



# IMS and Java for Application Modernization

Barbara Klein  
IMS Product Manager  
IBM Silicon Valley Lab, San Jose, California  
bk@us.ibm.com

GSE UK Conference

*November 4-5, 2009*

© 2009 IBM Corporation

This presentation discusses use of IMS and Java to modernize your applications.

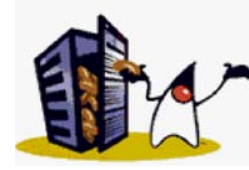
## Agenda

- Java basics
- IMS Java basics
- IMS Java Applications
- IMS Database Access
- DB2 Access
- Application Interoperability
- Technical Considerations
- Customer Examples
- Summary



I'll be taking you quickly through some Java basics, introduce you to IMS Java application support, Java access to IMS DB and DB2 databases, interoperability of Java applications with other applications, some technical considerations in doing all this and some examples of customers using Java with IMS.

## Java Basics

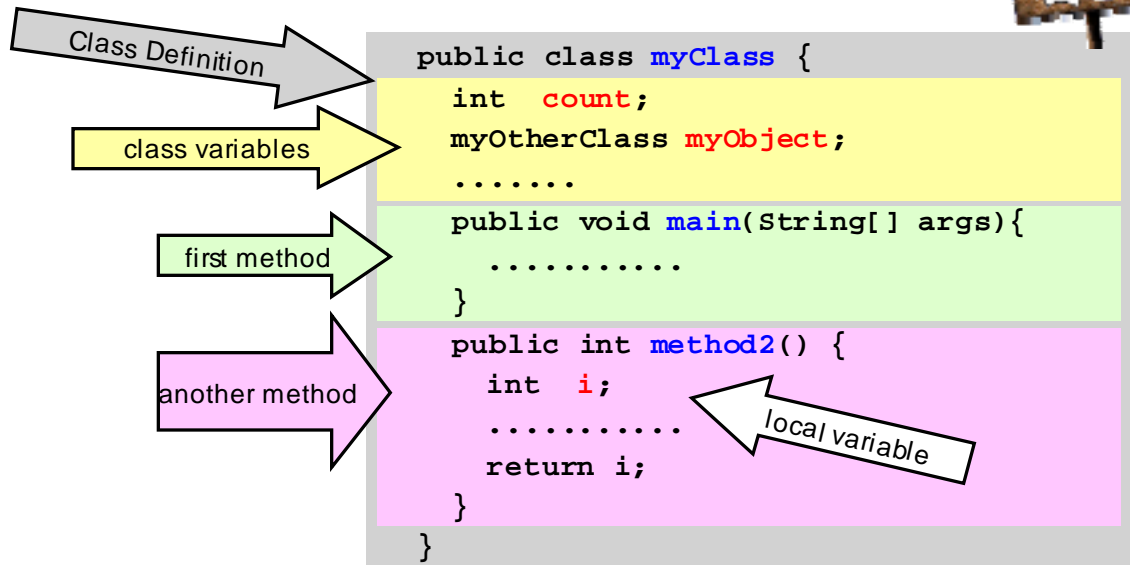


- It's another programming language!
- It's Object Oriented
  - A Program is written as a set of “Classes”
- A Class is made up of variables and methods
  - A Class often relates to an entity in the real world (e.g. Customer, Account, input message, etc), but can also be defined for programmer convenience
- An Object is a named occurrence of a class with values in the variables
- A method can create objects of its own class or other classes
- A method can execute methods of its own class or other classes, and can reference data in the same or different classes/objects

First, lets look at Java basics.

Java is an Object oriented programming language that uses classes and methods to define objects.

## Java Basics ...



- Execution of a Java program starts by running the “main” method of the specified class
- Method definition includes the return parameter (or “void”) and the input parameters (in any)

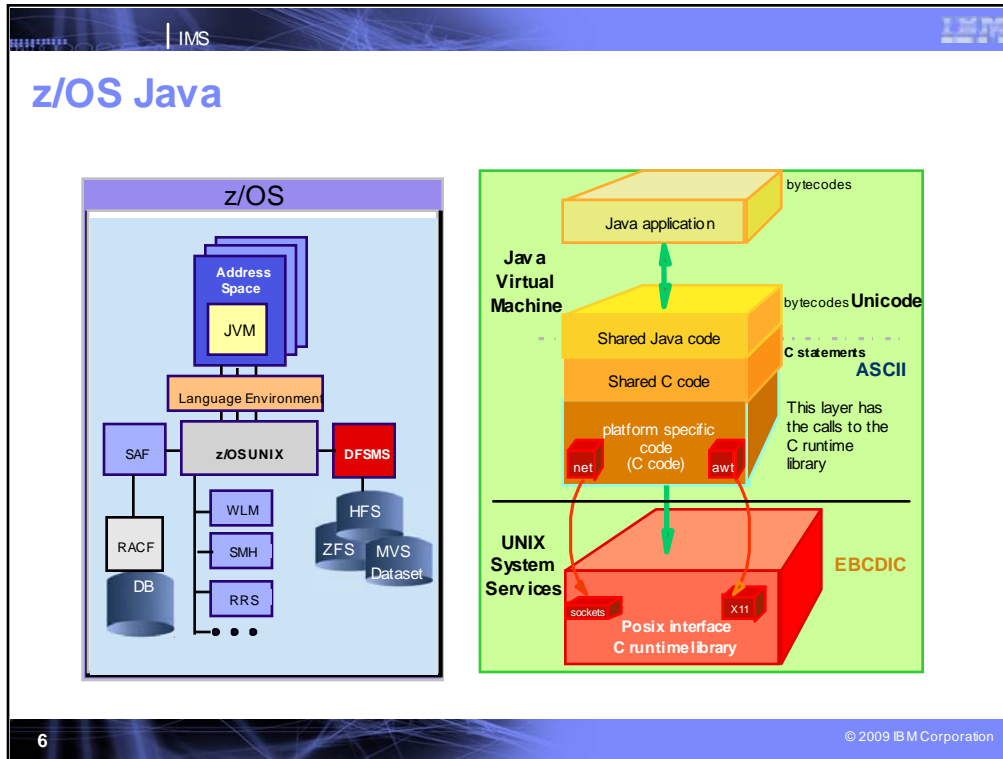
Java programs consist of definitions, variables and methods. Execution of a Java program starts by running the main program of the specified class. This is a sample application, defining class definition and variables, and the methods. Methods can inherit variables or use their own local variables.

## Java Class Libraries



- The Java language is quite small and simple
  - one of its strengths!
  
- But it comes with a huge number of pre-written classes
  - ready-to-use java methods (and associated variables and constants) for
    - file access
    - working with a GUI
    - data conversion
    - mathematical functions
    - accessing URLs
    - getting date/time
    - etc etc etc.
  
- Classes are bundled in named “packages”

The Java class library is simple, and includes a large number of pre-written classes of function in a number of packages



z/OS Java provides the facilities and environment for running Java applications.

z/OS provides a Java Virtual Machine to run the Java applications, and the System Services to support it. From the JVM z/OS provides access to Unix services and vice versa. Java applications go thru C code to get the Unix services. The System provides Java Native Interface for this. IMS will use the Java Native Interface for DL/I access. The Hierarchical File server is the equivalent of windows folders.

IMS

## z/OS Java ...

- Java Record I/O
  - Class library, similar to java.io
  - Provides record-oriented access on z/OS
    - VSAM data sets (KSDS only), Non-VSAM record-oriented data sets, System Catalog, PDS directory
    - DDName and GDGs support
    - SPACE and DISP parameter support
    - Navigational support for HFS directories
  - zAAP eligible
  - <http://www-03.ibm.com/servers/eserver/zseries/software/java/jrio/overview.html>
- JZOS
  - A set of tools that includes a native launcher for running Java applications directly as batch jobs or Started Tasks
  - A set of Java methods that make access to traditional z/OS data and key system services directly available from Java applications
    - Additional system services include console communication, multiline WTO (write to operator), and return code passing capability
    - JZOS provides facilities for flexible configuration of the run-time environment, and it allows intermediate data to be seen via SDSF
    - Java applications can be fully integrated as job steps in order to augment existing batch applications

7

© 2009 IBM Corporation

z/OS Java provides

- a Java record I/O which consists of a class library and record oriented access on z/OS,
- JZOS, a set of tools for running Java applications as batch or started tasks and a set of methods for making access to traditional data and system services available from Java applications

## Why Use Java ?

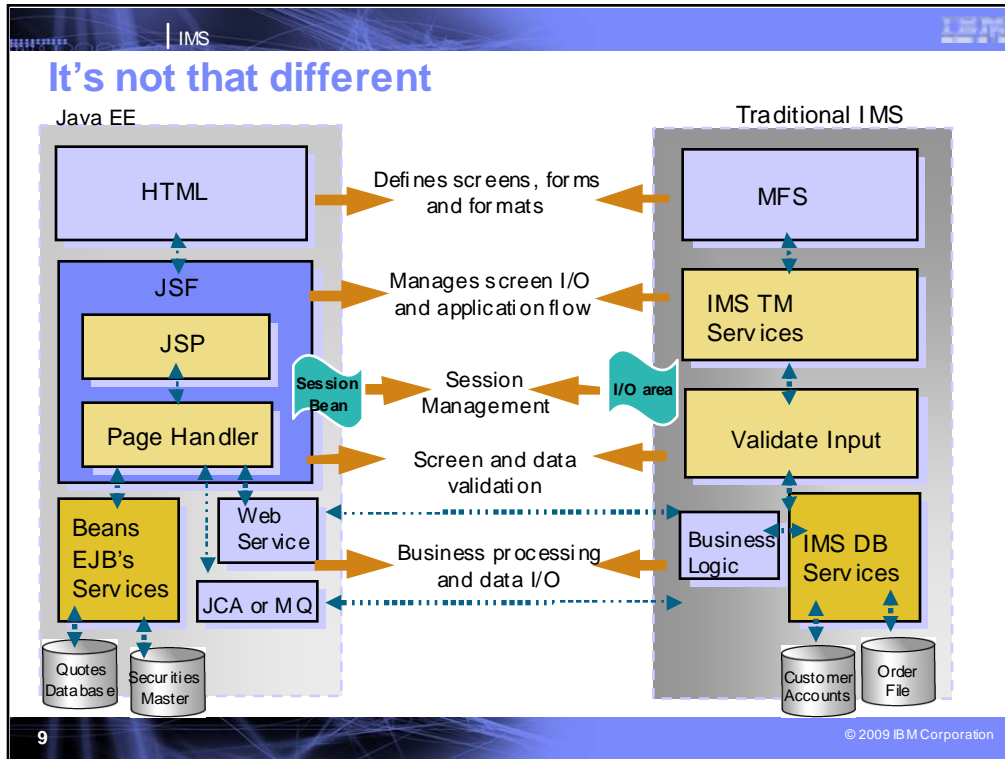
- Rapid Application Development
  - Reduces Total Cost of Ownership (TCO) and Total Time to Value (TTV)
- Exploit existing marketplace, industry-sanctioned standards
  - These are the slowest changing and most persistent
  - JDBC and Java EE (was J2EE) are standards which help to minimize specific back-end knowledge of IMS
- Exploit new and abundant skills in the marketplace and help customers avoid problems due to the loss of traditional zOS skills
- CPU cost has been reduced significantly with successive releases of IBM's SDK
  - And Java MIPS are very cheap with System z Application Assist Processor (zAAP)

Java allows you to reduce redundant development efforts, to take advantage of industry standards and tools, and to leverage a larger pool of resources and technology.

Java is becoming improved for efficient and more economic use. And Java MIPS can also be run on the zAAP processors.

IBM Java support for the new IBM zSeries Application Assist Processor (zAAP) for the IBM zSeries servers allows Java workloads to transparently execute on the zAAP processors without requiring application change. This allows you to integrate and run Java workloads on the same server as your database at a significantly lower total cost of ownership than previously possible. This helps reduce overall cost of computing for Java applications, increase system productivity by reducing the demands on general purpose processors, and makes capacity available for other workloads.





The standard Java runtime environment can be compared to the traditional IMS environment.

## Agenda

- Java basics
- IMS Java basics
- IMS Java Applications
- IMS Database Access
- DB2 Access
- Application Interoperability
- Technical Considerations
- Customer Examples
- Summary



So let's look next at IMS and Java basics

## IMS Java Basics

- A library of classes that allows you to write programs in java that access IMS messages and/or IMS DBs (and DB2)
  - Java IMS TM API – Offers IMS transaction processing support
  - Java IMS DB API - Offers IMS database access support using the IMS Java hierarchic database interface
  - JDBC driver for IMS
    - Implements Industry standards JDBC 2.0 API
    - Offers SQL and XQuery support for IMS DB
- A Java IMS application runtime environment in special dependent regions
  - JMP and JBP regions
- Language interoperability
  - Ability for Java and OO COBOL/PL/I to invoke one another within the same transaction
- DB2 interoperability
  - Access an IMS and DB2 database in the same transaction
    - Use either the DB2 JDBC or SQLJ drivers to access DB2
    - IMS and DB2 activity in the same unit of work
- XML support in IMS databases
  - An extension of the JDBC interface can be used for storing and retrieving XML documents in IMS databases

### Java is the base for new application development and connectivity.

IMS Java application support has enhanced the ability of our customers and business partners to provide integrated on demand business application development with IMS. The object of this function is to provide support for you to write Java applications and run them as IMS applications using WebSphere workstation and host tools for development and testing.

IMS Java provides a library of classes for access to IMS messages and data.

IMS provides access to IMS TM message queues and to IMS DB and DB2 data through JDBC.

Support is also provided for IMS DB access from CICS/zOS Java applications, DB2/z/OS Java Stored procedures and WebSphere z/OS applications, opening IMS DB up to better integration and use across platforms and across application environments. New Java Region Types were also being provided to replace initial support utilizing the High Performance Java Compiler. The newer support enhances the initial Java support to run with the new Scalable JVM, providing enhanced tool support for developing these Java applications to run in IMS. New Java Tooling is provided, along with Java, Cobol, PL/I, and XML Interoperation. IMS support for Java is augmented by IBM's Eclipse-based Application Development tools for JEE programmers, as well as a broad array of AIM tools designed to facilitate building Innovative SOA-based applications that combine the best of the z world with the best of the Java world.

The Java class library is simple, and includes a large number of pre-written classes of function in a number of packages initially for the classic drivers (shown below) and has since been providing more for the universal drivers:

**com.ibm.ims.base** - Provides classes for basic IMS Java functions and for problem determination.

**com.ibm.connector2.ims.db** - Provides classes for connecting to IMS databases from WebSphere Application Server for z/OS.

**com.ibm.ims.application** - Provides classes for processing IMS messages, and performs commits and rollbacks for JMP and JBP applications.

**com.ibm.ims.db** - Provides classes for the JDBC driver and for the IMS Java hierarchical database interface.

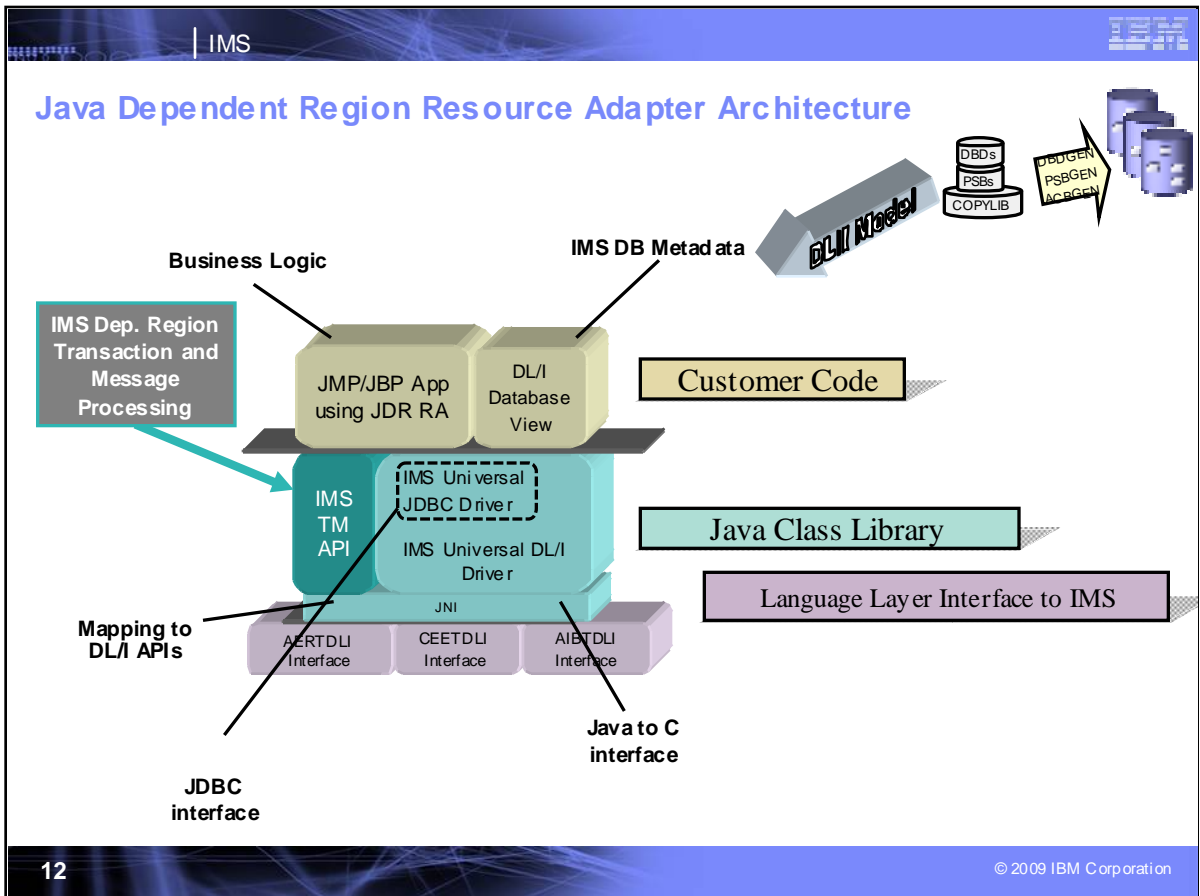
**com.ibm.ims.rds** - Provides classes for client-side WebSphere Application Server support of remote database services

**com.ibm.ims.rds.host** - Provides classes for server-side WebSphere Application Server support of remote database services

**com.ibm.ims.rds.util** - Provides classes for storing data that is passed between the client and server components for remote data access support

**com.ibm.ims.xmls** - Provides classes for storing and retrieving XML in Java applications .

Additional information about the IMS Java class library is in the IMS Java API Specification (Javadoc), available off the [www.ibm.com/ims](http://www.ibm.com/ims) link to the IMS Java page.



## Java Dependent Region Resource Adapter Architecture

The IMS Java dependent region (JDR) resource adapter is a set of Java classes and interfaces that support IMS database access and IMS message queue processing within Java batch processing (JBP) and Java message processing (JMP) regions. To access IMS message queues from your JMP and JBP applications, use the IMS Java dependent region resource adapter. To access IMS databases from your JMP and JBP applications, you can also use the IMS Universal JDBC driver and the IMS Universal DL/I driver. This architecture, based on the IMS Open Database solution, supports Java application development from simple up to advanced Java and IMS skill levels.

Along with the JDR resource adapter and IMS Universal drivers, your Java application uses metadata classes, which describe the IMS database view to your Java application. To generate these classes, use the IMS Enterprise Suite DLIModel utility plug-in. In addition to generating the metadata classes, this utility generates an easy-to-read report of the IMS database and an XML description of the database.

Under the covers, the IMS solutions for Java development use the Java Native Interface (JNI) to access lower-level C interfaces to get to the IMS functions.

## Agenda

- Java basics
- IMS Java basics
- IMS Java Applications
- IMS Database Access
- DB2 Access
- Application Interoperability
- Technical Considerations
- Customer Examples
- Summary



We now look closer at the IMS Java applications

## IMS TM Java Dependent Regions

- Two types of IMS Java Dependent Regions (JDR)
  - **Java Message Processing (JMP)** region
    - Analogous to an MPP region for message driven applications
    - New IMSJMP JOB that EXECs the new DFSJMP procedure
    - DFSJMP procedure added to IMS.PROCLIB
      - Similar to the DFSMPR procedure for MPPs
  - **Java Batch Processing (JBP)** region
    - Analogous to a non-message driven BMP or non-message driven Java applications
    - New IMSJBP JOB that EXECs the new DFSJBP procedure
    - DFSJBP procedure added to IMS.PROCLIB
      - Similar to the IMSBATCH procedure for BMPs
- JMP regions and JBP regions can run applications that are written in Java, object-oriented COBOL and PL/I, or a combination of the two.
- JMP regions and JBP applications can access DB2 for z/OS databases as well as IMS databases.

### JVM Support in IMS Dependent Regions

When IMS is your transaction manager, Java application programs run in IMS dependent regions that have a persistent reusable Java Virtual Machine (JVM). Java applications can run in two types of regions: Java Message Processing (JMP) and Java Batch Processing (JBP).

#### JMP (Java Message Processing)

For message-driven Java applications

New IMSJMP JOB that EXECs the new DFSJMP procedure

DFSJMP procedure added to IMS.PROCLIB

Similar to the DFSMPR procedure for MPPs

#### JBP (Java Batch Processing)

For non-message driven Java applications

New IMSJBP JOB that EXECs the new DFSJBP procedure

DFSJBP procedure added to IMS.PROCLIB

Similar to the IMSBATCH procedure for BMPs

## Java API for IMS TM

- Java library provides an API for use in Java dependent regions
  - Performs traditional transaction processing
    - Message queue processing (reading and writing)
    - Transaction demarcation
    - Program switching capabilities
      - Easy integration of Java with existing assets

Remember that since the Java APIs are built on top of existing assembler modules all traditional IMS functionality is supported via the Java libraries.

## More on the Java Dependent Regions

- Regions designed specifically to handle Java workload
  - IMS fully manages the JVM lifecycle
- The persistence model of IMS dependent regions tailors to JVM usage
  - IMS dependent regions are typically up for long periods of time
  - JVM initialized when the dependent region is brought up
    - JVM remains active and ready for Java workload until dependent region is brought down
- Shared cache feature of SDK 5/6
  - Works very well with IMS
  - Multiple dependent regions can share the same cache
  - Caches can be configured to be accessed by a certain set of applications
    - Similar to 'classes' of IMS applications

Regions designed specifically to handle Java workload

IMS fully manages the JVM lifecycle

The persistence model of IMS dependent regions tailors to JVM usage

IMS dependent regions are typically up for long periods of time

JVM initialized when the dependent region is brought up

JVM remains active and ready for Java workload until dependent region is brought down

Shared cache feature of SDK 5/6

Works very well with IMS

Multiple dependent regions can share the same cache

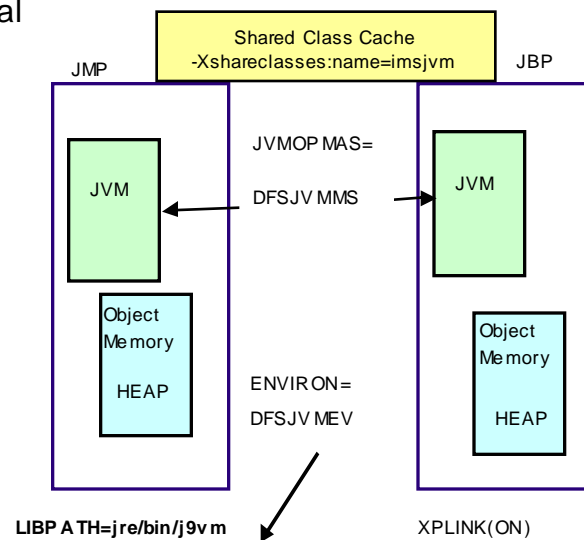
Caches can be configured to be accessed by a certain set of applications

Similar to 'classes' of IMS applications



## IMS TM Java Dependent Regions...

- Each JDR contains a Java Virtual Machine (JVM)
  - Can only connect to one PSB at a time (as always)
  - Can run any mixture of Java and Object Oriented COBOL
- To enable class sharing use `-Xshareclasses:name=` option when starting a JVM



`JVMOPMAS=DFSJVMMS`

Specifies the JVM options

`ENVIRON=DFSJVM EV`

Must contain the pathname to JVM

Must contain the pathname to the IMS Java native code

To enable class sharing use `-Xshareclasses:-name=` option when starting a JVM.

The `-name=<name>` connects a JVM to the specified name cache or creates the cache if it does not already exist.

`-Xscmx<size>[k|m|g]` Specifies cache size. This option applies only if a cache is being created and no cache of the same name exists.

Note shared class is not required.

IMS

## IMS TM Java Dependent Regions ...

- IBM SDK V5 for z/OS
  - Required for IMS 10 JMP and JBP dependent regions
  - Contains a re-engineered Java 2 virtual machine
  - Provides cache class sharing
    - Replaces persistent reusable function
    - Multiple dependent regions can share the same cache
    - Caches can be configured to be accessed by a certain set of applications
      - Similar to 'classes' of IMS applications
    - Simpler implementation
- Benefits of IBM SDK V5 for z/OS
  - Reduces virtual memory consumption
  - Reduces JVM startup time
  - Simpler setup
    - Does not require separate "trusted" and "shareable" classpaths
- IBM SDK V6 for z/OS
  - Offering improved performance

18 © 2009 IBM Corporation

z/OS delivers a complete Java 2 Software Developer Kit (SDK).

Previous versions of the SDK provided a reusable function to support transactional runtime environments like IMS. This capability allowed the Java Virtual Machine (JVM) to be initialized during IMS Java dependent region startup and to be "re-set" after the IMS application program completed processing. This avoided the overhead of loading the JVM for each IMS application program schedule.

The new SDK provides a Class Sharing capability to replace the persistent reusable function. z/OS APAR OA11519 is recommended for cache class sharing.

Benefits of SDK V5:

Uses dynamic recompilation with 5 optimization levels

The busiest methods are optimized most aggressively

Provides enhanced "garbage collector"

To delete objects which are no longer in use

Avoids JVMs having to be reset for each transaction (more later)

Provides shared class caches

Shared by multiple JVMs

## IMS TM Java Dependent Regions ...

- PCB Name or label required for any application
- Value of JAVA to be supplied for the LANG= parameter in the PSBGEN macro
  - LANG=JAVA can also be supplied in the APPLCTN macro for GPSBs
  - LANG=JAVA only required for JMP regions
  - Specifying LANG=JAVA will result in the transaction being scheduled in a Java dependent region
    - When IMS receives the name of the transaction and looks up the PSB associated with the transaction code, if JAVA is specified the transaction will be queued to execute in a Java dependent region

```
APPLCTN PSB=JAVTESTJ,PGMTYPE=TP,SCHDTYP=PARALLEL
TRANSACT CODE=JAVTRANJ,PRTY=(7,10,2),INQUIRY=NO,MODE=SNGL, X
MSGTYPE=(SNGLSEG, NONRESPONSE,1)
```

```
PHONEAP PCB TYPE=DB,DBDNAME=TELEDBD,PROCOPT=AP,KEYLEN=16
SENSEGNAM=TELE ROOT,PARENT=0,PROCOPT=AP
PSBGEN LANG=JAVA,PSBNAME=JAVTESTJ,CMPAT=YES,OLIC=YES
END
```

To schedule into JVM, the application macro is the same and the application's PSB is used. During the PSBGen, the Java language is specified, indicating that this is to be scheduled into JVM region. Class scheduling is the same as with other MPPs.

## Java Message Processing Applications

- Before your JMP application can access the message queue, you must define input and output message classes by sub classing `com.ibm.ims.application.IMSFieldMessage`
  - The subclass inherits all the data and methods of `IMSFieldMessage`
  - You add the metadata that describes the message content
- The IMS Java dependent region resource adapter provides the capability to process `IMSFieldMessage` objects.
- The following code sample shows how to define an input message:

```
public class FindCarInput extends IMSFieldMessage {
    final static DLITypeInfo[] fieldInfo = {
        new DLITypeInfo("InputMake", DLITypeInfo.CHAR, 1, 5),
        new DLITypeInfo("InputYear", DLITypeInfo.CHAR, 6, 4)};

    public FindCarInput() { /* method to create this msg object */
        super(fieldInfo, 9, false); /* "false" = not a SPA */
    }
}
```

Constructor  
Method

You do not define LL, ZZ or Trancode

This is an example of how we provide Java message classes, just as we would want to define input-output message areas. For Cobol, you define the LL-ZZ, but for IMS Java, these don't need to be provided. IMS provides these as methods, defining the field message. The Java Dependent Region resource adapter also supports scratchpad areas (SPA) for conversations with Java.

## Processing Messages in a JMP Application

- A transaction begins when the JMP application receives an input message and ends when the JMP application commits the results from processing the message.
- To get an input message, the application calls the `MessageQueue.getUnique` method.
- The following code example shows how an input message is processed in a JMP application:

```
Application app = ApplicationFactory.createApplication();
MessageQueue msg0 = app.getMessageQueue();
IOMessage inputMessage = app.getIOMessage("class://FindCarInput");
String inputMake;

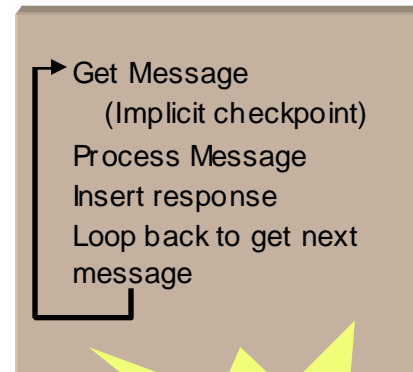
while (msg0.getUnique(inputMessage)) {
    inputMake = inputMessage.getString("InputMake");
    ...
}
```

- `getString()` is a method of the `IOMessage` interface
  - It references message fields by name defined in meta data
  - Similarly there are methods (`setString()`, etc) for building output message fields
- Similar classes are also coded to define each subsequent input message segment, output message segments, and a scratchpad area (SPA), as appropriate

When a program is scheduled, the “while” loops to get messages from the message queue to get data for the message area.

## JMP Program Structure

- IMS standard commit model no longer requires explicit checkpoint/rollback using the IMS DB Resource Adapter Transaction class
- Grouping of several classes including
  - Message segments and SPA classes
  - DLIDatabaseView class for IMS DB metadata
  - The class representing the transaction itself instantiates itself and other classes, gets the input message, controls processing, sends reply message and loops



Consistent with  
all other  
Programming Languages

IMS Java Dependent Region programming does not require an explicit checkpoint/rollback call using the IMS DB Resource Adapter Transaction class before obtaining the next input message. It operates just like other languages now. This reflects the changes of the JMP model back to the standard model, like COBOL, doing synchpoint and get next.

## Agenda

- Java basics
- IMS Java basics
- IMS Java Applications
- IMS Database Access
- DB2 Access
- Application Interoperability
- Technical Considerations
- Customer Examples
- Summary



Java database access is also provided to IMS databases

## IMS Solutions for Java Development

- IMS 11 Open Database APIs JDBC 3.0
  - IBM SDK V5 z/OS
    - CICS,DB2,WebSphere
  - IBM SDK V6 z/OS
    - IMS TM
- IMS 9,10 Java Drivers JDBC 2.1
  - IBM SDK V1.3.1 IMS 9
  - IBM SDK V1.4.2 IMS 9
  - IBM SDK V5 z/OS IMS 10

© 2009 IBM Corporation

The IMS DB Resource Adapter enabled JDBC access to IMS DB from IMS TM JMP/JBP environments, CICS Java application, DB2 Java Stored procedure, and Enterprise Java Beans running on WebSphere and z/OS environments, but initially from within the same LPAR. Now the IMS DB Resource Adapter has been extended to provide access from distributed, as well as, z/OS environments.

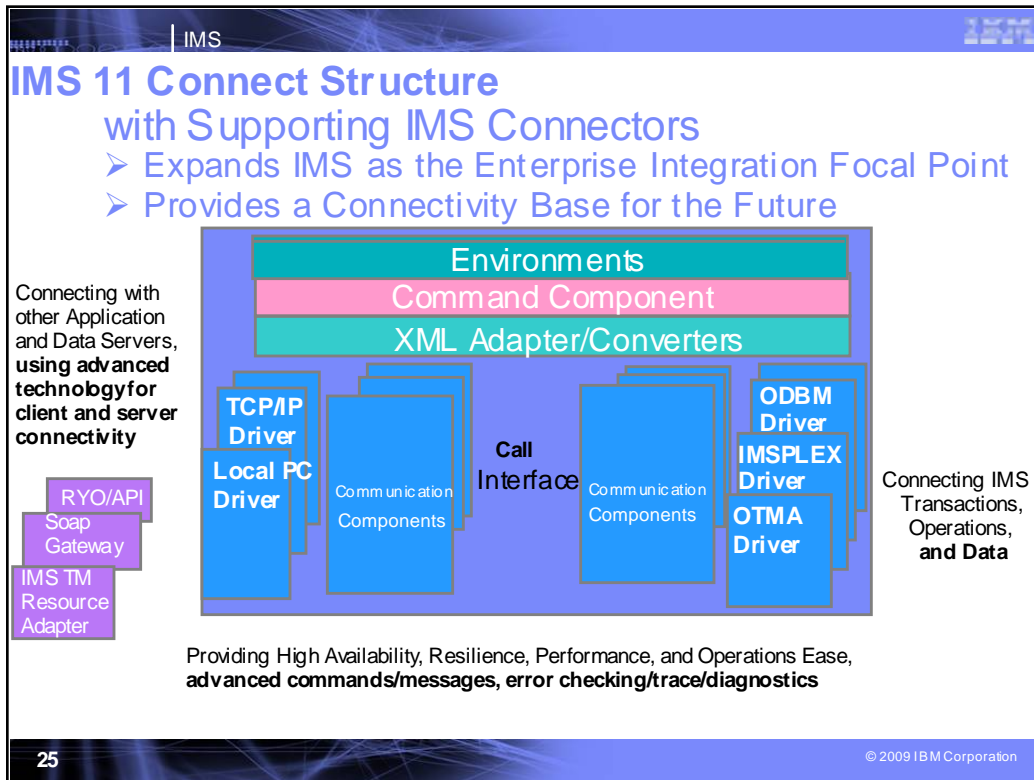
IMS V9 requires SDK V1.4.2 for JMP and JBP regions, IMS DB Resource Adapter for CICS, DB2 or WAS requires SDK V1.3.1 or higher.

IMS V10 requires SDK V5 for JMP and JBP regions, IMS DB Resource Adapter for CICS, DB2 or WAS requires SDK V1.4.2 or higher.

JDK is a subset of SDK and is what is needed for writing and running Java programs. SDK consists of Applications, debuggers, and documentation. 1.5 and 5, etc., are interchangeably used.

The IMS Universal drivers require Java Development Kit (JDK) 5.0 or later, CICS Transaction Server for z/OS Version 3, DB2 for z/OS Version 9 or DB2 UDB for z/OS Version 8, WebSphere Application Server for z/OS or WebSphere Application Server for distributed platforms Version 6.1, and JMP and JBP regions. Java Development Kit JDK 6.0 or later can also be used.





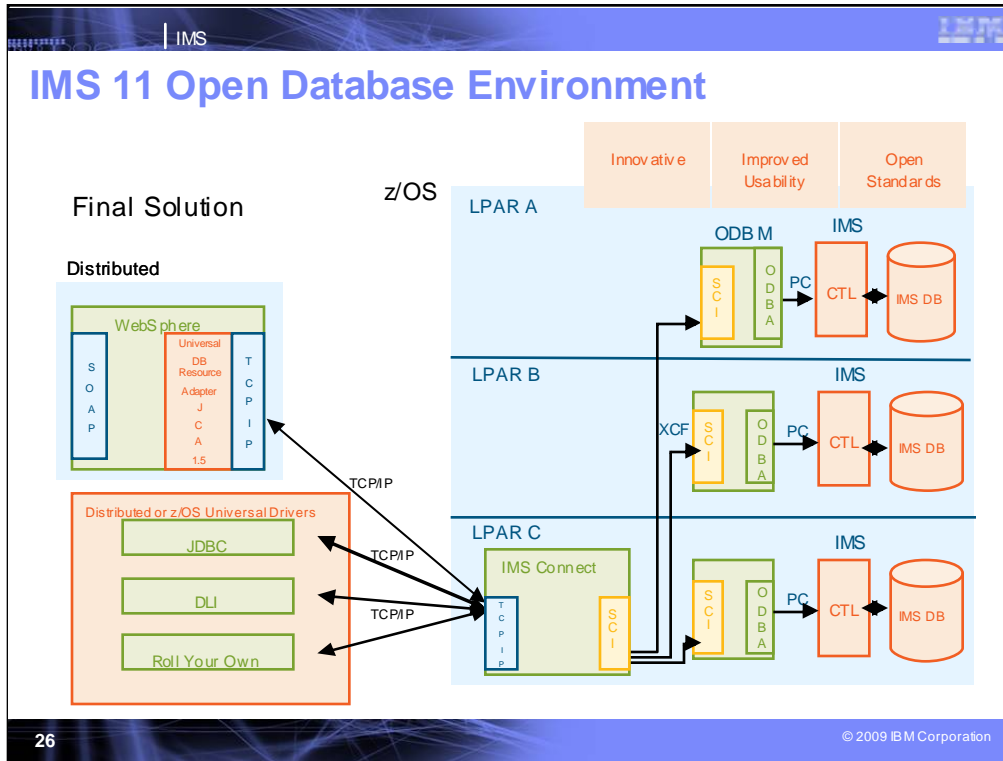
**Key Message: IMS provides and continues to enhance the integrated IMS Connect function.**

IMS Connect function is part of the overall restructure of IMS for the 21st Century and is architected as the base for all future IMS Connectivity. Much of the function of IMS Connect can also be used with earlier IMS Versions so you can start to take advantage of it before migrating your networks/applications/databases to IMS V9. The structure of IMS Connect is designed such that drivers can be interchangeable. That is, alternatives for the TCP/IP front end or OTMA back end interfaces are already being provided. These are allowing IMS to exploit newer, additional, and enhanced protocols and/or interfaces. Along with IMS Connect is provided the IMS Connector for Java for access from Java applications, SOAP Gateway and parsers, and samples for other language access as well.

With IMS Version 8, IMS extended its use of XCF for use by other IBM subsystems, such as IMS Connect, for distributed operations access through the Structured Call Interface to the Operations Manager from the DB2 Version 8 Control Center as a single point of control.

With IMS Version 9 this function was integrated in.

With this structure IMS 11 Connect has evolved to also support direct distributed database access to IMS DB from Java and other applications



This provides the ability to leverage IMS Connect as the complete gateway solution for IMS TM, Operations, and now DB. IMS Connect is augmented to be an ODBM client. This allows distributed applications to leverage the TCP/IP protocol to communicate with IMS Connect, which can then access any database in the entire IMSplex.

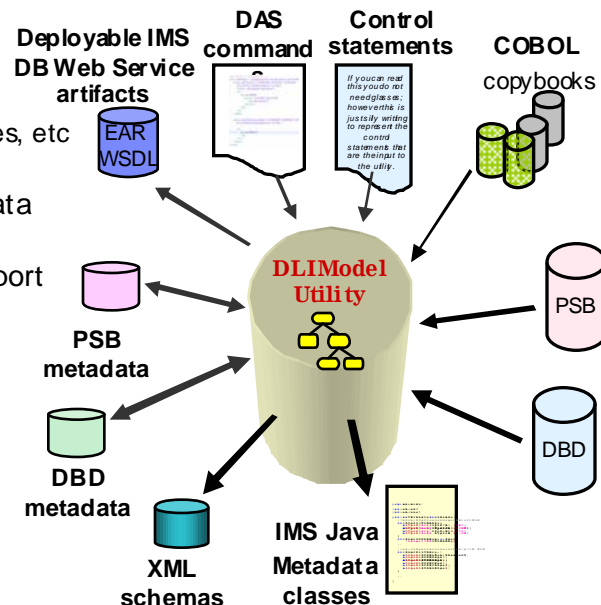
IMS Connect becomes the IMS Gateway to both IMS TM and IMS DB.

WebSphere and DB2 Stored Procedures no longer have to be on the same LPAR with IMS when they interface with the IMS ODBM (Open Database Manager) address space. The ODBM address space must be on the same LPAR with IMS due to the use of the ODBA (Open Database Access) interface.

Distributed clients would now have the option of going directly to IMS Connect for IMS DB requests. Existing DB Resource Adapter applications are unaffected by Open Database. In order to **exploit** Open Database from existing DB Resource Adapter applications, a migration to the JCA 1.5 programming model would have to be done.

## DLIModel Utility

- IMS DB visualization tool
  - Visualize an entire IMS PSB
  - Can view each PCB individually
    - Hierarchy, segments, fields, types, etc
- IMS DB metadata generation tool
  - Generates the necessary metadata consumed at runtime by IMS DB Resource Adapter, XML-DB support
    - Database metadata
    - XML schema
- Bottom up tooling approach
  - Parses PSB and DBD source
  - Optionally COBOL copybook definitions of segments
- An Eclipse 3.x plug-in



Two versions of the IMS DLIModel utility have been made available: An IMS 9/10-shipped version that runs from System Services or from the z/OS® BPXBATCH utility, and an IMS 10 and IMS Enterprise Suite version that can be web downloadable as a plug-in to Eclipse. The GUI can be installed in an Eclipse 3.0.1 or 3.0.2 level tool. It can also be installed in WebSphere Developer or z IDE.

In IMS 10, the IMS DLIModel utility has been enhanced to generate XML from PSB and DBD source. The generated XML can also be used as input to the DLIModel utility. GSAM now uses the GSAMDLIDatabaseView IMS Java class for metadata information about the GSAM database. The DLIModel Utility now supports GSAM databases.

This chart shows the inputs and outputs of the DLIModel utility. The actions of the utility are directed by control statements that you supply. PSB and DBD source members are read from their PDS or PDSE data sets and parsed by the utility to build an in-memory object model of the database structure and the PSB's view of that structure. Note that the IMS COBOL copybooks can only be processed by the GUI and the BPXBATCH utility can only process COBOL XML representations of the COBOL copybooks. The utility generates various outputs that were requested through control statements. You can specify to have an IMS Java metadata class be generated for the PSB processed, together with a corresponding easy-to-read DLIModel Java Report for the Java programmer to use. You can specify an XML description of the entire in-memory model. One description covers PSB and all DBDs processed in the run. You can also request a detailed trace file of the utility execution if one is necessary for problem resolution. The DLIModel utility produces the necessary metadata classes needed to develop IMS Java applications. However, the Java developer needs only to reference the DLIModel Java Report for information about the classes. The DLIModel Java Report summarizes the structure of the IMS databases in a way that allows you to create IMS Java applications and to code SQL queries against the databases. With the DLIModel Java Report, you do not have to interpret the syntax of the IMS Java classes or refer to the DBD or PSB source.

An XML file, written in UTF-8 encoding, is produced by the utility if you specify `genXML=YES` in the `OPTIONS` control statement. It describes all of the PCBs and their referenced DBDs processed in the entire run of the utility. The XML that is produced by the utility is based on a metamodel of IMS database defined in UML. This model is a package with a number of inheritance relationships to the OMG Common Warehouse Metamodel (CWM). However, only the IMS package itself is included and used in the DLIModel utility. The generated XML schema, written in UTF-8 encoding, is an XML document describing an IMS database based on a PCB. An XML schema is required to retrieve or store XML in IMS. IMS uses an XML schema to validate an XML document that is being stored into or retrieved from IMS. The XML schema, not the application program, determines structural layout of the parsed XML document in the database during storage and the generated XML document during retrieval.

IMS | IMS

## Database Visualization (UML View of the Database Metadata)

The screenshot shows the DLIModel Utility GUI in Eclipse Platform. The main window displays a UML class diagram representing database metadata. The diagram includes the following tables and their attributes:

- ORDER** (Total length: 74)
  - ORDERID
  - ORDERDATE
  - ORDERTIME
  - ORDERSTATUS
  - ORDERDATE
- MODEL** (Total length: 33)
  - MODELID
  - CITY
  - ZIP
  - PHONE
  - ADDRESS
  - STATE
- SALES** (Total length: 43)
  - SALID
  - SALDATE
  - SALTIME
  - SALPRICE
  - SALLOT
- STOCK** (Total length: 44)
  - STOCKID
  - STOCKDATE
  - STOCKTIME
  - STOCKPRICE
  - STOCKLOT
- SALESP** (Total length: 44)
  - SALESPID
  - SALESPDATE
  - SALESPTIME
  - SALESPPRICE
  - SALESPLOT

The diagram shows relationships between these tables, with lines connecting them to indicate dependencies or associations. The GUI also includes a Package Explorer on the left and a Properties window at the bottom.

28

© 2009 IBM Corporation

This shows an example of the GUI view that the DLIModel Utility provides of the IMS Database Metadata.

IMS

## IMS Enterprise Suite 1.1 DLIModel Utility Plug-in

- **Graphical User Interface (GUI)**
  - Leverage Eclipse, Eclipse Modeling Framework (EMF) and Graphical Editor Framework (GEF)
  - Can be installed as a stand-alone function or on top of other Eclipse based products (i.e. RAD 7.5, RDz 7.5, Data Studio) **using IBM Installation Manager**
- **IMS Database Visualization Tool**
  - User can visualize an entire IMS PSB and DBD in a multi-page graphical editor.
    - Each PCB can be viewed, saved and **printed** individually. Each PCB editor shows the IMS DB hierarchy with the segments, fields, field types, etc.
  - User can also **search** an entire IMS PSB for a specific PCB, segment, or field.
- **IMS Database Metadata Generation Tool**
  - It has been used to generate the necessary metadata that is consumed at runtime by the IMS Universal driver, XML-DB, XQuery and IMS DB Web services.
    - **DLIDatabaseView for IMS Universal driver**
    - XML schema for XML DB and XQuery
    - Deployable artifacts (EAR and WSDL files) for IMS DB Web services via the DAS commands in a syntax assist and syntax highlight editor.
  - This tooling currently uses a bottom-up approach, parsing PSB and DBD source using either C control statements or Wizard panels. User can optionally import COBOL copybook and **PL/I Include** definitions to define field layouts for each segment.

29

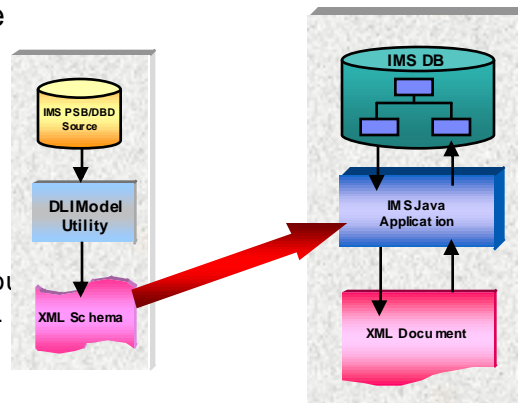
© 2009 IBM Corporation

DL/I Model Utility plug in provides a user friendly interface, simplifies IMS metadata generation, eases IMS Java and XML database application development and access, and offers a visual representation of IMS databases. Enhancements ease use of this utility, and its users can now import PL/I Include to redefine segment layout in IMS metadata, as well as take advantage of the new IMS Universal JDBC driver. Enhancements also include:

- Export PSB graphical view as graphic files (JPG or BMP)
- Auto select DBDs that referred by a PSB in wizard
- Support PL/I Include Import
- Add PROCOPT to IMS metadata for the IMS JDBC driver
- Add Virtual Foreign Key view to the PSB graphical editor
- Change GUI messages to match with product messages prefixes
- Add the search capability to the PSB graphical view
- Update the existing metadata with newly updated PSB/DBD source
- Ship under the new IMS Enterprise Suite through IBM Installation Manager

## IMS Java and IMS XML Databases

- Two Types of IMS XML Database
  - Decomposed or Virtual XML DB
    - A standard IMS DB, which has data automatically transformed into XML when retrieved (and v.v.)
  - Intact XML DB
    - Where XML data is stored without transformation (i.e. with its XML tags) on the IMS DB
- Java API - retrieveXML() and storeXML()
- DLIModel utility - used to generate XML schema from DBD



Since IMS 9, it has been possible to use an IMS database as an XML database. There are two possible implementations.

Firstly, the IMS database can be a **Virtual** or **Decomposed** XML database. The data in the database is absolutely standard IMS data. However, by using an XML schema that describes how the IMS data would be represented in an XML document, it is possible to transform the data between the XML document and the IMS database in either direction. This is an extremely powerful feature of IMS, and is simplified because both XML and IMS use hierarchical data structures, and hence it is easy to map between the two formats. As mentioned earlier, the XML Schema can be created by the DLIModel utility.

The second possibility is to use an IMS database to hold **intact** XML documents – both the data and the XML tags are stored on the database.

In both cases, the XML document is stored or retrieved using JDBC.

## IMS Java and IMS XML Databases ...

- XML Data is retrieved and stored using an extension of JDBC
- Examples:

```
SELECT retrieveXML(Employee)
FROM employeeDB.Employee
WHERE Employee.serialNumber = '3A0140'
```

- ▶ Build an XML document out of the Employee Segment and all its Dependent Segments in this PCB, for the employee with serial number 3A0140

**Note: SELECT of XML still populates a result set. The XML document(s) will form one column. You could select other things into other columns.**

```
SELECT Employee.serialNumber, retrieveXML(Employee) FROM ...
```

```
INSERT INTO custXMLDB.Customer
(CustNo, storeXML()) VALUES ( ?, ? )
```

- ▶ custXMLDB is an intact XML DB. The root segment is called "Customer", and contains a key field, "CustNo" and the first piece of the XML document

To retrieve IMS data as an XML document, you would use the SQL SELECT function, as usual. This will still return a result set – some columns can be used for DB fields, and other columns can be used for XML documents. This requires a "JDBC extension" to specify that an XML document is to be SELECTed rather than a field. In the first example on the slide, the SELECT specifies a single value to be SELECTed – "retrieveXML(Employee)". The WHERE clause limits this to the employee with serialNumber '3A0140'. So the result set will contain one column and probably one row. The content will be the XML document created from the EMPLOYEE segment and all its dependent segments in the employeeDB PCB view.

The example in the middle of the slide shows a variation on this. In this case the result set will contain two columns – the first will be the employee serial number and the second the XML employee document.

The third example shows an INSERT of an XML document onto a customer database. This is actually an example using an intact XML database, and so IMS requires a separate root key value to be specified as well as the XML document itself. Before executing the insert, the application will have to set the first "?" equal to the customer key value, and the second "?" will be set equal to the customer XML document. Then the SQL INSERT can be executed.

IMS

## IMS DB Resource Adapter XQuery API

- XQuery is a language for querying XML data
  - Result of query is itself an XML document
  - Note XQuery is currently a read only language
- IMS 10 XQuery support
  - Supports Decomposed (Virtual) XML DB only
  - Views result of SQL SELECT as the input document to be XQueried
  - Allows any standard IMS DB to be queried with XQUERY
- Supports the XQuery 1.0 and XPath 2.0 Data Model
  - XQuery is based on the structure of XML to provide query capabilities
- Extends IMS DB Resource Adapter API
  - SELECT retrieveXML( <segment>, <xquery y> )
  - FROM <pcb.segment>
  - WHERE <predicate>

- Initially, the IMS XQUERY is implemented via an SQL QUERY
  - You should use the WHERE clause to reduce the amount of IMS data that is searched
- At a later time, the XQUERY internal processing will be enhanced to limit the search to the necessary DB records

32 © 2009 IBM Corporation

XQuery is a functional programming language that was designed by the World Wide Web Consortium (W3C) to meet specific requirements for querying XML data.

XQuery is based on the structure of XML and leverages this structure to provide query capabilities for the same range of data that XML stores.

The [IMS DB Resource Adapter](#) XQuery support extends the *retrieveXML* User Defined Function (UDF) by adding a second parameter. The second parameter allows the passing of an XQuery 1.0 expression.

The expression is evaluated relative to the *retrieveXML* context and returned to the result set as a CLOB value. This implementation views the entire IMS DB as an XML document and enables the return of specific IMS data based on the XQuery. For IMS XQuery support the XQuery 1.0 and XPath 2.0 Data Model serves two purposes. First, it defines the information contained in the input to be used by the IMS XQuery processor. Second, it defines all permissible values of expressions in the XQuery, and XPath languages that can be evaluated by the IMS XQuery processor. The [IMS DB Resource Adapter](#) is packaged in *imsjava.jar*. The IMS XQuery function resides in a separate package (*imsxquery.jar*).

### Benefits

IMS participates in industry standards

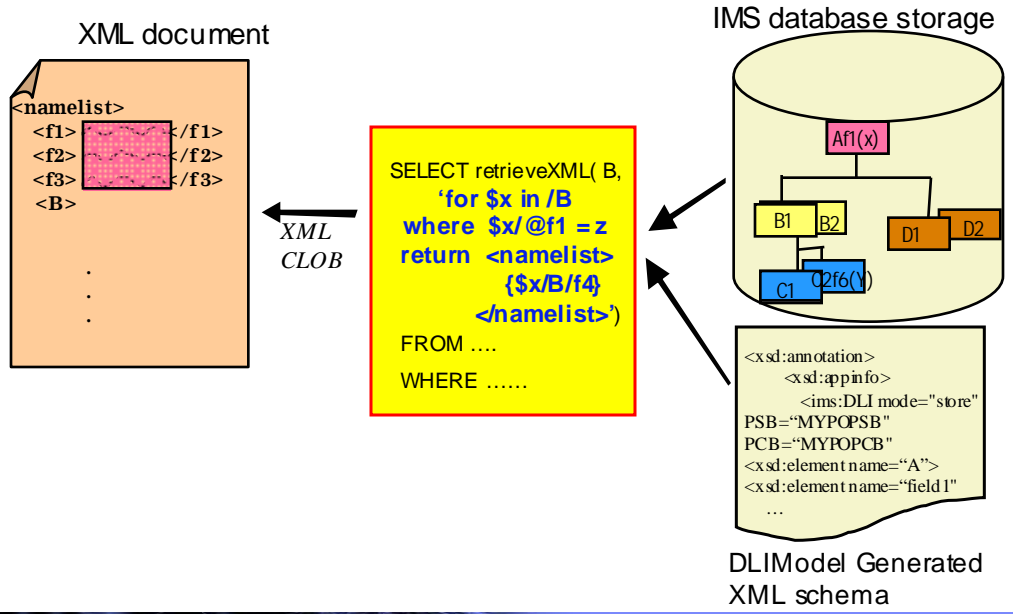
Can use Application Development tools that support XQuery

Can be used with existing IMS data and IMS XML Schemas

As GUIs are developed for front-ending XQUERY, the objective is that the target DB can be an IMS database



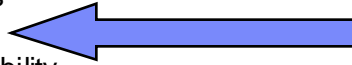
## IMS DB Resource Adapter XQuery API ...



Since IMS XQuery is an extension to IMS XML DB, existing DLIModel generated XML Schemas can be used by the IMS XQuery processor to compose XML documents.

## Agenda

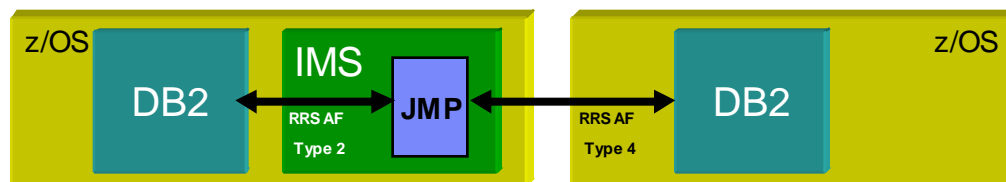
- Java basics
- IMS Java basics
- IMS Java Applications
- IMS Database Access
- DB2 Access
- Application Interoperability
- Technical Considerations
- Customer Examples
- Summary



DB2 can also be accessed with IMS Java programs

## Accessing DB2 from IMS Java Program

- IMS JMP and JBP regions use a different DB2 attachment facility from other types of dependent region
  - Recoverable Resource Manager Services attachment facility (RRSAF) rather than the External Subsystem attachment facility (ESAF)
- RRSAF supports attachment from IMS on one z/OS to DB2 on a different z/OS
  - The JDBC driver type should be specified when creating the connection
    - Type 2 for DB2 on same z/OS
    - Type 4 for DB2 (potentially) on a different z/OS



The IMS Java regions use a different DB2 attach facility from other IMS regions. If using a distributed attachment, RRS provides the synchpoint management.

## Accessing DB2 from IMS Java Program ...

- All DB2 calls are part of one UOW
- RRS set up in z/OS and activated in IMS (RRS=Y)
- DB2 setup for using RRSAF from IMS required
  - SSM member of IMSxxx.PROCLIB or DB2 subsystem example
    - SST=DB2,SSN=DB2E,COORD=RRS
  - Add DB2 to trustedmiddleware
  - Add DB2 to LIBPATH
    - LIBPATH=/usr/lpp/db2/db2910\_jdbc/lib
  - Add the DB2 library to JMP region with the DFSDB2AF DD (which must all be APF authorized libraries)
- Plan with the name of the PSB/Program must be bound or RTT
- Packages for all COBOL modules and the four DSNJDBCx packages must be bound to the corresponding plan

Considerations for accessing DB2 from IMS Java programs are shown here.

## Agenda

- Java basics
- IMS Java basics
- IMS Java Applications
- IMS Database Access
- DB2 Access
- Application Interoperability
- Technical Considerations
- Customer Examples
- Summary



We discussed IMS Java applications and database access. Next we'll talk about interoperability between applications.

## Integrating with Existing IMS Transactions

- The goal should not be to rewrite all applications in Java
  - Not even possible in many instances
  - Java is not meant to replace existing applications
    - Meant to complement them
- Instead integrate the two environments
  - Program switching (immediate and deferred)
    - Between transactions in different languages
  - COBOL/PL/I and Java language interoperability
    - To exploit COBOL/PL/I subroutines in a Java program
    - To exploit Java classes in COBOL/PL/I program

So now you might be thinking “Ok, great! I’m reducing the amount of code I write by leveraging Java code. Wait a minute... am I making COBOL less relevant”. No this works both ways you can also extend COBOL applications so that Java developers can now access them. This means Java developers can take up a lot of the maintenance and enhancement work on existing COBOL code. Yes, this may seem like it’s still not helping the COBOL developer since it is shifting work to Java but it does. We’re moving off the menial less interesting work and freeing up COBOL developers to create new COBOL applications.

## zOS Language Interoperability

- IBM's Enterprise COBOL for z/OS V3R4 (or higher) and IBM Enterprise PL/I for z/OS V3.8 (or higher) support interoperability between COBOL/PLI and Java
  - When running in an IMS Java dependent region (JMP or JBP)
  - When running in MPP & BMPs
- Supported Scenarios
  - Java Calls OO COBOL module (COBOL Compiler generates Java Wrapper and Shared Library containing COBOL code)
  - OO COBOL program can call Java Classes
  - Procedural modules can be called from OO COBOL program
  - PL/I modules can be called from Java through JNI (Java Wrapper must be created)


IBMs Enterprise COBOL and Enterprise PL/I support interoperability between COBOL/PL/I and Java applications

IMS

## Interoperability in Java Regions

- IMS Java can be used to call COBOL and PL/I modules through JNI
- In JMP/JBP IMS launches JVM during region initialization.

TCP/IP or SNA



*Server-Side Presentation Management*

*Server-Side Business Logic*

**IMS Application Server**

**IMS TM**

Control Region

IMS Connect

**IMS JMP Region**

Java Application

```
Class.forName(DLIDriver)
get.connection(IMS psb)
Select
From
Where
Close
```

**IMS JBP Region**

Java Application

```
Class.forName(DLIDriver)
get.connection(IMS psb)
Select
From
Where
Close
```

IMS z/OS Platform

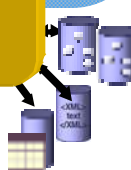
JDBC Drivers

JDBC Drivers

IMS DB and/or DB2

JNI

OO COBOL Class or PL/I Module ...



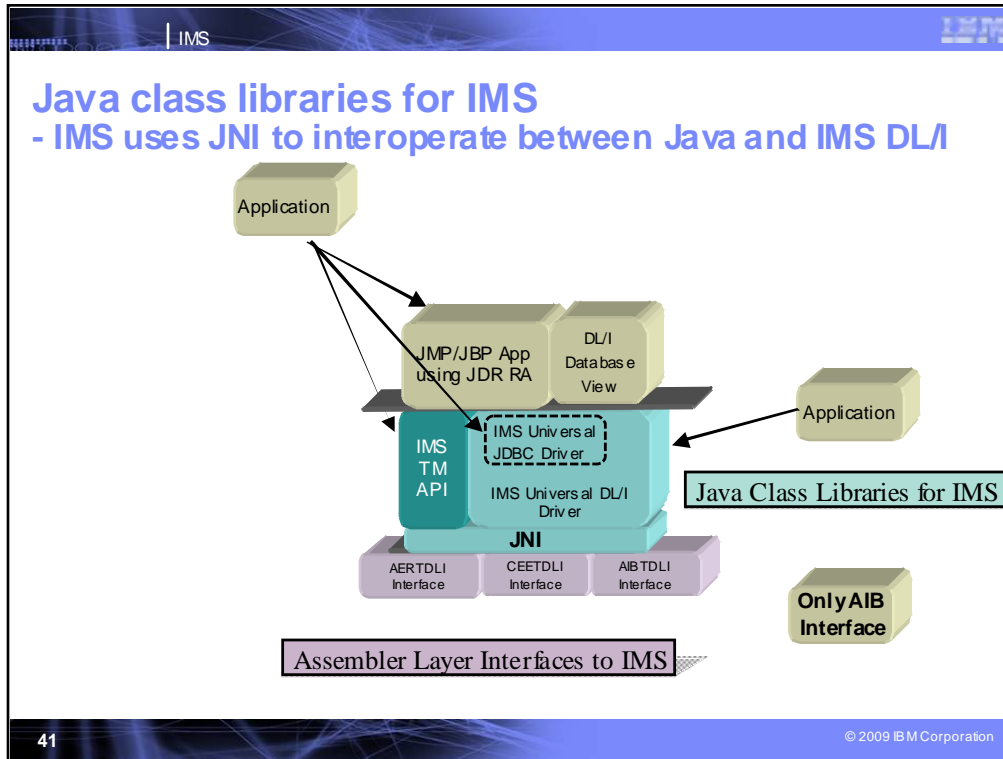
- JMP same as MPP+JVM
- JBP same as non-message driven BMP+JVM

40

© 2009 IBM Corporation

IMS Java can be used to call COBOL and PL/I through the Java Native Interface.





Java Native Interface is shown here for use in accessing IMS procedural code

Bottom – language interface (depending on the environment 3 interfaces) – have to drop down to c (thin jni layer)

Base – 1-1 mapping of the way ims works under covers and in java build SSAs & make db calls using DL/I

DB – really what is turning this in to our jdbc driver making sql calls

Application is running in an IMS dependent region and offers reading/writing messages to ims message queue

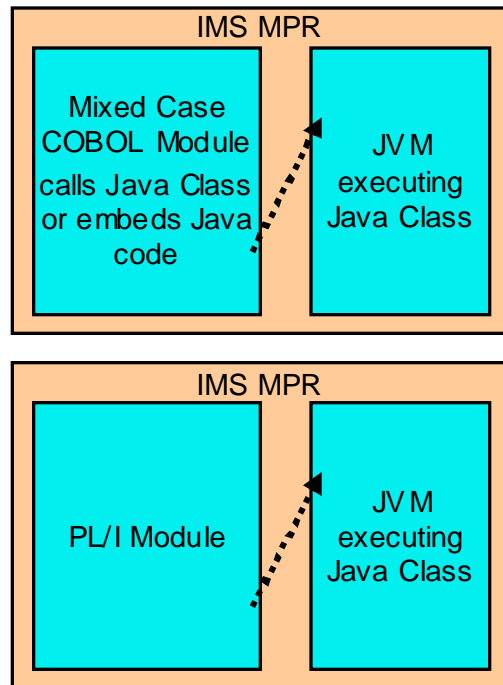
Customer code doesn't have to worry about the DL/I call, only JDBC

Dbview- added new stuff (XML)

Tooling – generates database view called dli model utility

## Interoperability in MPRs

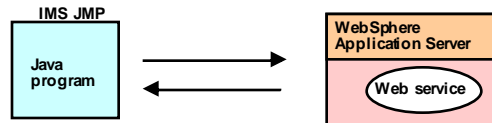
- COBOL
  - JVM created implicit
  - Java code can be embedded
  - Parameters to and from Java prepared through JNI API
- PL/I
  - JVM created through JNI API
  - Parameters to and from Java prepared through JNI API
  - Java class called with JNI API



COBOL and PL/I applications in Message Processing Regions interoperate with JVM executing Java classes.

## Consuming Web Services...

- IMS Java application callout capabilities
  - Natively, this capability exists in Java.
    - Direct call (synchronous and asynchronous)
    - JAX-WS
    - JAX-RPC



- References
  - SC18-7821 IMS Java Guide and Reference

IMS Java applications can themselves call out to other applications.

IMS

## IMS Connect Solutions provide Interoperation between IMS Applications and other Application Environments

- IMS Connect provides the IMS interface for TCP/IP solutions
  - IMS SOA Integration Suite at <http://www.ibm.com/ims>
    - IMS TM Resource Adapter
      - IMS MFS Web Support
    - IMS SOAP Gateway
    - IMS Web 2.0 Solution
    - IMS Enterprise Suite
  - Write your own clients
  - Other Vendor solutions

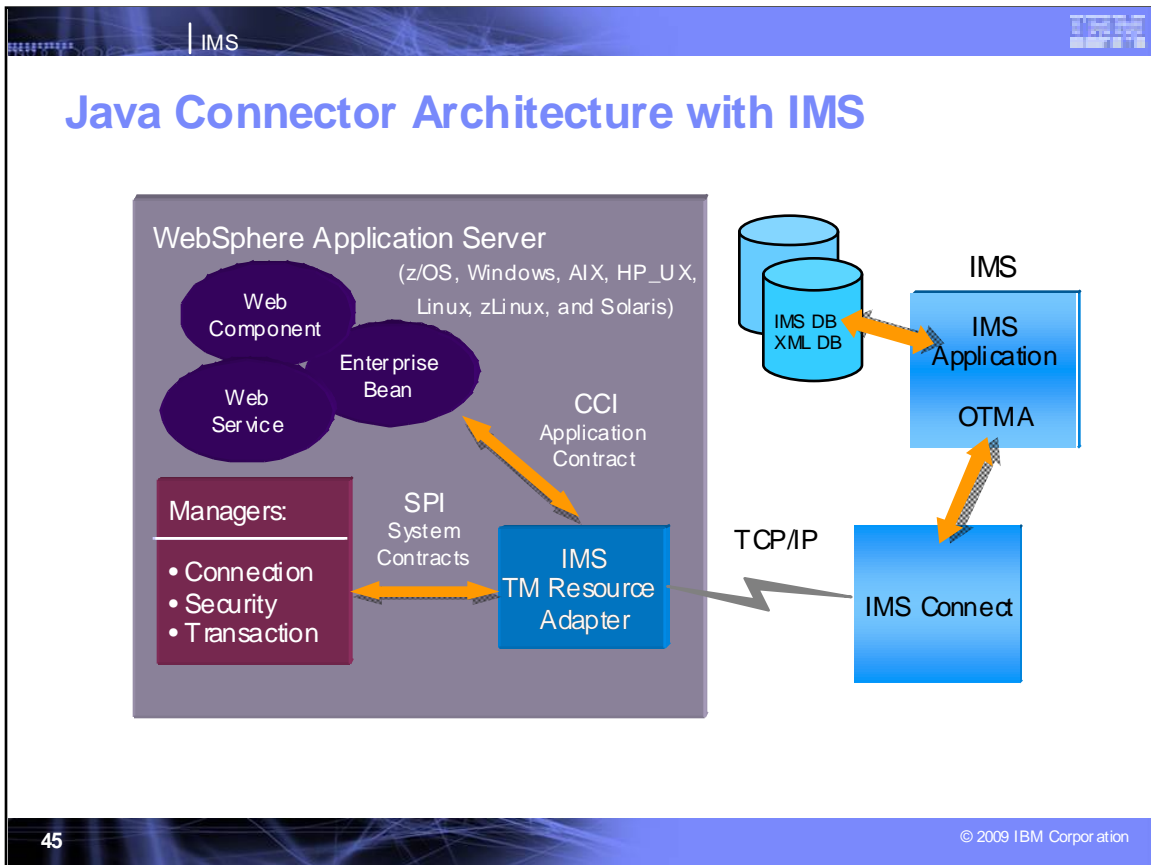
Existing IMS transactions can be integrated into the SOA by implementing a Web service as a front-end access-point interface

44

© 2009 IBM Corporation

IMS Connect provides TCP/IP access to and from IMS.

For access to IMS through IMS Connect from a JEE environment, IMS provides the IMS TM Resource Adapter (earlier known as the IMS Connector for Java) for access from Java applications, SOAP Gateway and parsers, and samples for other language access as well. Support for MFS is also available. Other solutions providing access to IMS through IMS Connect include the IMS SOAP Gateway for access from non-J2EE environments, IMS Web 2.0 support for Mashups, and many more solutions.



IMS TM Resource Adapter is supported with WebSphere servers such as WebSphere Process Server and WebSphere Application Server for z

IMS TM Resource Adapter is both scalable and flexible. You can configure your environment to use IMS TM Resource Adapter in a distributed application server or, for higher scalability, you can use IMS TM Resource Adapter in WAS for z/OS. Being a Java workload, if you run WAS for zSeries, the workload is eligible for off-loading to a more cost-effective zAAP processor.

IMS

## IMS 10 as Integration Focal Point - Callout Support

- IMS 10 Asynchronous Callout
  - Enable IMS applications to act as a client to asynchronously invoke Java EE applications and Web Services
  - Receiving output from external application is possible

- IMS 10 SPE Synchronous Callout Support
  - Enable IMS applications to invoke Java EE applications and Web Services, and synchronously wait for the response

46

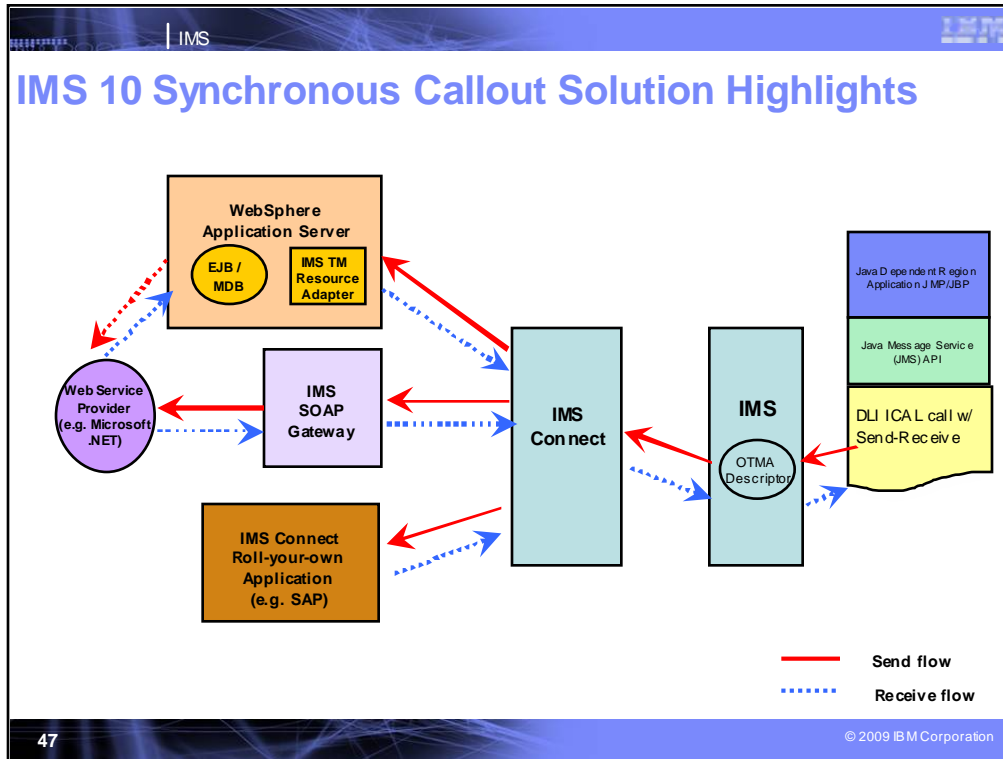
© 2009 IBM Corporation

### Key Message: IMS 10 Callout support enables IMS as the Integration Focal Point

One of the key customer requirements that we have heard regarding SOA support for IMS is for Callout support, where an IMS application could call out to another application across the IMS TM Resource Adapter to WebSphere server applications or to SOAP application environments. IMS Callout support enables IMS applications as clients, interoperating with business logic outside of the IMS environment. This support includes correlation mapping between the callout request and the external application, enhanced security, and assistance on destination routing. This support allows for better integration in an SOA environment.

For example, an application may need to know the current stock price, or may need to look up the current sales tax rate.

Callout support has been provided for IMS 10, first as an asynchronous transaction, and later as a synchronous call where the IMS application will wait for the response before continuing. Synchronous callout support is being provided through the service process



Callout can be provided from IMS Java and other IMS applications. Synchronous callout specially addresses the need for an IMS application to act as a client to go outbound synchronously to invoke external application and receive the output back. This enhancement allows your IMS application to invoke one of the following external applications and synchronously get the output back:

- (1) an J2EE application (like an Enterprise Java Bean/EJB or an Message Driven Bean (MDB)) or Web service providers running in the WebSphere Application Server using the IMS TM Resource Adapter
- (2) other Web service providers (e.g. like Microsoft .NET or SAP XI) using IMS SOAP Gateway
- (3) any other applications (like RYO, SAP apps) using the IMS Connect interfaces

This diagram gives you a very high-level overview of the synchronous callout SPE.

The red arrow represents the send flow. From the right hand side of the diagram, an IMS application makes a DLI call to send out a synchronous callout request. The OTMA descriptor function inside IMS routes the callout request via IMS Connect to invoke one of the outbound destination as shown in the boxes on the left – i.e. the WebSphere, Web Services or RYO applications. After the callout request has been processed, the output data would be returned back to the same IMS transaction instance as shown in the blue arrows.

IMS

## IMS Enterprise Suite 1.1 JMS API Support for Synchronous Callout

- Provides Java Message Server (JMS) API for accessing IMS Synchronous Callout function.
  - Enables business growth -- Allows more flexibility in accessing cross enterprise data and functionality from within IMS applications to meet growth challenges.
  - Exposes core IMS functionality through a Java standard interface - Makes IMS function more accessible to application developers with modern skill sets.
  - Offers standards-based approach - Exposes IMS industry leading transaction management capability through a Java standard interface, JMS
  - IMS Callout function included in IMS; JMS API packaged with IMS Enterprise Suite.
  - Enables new application design frameworks and patterns
  - Synchronous callout support is the first IMS function to fully embrace the JMS standard in IMS application development.

48

© 2009 IBM Corporation

The Java Message Server (JMS) API can be used for synchronous callout from an IMS Java application. The IMS Enterprise Suite is providing the JMS API Jar file for this.

The JMS API improves programmer productivity by defining a common set of messaging concepts and programming strategies that will be supported by all JMS technology-compliant messaging systems. By making IMS a JMS provider we address the skills issue impacting client's ability to develop new applications which goes to revenue. Although heavily used by IMS customers, the DL/I API isn't an industry standard and skills may be limited. Providing modern standards based access to IMS functions reduces customer costs.

Synchronous callout support is the first IMS function to fully embrace the JMS standard in IMS application development. We are considering future enhancements to front-end IMS message queue processing with the JMS interface.



IMS Enterprise Suite 1.1 Connect API for Java

- Simplifies interaction with IMS Connect and IMS
  - Internally creates IRM header for request and interprets non-data response information
  - Automatically opens socket connections to target IMS Connect as needed
  - Re-usable profiles specify target IMS Connect and IMS and interaction
  - Allows client applications to provide data in a variety of ways
  - Lower-level calls provided
  - High-level execute call to open connection and performs interaction on behalf of client
  - Applications must call disconnect() on all connections before termination
  - Supports IMS Connect PING and RACF password change commands
  - Supports all OTMA-supported IMS commands

49

© 2009 IBM Corporation

### IBM is also enhancing IMS Connect use with the IMS Enterprise Suite Connect API

The IMS Enterprise Suite Connect API simplifies design, development, and test of IMS access for client TCP/IP applications. Support for Java applications in Windows and z/OS environments is provided initially with support for C applications in Windows environments being delivered next through the service process. The API provides a customizable set of profiles that define the connections and types of interactions to be performed and high- and low-level methods for performing these interactions. This simple API allows user-developed applications to interact with IMS Connect while shielding the applications from the complexities of the IMS Connect protocols by automatically generating the IMS Request Message header, interpreting the non-data portions of response messages and opening socket connections on behalf of the applications. Using the reusable profiles, the API provides a simple way to describe the TCP/IP socket connections needed and the interaction to be performed. In addition, it gives the client application flexible ways to provide the data to be sent including 1- and 2-dimensional byte arrays, Strings and arrays of Strings. For more direct control of an interaction by a client application, limited, more granular, lower-level calls are provided which allow the client application to explicitly open a socket connection and send and receive data to IMS Connect. The disadvantage of using the lower-level send and receive calls is that you lose the benefit of the API creating the message for you and interpreting the non-data portions of the response.

The API will communicate with IMS Connect on behalf of the client. Upon request by the client application, for example in an execute() or connect() call, the API will create a connection for use by that application. The client application does not need to deal directly with the connection, other than to keep track of it through the Connection object so that the client can call disconnect() on the Connection object before exiting. Of course in Java, orphaned connections would eventually be cleaned whenever there are no longer any references to those connection objects or when the application terminates. However, failure to explicitly disconnect a socket before an application shuts down may lead to warning messages being displayed on the IMS Connect console when TCP/IP closes a connection without the application or the API sending a disconnect request to IMS Connect.

In addition to IMS transactions, the initial release of the IMS Connect API will support the IMS Connect user message exit-supported PING and RACF password change commands along with all IMS commands supported by OTMA.

This figure shows the environment in which the IMS Connect API can be used. It also depicts the fact that multiple client applications that use the IMS Connect API can be invoked simultaneously. The API will communicate with IMS Connect on behalf of the client applications.

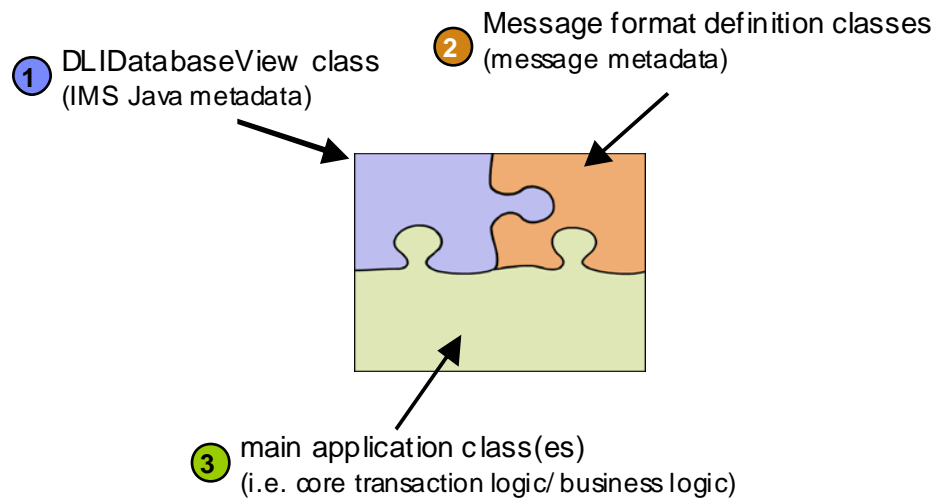
## Agenda

- Java basics
- IMS Java basics
- IMS Java Applications
- IMS Database Access
- DB2 Access
- Application Interoperability
- Technical Considerations
- Customer Examples
- Summary



There are some technical considerations you might want to be aware of in doing all this.

## IMS Java Basic Transaction Program Skeleton



**Note:** JBP applications cannot read messages from the message queue, but they can write output messages to the queue.

## Developing Java Transaction Programs

- Using Rational Developer for System z (RDz) enables the Java code to be created and maintained on the host system
  - Saves porting between development and test environments
- BTS fully supports development of message driven Java programs (JMPs) and Java Batch Programs (JBPs)
- Debugging aided by IMS Java's JVM Logging facility

A number of tooling can provide development assistance for IMS Java applications and access to/from these and other applications.

## Technical Considerations: Language Interoperability

- All Environments for COBOL/PLI Java interoperability
  - Do not mix Message Processing and Synchronization, use one language for both
  - If same PCBs are used in both languages, they might have changed
    - Especially important when integrating existing modules that expect a PCB with a certain value
  - AIB interface is used in Java (PCBs must have names)
  - If subroutines require PCB pointers they can be created using INQY FIND call and provided with CALL to subroutine

Here are some environmental considerations.

## Technical Considerations: IMS Java Dependent Regions

- For JMP or JBPs
  - Java is the starting language
  - PSB LANG=JAVA (required for JMP only)
  - STOP run in COBOL causes application to end
  - COBOL Local Storage instead of Working Storage
  - JBP is non-message driven region so doesn't read messages from message queues
  - Program must start with a main method. COBOL class with main factory also meets that requirement
  - PCB name or label needs to be specified.

Some considerations for running the IMS Java Regions

## Technical Considerations: IMS MPP Regions

- **Language Interoperability in MPPs**
  - Solution for calling Java class from procedural program of any language
  - Requirements
    - Enterprise COBOL for z/OS V3R4 or higher
      - Enterprise COBOL for z/OS V4.2 provides support for Java 5 and 6, new XML, Unicode and performance and usability enhancements.
    - IBM Enterprise PL/I for z/OS V3.8 or higher
    - IBM Java 4.2 SR3 or higher
- **How it works**
  - JVM is created with first Java class invocation
  - JVM is destroyed at end of schedule
  - Schedule to first call is high due to JVM startup
  - Use WFI regions or dedicated transaction class
  - Create COBOL module that calls Java and link it with procedural caller (caller needs just call statement)
  - Use of CEEUOPT member for custom LE Parameters requires LE caller
  - DB2 Access possible, but different UOW and using Type-4 Connection only

The next few charts show how you might choose to run a JVM and interoperate with Cobol from within an IMS MPP regions. This allows a COBOL application to invoke JVM within an MPP region.

## Technical Considerations: IMS MPP Regions ...

- COBOL
  - IMS DL/I calls possible with no restrictions
  - DB2 calls possible with no restrictions
- Java
  - IMS DL/I calls only with environment variable
    - `com.ibm.ims.jdbcenvironment=IMS`
    - Includes message queue and system services calls due to check for execution environment
  - DB2 calls only possible with Type-4 Connection URL and without `com.ibm.ims.jdbcenvironment=IMS` being set
  - So any IMS DL/I calls and DB2 Database access can be used mutually exclusive only
    - That means with DB2 Type-4 being enabled, no IMS System services calls can be made and thus e.g. no Message sending is possible from Java

Here are considerations for running Java and COBOL in the same region



## Technical Considerations: Language Environment

- Java is compiled with DLL option and XPLINK(ON)
- OO COBOL or Mixed Case COBOL is DLL compiled
- Existing Modules might be NODLL and NODYNAM compiled
- NODLL Module cannot call DLL Module dynamically and vice versa
  - NODLL Module requires statically linked wrapper to be called from DLL
  - DLL Module requires statically linked wrapper to be called from NODLL
  - Free mixing of existing modules and Java classes requires two versions (DLL and NODLL) of each module
- PL/I Main requires PLIXOPT with XPLINK(ON)
- IMS Dynamic LE Options feature (UPDATE LE command) can be used to set XPLINK(ON)

Some considerations are shown here for the language environment for this JVM within the MPP region..

## Technical Considerations: Language Environment ...

- CEEUOPT module can be used to set XPLINK(ON)
  - POSIX(ON) is required
  - Path to the ENV file is required
    - Contains USS Environment variables (LIBPATH, PATH, CLASSPATH)
- LE Settings and ENV File
  - Locate the HFS path of the JVM  
e.g. /usr/lpp/java/J1.5/bin
  - Create the ENV file in the HFS path  
Edit the ENV file and according to your environment

```
//SYSIN DD *
      TITLE 'CEEUOPT'
CEEUOPT CSECT
CEEUOPT AMODE ANY
CEEUOPT RMODE ANY
CEEUOPT XPLINK=(ON), X
      POSIX=(ON), X
      ENVAR=('_CEE_ENVFILE=/u/gaebler/hello/ENV')
      END
//*
```

```
JAVA_HOME=/usr/lpp/java/J1.5
PATH=/bin:/usr/lpp/java/J1.5/bin:.
LIBPATH=/lib:/usr/lib:/usr/lpp/java/J1.5/bin/j9vm:/usr/lpp/java/J1.5/bin:/usr/
lpp/java/J1.5/bin/classic:/u/gaebler/hello
CLASSPATH=/u/gaebler/test.jar:/u/gaebler/hello
```

Some additional considerations are shown here for the language environment to launch the JVM in the MPP. The CEE User option module is used to set the communications link, XPLINK. And this shows the LE settings and Environment file.

## Agenda


- Java basics
- IMS Java basics
- IMS Java Applications
- IMS Database Access
- DB2 Access
- Application Interoperability
- Technical Considerations
- Customer Examples ←
- Summary



The following show some customers using IMS and Java .

IMS

## Northwest Airlines (now merged into Delta Airlines)



**Environment**

- × Have been running a successful airline very efficiently and effectively with IMS/DB2 on z/OS as a backend technology and plan to continue on that path
- × Why System z? Reduced operating costs; reduced operational and architectural complexity; ability to more easily consolidate workload due to merger
- × Why IMS? Industry proven; can handle high volumes of transactions better than other solutions; excellent support from IBM

**Business Objectives**

- × Integrate critical applications after merger
- × Implement a distributed application front end using SOA on top of our existing z/OS

**Solution**

- × Implement IMS/JDBC on z/OS to migrate non-critical applications to distributed technologies
- × Allows NWA to better manage the risk of depleting technical experts within COBOL/DB2/IMS technology

**Business Benefits**

- × NWA's technical infrastructure is much more open and able to integrate with all different types of technologies.
- × Smoother integration of all critical applications running on z/OS after merger with Delta

60 © 2009 IBM Corporation

**NWA's System z Strategic Direction:** Reduce operating costs; Reduce operational and architectural complexity; Workload consolidation due to merger

**The Strategic Infrastructure to Support their Goals:** IMS for day to day operations; DB2 day to day operations as well as for reporting; Other IM for System z products

**Why IMS?** Industry proven; Can handle high volumes of transactions and still perform better than many relational databases; Excellent support from IBM


**What we've accomplished:** We have been running a successful airline very efficiently and effectively with IMS/DB2 z/OS as a backend technology and planning to continue on that path. We have been working with IBM to implement a distributed application front end using SOA on top of our existing z/OS infrastructure. With the implementation of IMS/JDBC on z/OS, we have been able to manage the risk of depleting technical experts within COBOL/DB2/IMS technology.

**Benefits Realized:** A familiar and proven technology which minimizes the risks. With IMS/JDBC on z/OS, we are implementing a planned and well managed migration of non-critical applications to distributed technologies, but still running on z/OS. NWA's technical infrastructure is much more open and able to integrate with all different types of technologies.

**Looking Ahead:** With the help of IBM, our integration plans with Delta during merger are expected to be much smoother with all the critical applications running on z/OS; Keep working with IBM's On Demand division to implement better and faster solutions; Keep running a premier global airline successfully with existing technologies

IMS

## Canadian Bank



**Challenge**

- ✖ Core banking system written mostly in COBOL
- ✖ Lack of marketplace skills in order to maintain and develop new applications within their core banking system
- ✖ Need strategy to provide new service layers in their core banking system – cannot rip and replace

**Solution**

- ✖ Leverage the IMS JDBC driver as well as the core Java class libraries for IMS
- ✖ Interoperate between COBOL and Java within the core banking system
  - ✖ New services will be written in Java

**Benefit**

- ✖ Provide the ability to move the core banking framework into a new era where Java is the development language of choice
- ✖ Do NOT need to completely replace decades of infrastructure
- ✖ Over time inject new Java services into this framework
- ✖ Time to market for new apps dramatically decreased

61

© 2009 IBM Corporation

### Bank in Canada

Core banking system managed by IMS and written mostly in COBOL

#### Situation

Lack of marketplace skills in order to maintain and develop new applications within their core banking system

Need strategy to provide new service layers in their core banking system – cannot rip and replace

#### Solution

Leverage the JDBC driver for IMS as well as the core Java class libraries for IMS

Interoperate between COBOL and Java within the core banking system

New services will be written in Java

#### Value

Provide the ability to move the core banking framework into a new era where Java is the development language of choice

Do NOT need to completely replace decades of infrastructure – this would fail and is simply not an option

Over time inject new Java services into this framework

Time to market dramatically increased with the abundant skillset available in the marketplace

IMS looks like any other database (from an application development perspective)

Standards, standards, standards

## German Bank

### ▪ Challenge

- Mainly PL/I based with conversational transactions
- Purchased 3rd party credit checking technology as part of a Java package
  - Replaced existing PL/I-based transaction
- Wanted to deploy this in a Java Dependent Region and integrate with existing PL/I applications
  - Just another service...

### ▪ Solution

- Leverage the deferred program switching support already in IMS (and supported within the Java class libraries) to switch conversation iterations from MPP to JMP regions and back
- In production within a month with this solution

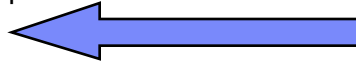
## And Many More European Customers...

- **Customer A**
  - Strategy to implement every business functionality as POJO (plain old Java object) which is runtime independent code and could be run either in Pure Java JVM, JCICS, Java Stored Procedure, IMS Java, WebSphere EJB, Java Servlet, + many other Java runtimes.
  - Migration of the IMS Batch to Java JBP
- **Customer B**
  - IMS Java for application for about two years
  - Eclipse for development
- **Customer C**
  - Java in COBOL regions to callout to EJBs – One year production without issues
- **Customer D**
  - Core banking application under IMS – COBOL and Java
- **Customer E**
  - Start with running Java in COBOL regions
- **Customer F**
  - Java runtime for some integration scenarios that require calling Pure Java in IMS

And there are many more customers implementing IMS and Java.

## Agenda

- Java basics
- IMS Java basics
- IMS Java Applications
- IMS Database Access
- DB2 Access
- Application Interoperability
- Technical Considerations
- Customer Examples
- Summary





## Getting More Information

Additional information can be accessed via the IMS home page at <http://www.ibm.com/ims>

- IMS SOA Integration Suite
  - IMS DB resource adapters
  - IMS XML DB
  - IMS TM resource adapter
  - IMS Enterprise Suite
    - SOAP Gateway
    - DLIModel utility
    - Connect API
  
- WW IMS Conferences and Seminars
  - IMS Seminars coming to a city near you
  - IMS Teleconferences, with replays available
  - Customized IMS offerings at [ibmdds@us.ibm.com](mailto:ibmdds@us.ibm.com)

IMS

## Simplifying IMS Application Development using Java

### Unique Offering

- Free application development workshop to learn how to easily and quickly develop and run Java applications to access IMS data using SQL and DLI APIs, which will allow you to more easily develop IMS applications, ensuring that your company's investments in IMS continue to pay dividends for years to come

### Workshop Description:

- Learn how to develop and run Java/JDBC applications against the IMS database easily. Test drive the newest application development API in Java for IMS™. In this workshop attendees will have the opportunity to develop, debug, and run sample Java applications using Eclipse based development IDE like Rational Developer for System z.

### Target Audience:

- Application Architects and Application Developers who are responsible for Java applications.

66 © 2009 IBM Corporation

We are currently offering some workshops, should you like to request one. This Java one covers:

### **What is special about IMS & Java?**

The IBM IMS™ hierarchical database has served as the backbone for industry-leading companies demanding the highest performance, stability and reliability for over 40 years now. Although the heart of this industry-proven data store has changed little, IMS continues to make great strides in new application development, connectivity, and data representation and its strategic role in an SOA environment.

This section will introduce you to IMS basics. Topic covered include: hierarchical database, IMS hierarchical terms such as: IMS records, IMS segments, Fields, key fields, search fields, etc. The IMS DLI application programming Interface (API) and how to issue DLI calls to access IMS databases and how to issue the same DLI calls in Java.

### **Tooling support for Application Development**

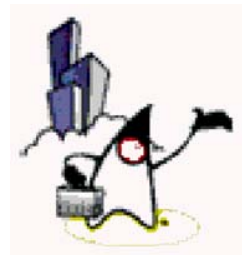
The IMS Enterprise Suite DLIModel Utility is an Eclipse plug-in tool that generates metadata that is consumed at runtime by the IMS Java/JDBC drivers. This session demonstrates how the utility can easily create the metadata from existing PSB, DBD and any COBOL or PL/I copybooks. The tool also generates visual depictions of the IMS database hierarchy which helps during application development.

### **Writing applications in Java for IMS**

Hands-on lab session that takes you through the steps for developing, deploying and running a Java application accessing IMS data. You will be building and executing applications to run in two runtime environments: Windows and in IMS dependent regions using the IMS Universal Java API drivers. The beauty of the Universal driver support is that there is virtually no code change required when

## Summary

- Java is another programming language for IMS alongside COBOL, PL/I, C etc.
- There is an increasing demand for Java to be used as a general purpose language, including use on the mainframe
  - single standard, most IT graduates know it
  - easy and economical to find Java programmers
- IMS Java introduces the power of IMS TM/DB to Java people, who can develop IMS transactions with workstation tools (e.g. Eclipse) in Java
- IMS transaction and JBP development is easy in java
- IMS java performance is continually being enhanced



Java is just another programming language, but is receiving increasing demand for enabling enterprise modernization. IMS Java brings the power of IMS TM and IMS DB to the Java community and the standards and tools provided for it.

## Disclaimer

© Copyright IBM Corporation 2008. All rights reserved.  
U.S. Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

**THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS AND/OR SOFTWARE.**

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml)

Other company, product, or service names may be trademarks or service marks of others.