

Department

Methods:

Work

Eat lunch

Manager Class

Attributes:

Salary

Department

Methods:

Work

Eat lunch

Hire

COBOL native method - syntax

```
Identification Division.
Class-id. Manager inherits Employee.
Environment Division.
Configuration section.
Repository.
Class Manager is 'com.acme.Manager'
Class Employee is 'com.acme.Employee'.
Identification division.
Object.
Procedure Division.
  Identification Division.
  Method-id. 'Hire'.
  Data Division.
  Linkage section.
  01 anEmployee usage object reference Employee.
  Procedure Division using anEmployee.
     …
  End method 'Hire'.
End Object.
End class Manager.
```

Nested Divisions

COBOL methods can be overloaded

```
Identification Division.
Class-id. Account inherits Base.
  ...
 Identification Division.
 Method-id. 'credit'.
 Data Division.
 Linkage section.
 01 amount pic S9(9) binary.
 Procedure Division using
    amount.
   ...
 End method 'credit'
 Identification Division.
 Method-id. 'credit'.
 Data Division.
 Linkage section.
 01 amount comp-3.
 Procedure Division using
    amount.
   ...
 End method 'credit'.
End Object.
End class Account.
```

Same method name

Different parameter datatypes

Access to Java from COBOL

- Function pointers for JNI services are in the JNI Environment Structure

Access JNI Environment pointer

- New special register JNIEnvPtr

Access JNI Environment Structure and JNI callable services

```
Linkage section.
COPY 'JNI.cpy'
Procedure division.
Set address of JNIEnv to JNIEnvPtr
Set address of JNINativeInterface to JNIEnv
```

Check if an exception has been thrown by a Java routine

```
Invoke aJavaObject 'someJavaMethod'
Call ExceptionOccurred          ←this is a JNI function pointer
  using by value JNIEnvPtr
  returning exceptionObject
If exceptionObject not = null
  Display 'Caught an unexpected exception'
  Call ExceptionClear using by value JNIEnvPtr
  Invoke exceptionObject 'PrintStackTrace'
  Goback
End-if
```

JNI services for string data

- Unicode-oriented JNI services for Strings, part of the standard SDK:

NewString	GetStringChars
GetStringLength	ReleaseStringChars

  - Convert between Java String objects and COBOL Unicode data

    (PIC N(*n*) USAGE NATIONAL)
  - Access these services with CALL *function-pointer* statements
    - Function pointers are in the JNI Environment Structure

- EBCDIC-oriented services, provided by IBM Java 2 SDK for z/OS:

NewStringPlatform	GetStringPlatformLength	GetStringPlatform

  - Convert between Java String and COBOL alphanumeric data

    (PIC X(*n*) USAGE DISPLAY)
  - Access CALL '*literal*' statements
    - These services are DLLs

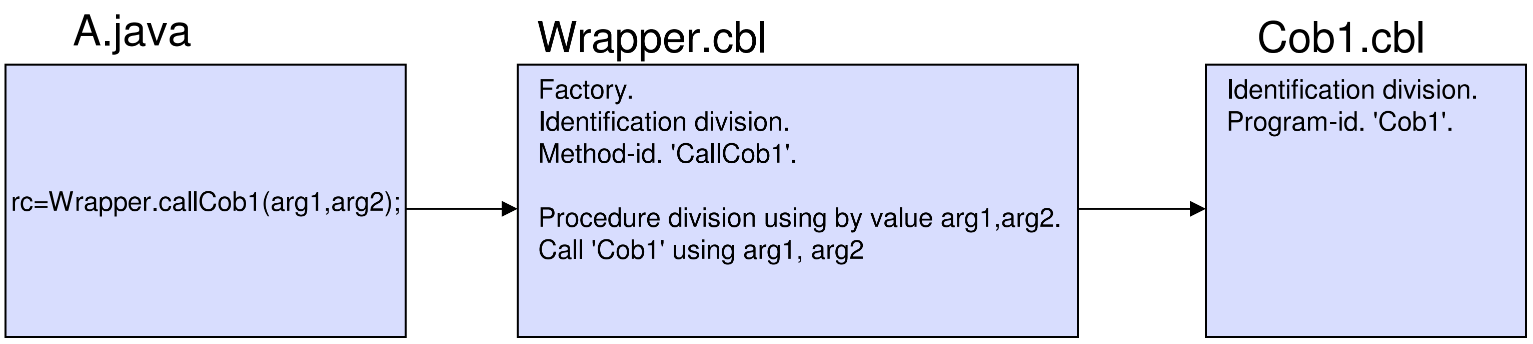
Interoperable data types for method parameters

Java	**COBOL**
boolean	01 B pic X. 88 B-false value X'00'. 88 B-true value X'01' through X'FF'.
byte	Pic X or Pic A
short	Pic S9(4) usage binary or comp-5
int	Pic S9(9) usage binary or comp-5
long	Pic S9(18) usage binary or comp-5
float	Usage comp-1
double	Usage comp-2
char	Pic N usage national
class types (object references) including strings and arrays	Usage object reference *class-name*

Accessing existing procedural COBOL code from Java

- What about our preexisting procedural COBOL?
- Write an OO COBOL wrapper class for the existing procedural COBOL program
- Define a Factory method containing a CALL to the COBOL program
- Java client uses a static method invocation to invoke the wrapper, e.g.

```
rc=Wrapper.callCob1(arg1,arg2);
```



```
rc=Wrapper.callCob1(arg1,arg2);
```

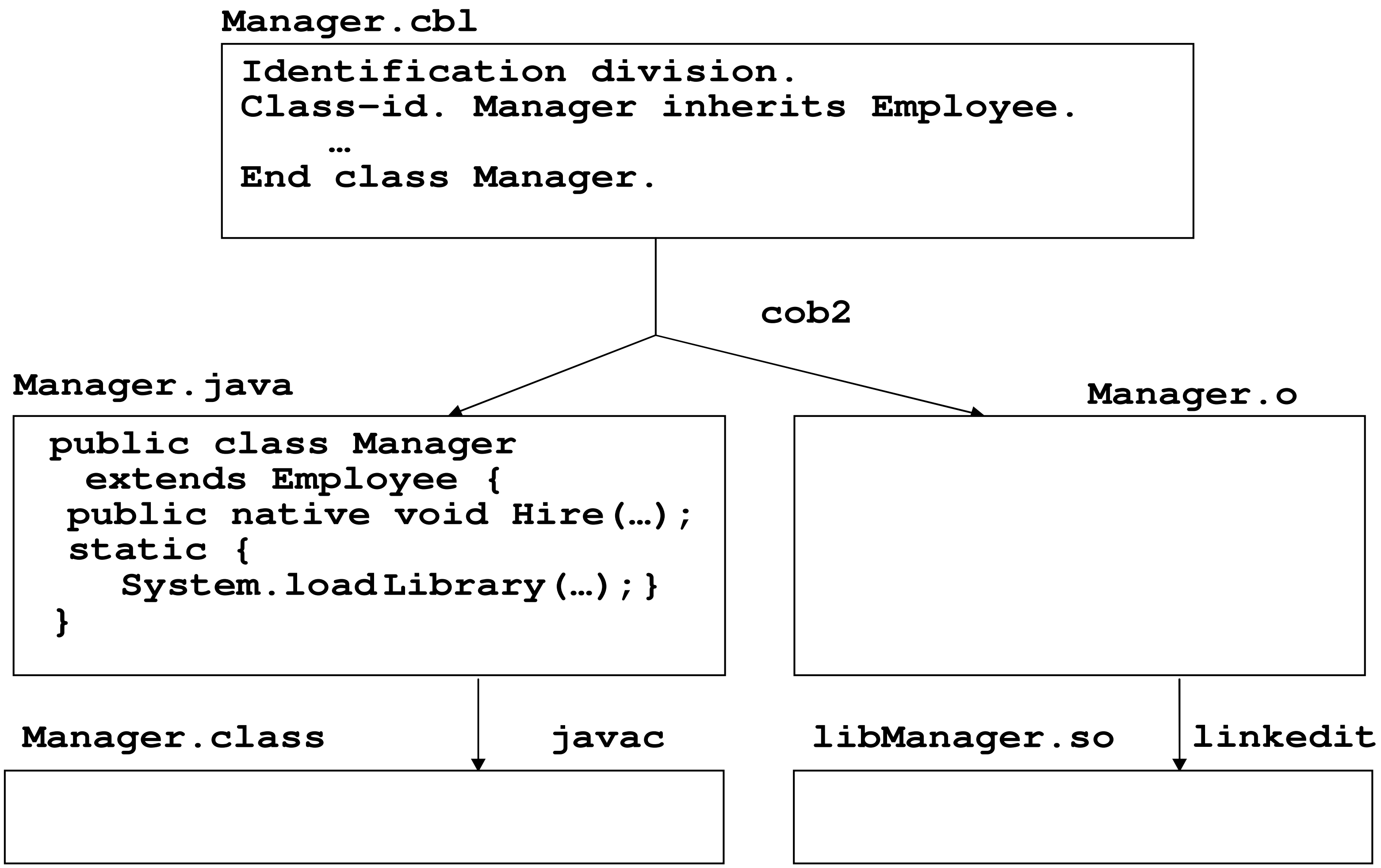
```
Factory.
Identification division.
Method-id. 'CallCob1'.

Procedure division using by value arg1,arg2.
Call 'Cob1' using arg1, arg2
```

```
Identification division.
Program-id. 'Cob1'.
```

Compile and link of COBOL class definition

- Compile of COBOL class definition generates two outputs:
  - COBOL object program implementing native method(s)
  - Java class source that declares the native methods and manages DLL loading
- COBOL object program is linked to form DLL: lib*classname*.so
- Java class is compiled (with javac) to form *classname*.class



```
Identification division.
Class-id. Manager inherits Employee.
         …
End class Manager.
```

```
public class Manager
  extends Employee {
 public native void Hire(…);
 static {
    System.loadLibrary(…);}
}
```

Key points on COBOL and Java interoperability

- Object Oriented COBOL and Java can be easily integrated
- No need to alter old procedural COBOL to leverage this interoperation
  - Can be done with a few lines of code by creating a Object Oriented COBOL wrapper
- Bridges the gap between different skill sets
  - Allows more synergy between COBOL and Java developers
- Providing additional value to existing COBOL code repositories

Why is SQL important?

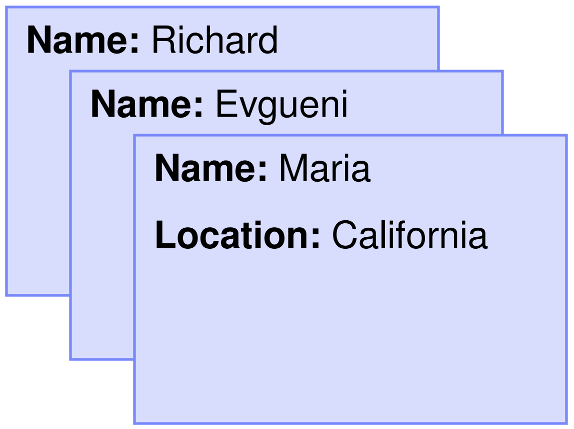
- Language for querying relational databases
  - IMS V11 supports a subset of SQL operations
- Vendor independent
  - SQL programs can be moved from one DB to another with minimal conversion
- Portable
  - Used in mainframes, workstations, and handheld devices
- Very Popular
  - SQL is the 11th most popular programming language (Tiobe Index, June 2009)

SQL and DLI

- Both are database languages
- SQL is for Java developers and DLI is for COBOL developers
  - Line is fuzzy due to Java-COBOL Interoperability
- Most SQL and DLI statements have a one-to-one correspondence
  - Meaning one line of SQL is equivalent to one line of DLI

Things SQL make easier

Segment: Employees



- SQL makes manipulating multiple instances of segments easier
- To retrieve all instances of all Employees
  - SQL: Select * From Employees
  - COBOL: GU Employees, GN Employees, GN Employees, etc.
  - SQL requires 1 line of code, COBOL requires n lines where n is the # of employees
- To update all instances of Employees
  - SQL: Update Employees Set Location='New York'
  - COBOL: GHU, REPL, GHN, REPL, GHN, REPL, etc.
  - SQL requires 1 line of code, COBOL require 2n lines of code

Key Points on SQL

- SQL is very popular and well known among developers
- College graduates with Java knowledge will be able to leverage the native Java support for SQL
- Employees with background in DB2, Oracle, or any other relational database will have knowledge of SQL
- Great for mixed customer environments as it simplifies database usage
  - e.g., IMS and DB2
- Simplifies handling of multiple instances of an IMS data segment compared to DLI
- Brings more value to Java-COBOL interoperability as Java developers can take more of the tedious data manipulation work off of the COBOL developers.

Getting started with COBOL and Java interoperability

- Ensure you have the Java 2 Technology Edition SDK installed
  - SDK 1.4,
- Ensure that the optional HFS components of Enterprise COBOL V3 have been installed
- See the sample OO application and makefile shipped with COBOL in /usr/lpp/cobol/demo/oosample. Try compiling and running this application.

Presentation summary

- The benefits of integrating COBOL with Java
- Object-Oriented COBOLExtending COBOL to Java
- Using SQL in Java for IMS Database access
- Define system requirements for interoperability

Questions