E81-B

Variety of Techniques to Access IMS Using Java
Part Two
IMS Java Classes

Anaheim, California October 23 - 27, 2000

Ken Blackman
kblackm@us.ibm.com

RETURN TO INDEX

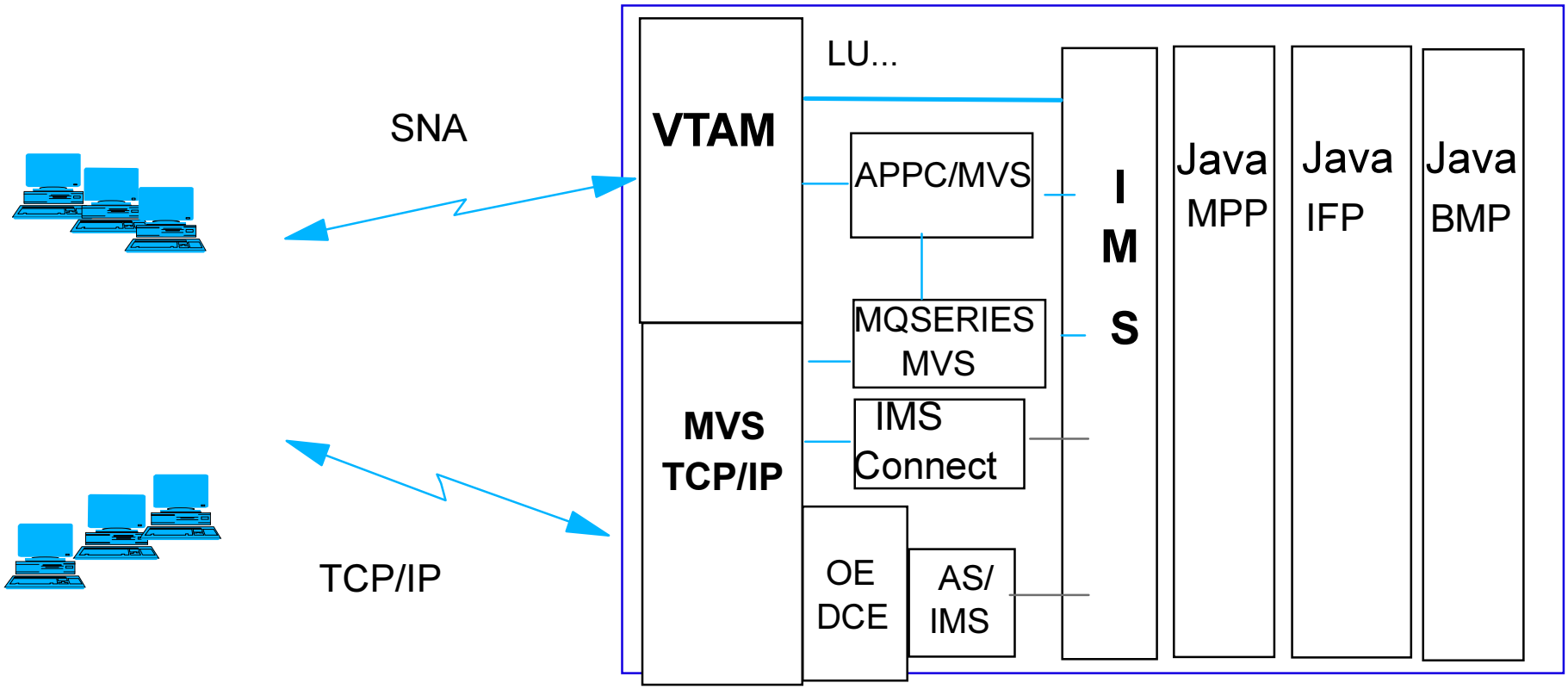
IMS Java

- **New function of IMS V7**
- **Capability to write, compile and run IMS Java programs**
  - Development environment
    - Provides a set of packages (groups of classes)
      - Allow access to IMS services
      - Support APIs familiar to Java programmers
    - Uses the S/390 HPJ (high performance java) compiler
      - Compiles bytecode into high-performance executables/DLLs
  - Runtime environment supports
    - Dependent regions (MPP, IFP, and BMP)
- **Benefit**
  - Incorporation of the Java programming model into the IMS environment

Software Prerequisites

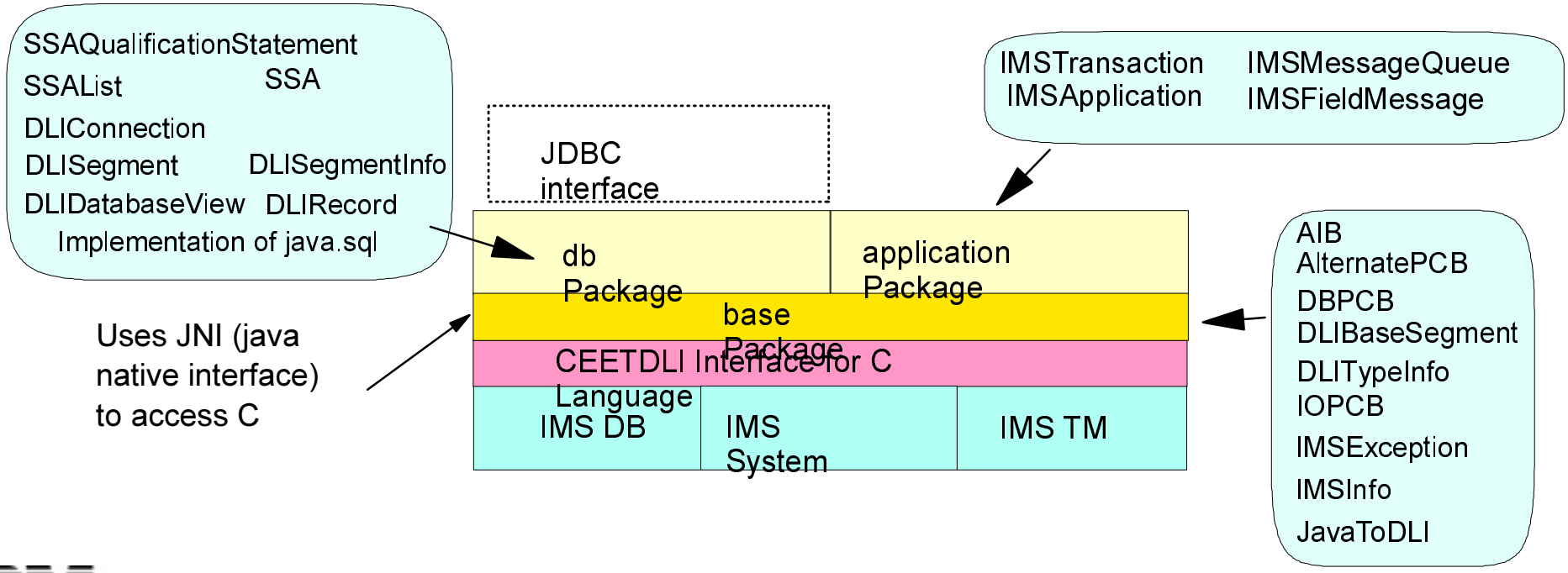
- **IMS V7 Transaction Manager**
- **OS/390 V2.6 or higher**
  - Java for OS/390 (5655-A46)
- **OS/390 LE Version 1 Release 7**
- **VisualAge Java Enterprise Edition for OS/390 (5655-VAJ)**
  - Provides High Performance Java (HPJ)
- **For DB2 access,**
  - Minimum of DB2 V5 with APAR PQ19814

Access to IMS Java Application Programs



IMS Java Class Library

Java program uses the APIs that are provided
- application Package classes to
  - initialize and begin the program
  - get the input message from the message queue
  - put the output message on the message queue
  - commit
- JDBC interface or db Package classes to
  - access the IMS databases



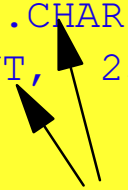
Building an IMS Java Application

- Define IMS Trancode as MODE=SNGL
- Define classes for input and output message segments
- Define classes to access IMS DB segments
- Use IMSMessageQueue to receive and send messages
- Use JDBC to access IMS DB
- Use JDBC to access DB2
- Use IMSTransaction to commit or rollback resources
  - Note:U118 abend if you do not commit/rollback

Subclass IMSFieldMessage to Define Messages an Input Message Example

|LL|ZZ|TRANCODE|RequestCode|DealerName|DealerID

```
public class InputMessage extends IMSFieldMessage {
final static DLITypeInfo[] messageInfo = {
    new DLITypeInfo("RequestCode", DLITypeInfo.INT,    1,  4),
    new DLITypeInfo("DealerName",    DLITypeInfo.CHAR,   5,  20),
    new DLITypeInfo("DealerID",   DLITypeInfo.INT,  25,  4)};

public InputMessage() {
    super(messageInfo, 28, false);
}
} // end InputMessage
```



NOTE:do not define LLZZTRANCODE fields.
use get methods provided by IMSFieldMessage

```
String trancode = null;
    trancode = InputMessage.getTransactionID();
```

Subclass IMSFieldMessage to Define SPA

|LL|ZZ|TRANCODE|RequestCode|DealerName|DealerID

```
public class SPAMessage extends IMSFieldMessage {
final static DLITypeInfo[] spaInfo = {
     new DLITypeInfo("RequestCode", DLITypeInfo.INT,     1,  4),
     new DLITypeInfo("DealerName",     DLITypeInfo.CHAR,    5,  20),
     new DLITypeInfo("DealerID",    DLITypeInfo.INT,  25,  4)};

public SPAMessage() {
     super(spaInfo, 28, true);
}
} // end SPAMessage
```

NOTE:do not define LLZZTRANCODE fields.
use get methods provided by IMSFieldMessage

```
String trancode = null;
     trancode = SPAMessage.getTransactionID();
```

Subclass IMSFieldMessage to Define Messages an Output Message Example

|LL|ZZ|RequestCode|OutputMessage

```
public class OutputMessage extends IMSFieldMessage {
final static DLITypeInfo[] messageInfo = {
    new DLITypeInfo("RequestCode",   DLITypeInfo.INT,   1,  4),
    new DLITypeInfo("Outputmsg", DLITypeInfo.CHAR,  5, 50)};

public OutputMessage() {
    super(messageInfo, 54, false);
}
} // end OutputMessage
```

NOTE:do not set LLZZ fields.
IMSFieldMessage will set the values

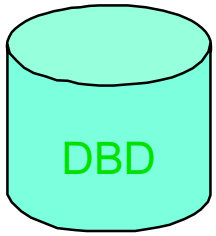
OutputMessage.setString("**Outputmsg**","Entry was Processed");

Subclass IMSApplication to implement main() and doBegin()

```
public class UserApplication extends IMSApplication {
public static void main(java.lang.String[] args) {
        UserApplication application = new UserApplication();
        application.begin();
}
public void doBegin() {

    IMSMessageQueue messageQueue = new IMSMessageQueue();

    InputMessage inputMessage = new InputMessage();
     while(messageQueue.getUniqueMessage(inputMessage)) {
          if (inputMessage.getMessageLength() > 0) {

    //  Add application/database logic here ...

    OutputMessage outputMessage = new OutputMessage();
    outputMessage.setDouble("RequestCode", inputMessage.getInt("RequestCode"));
    outputMessage.setString("Outputmsg", "Request successful");
    messageQueue.insertMessage(outputMessage);

    IMSMessageQueue alternatePCB = new IMSMessageQueue("PCBNAME");
      alternatePCB.insertMessage(outputMessage);

    // Commit changes to the database
    IMSTransaction.getTransaction().commit();
}
public UserApplication( ) {}
}
```

Define IMS Database Classes

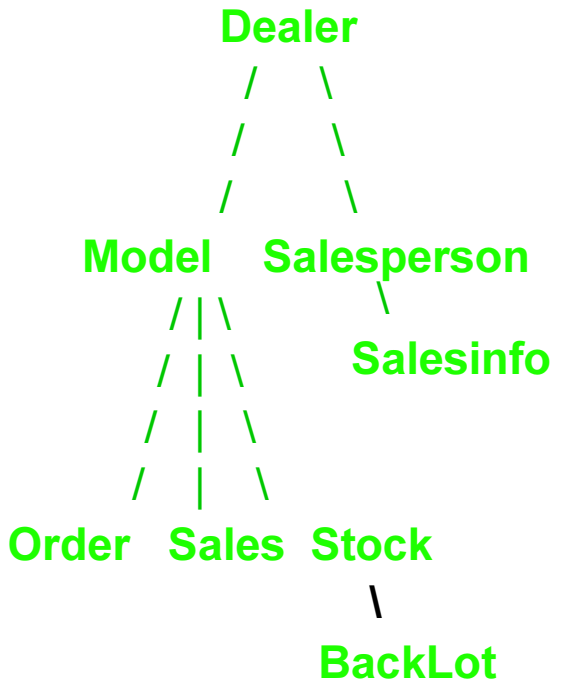
Determine Array Offsets
0 Dealer
1 Model
2 Order
3 Sales
4 Stock
5 BackLot
6 Salesperson
7 Salesinfo




```
DBD NAME=AUTODBD,ACCESS=DEDB
SEGM NAME=DEALER,PARENT=0
SEGM NAME=MODEL,PARENT=DEALER
SEGM NAME=ORDER,PARENT=MODEL
SEGM NAME=SALES,PARENT=MODEL
SEGM NAME=STOCK,PARENT=MODEL
SEGM NAME=BACKLOT,PARENT=STOCK
SEGM NAME=SALESPERSON,PARENT=DEALER
SEGM
NAME=SALESINFO,PARENT=SALESPERSON
```

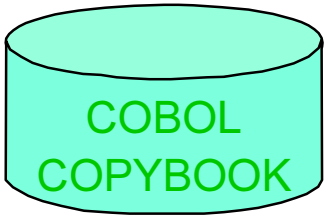
DLIDatabaseView subclass: Defines the hierarchy of DLISegments

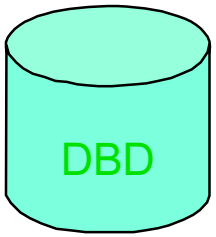
```
public class DealerDatabaseView extends DLIDatabaseView {
static DLISegmentInfo[] segments =  {
   new DLISegmentInfo(new Dealer(),DLIDatabaseView.ROOT),
   new DLISegmentInfo(new Model(),0),
   new DLISegmentInfo(new Order(),1),
   new DLISegmentInfo(new Sales(),1),
   new DLISegmentInfo(new Stock(),1),
   new DLISegmentInfo(new BackLot(),4)
   new DLISegmentInfo(new Salesperson(),0),
   new DLISegmentInfo(new Salesinfo(),6)
  };
public DealerDatabaseView() {
     super("AUTOPCB", segments);
}
} // end DealerDatabaseView
```



PCB TYPE=DB,DBDNAME=AUTODBD,PROCOPT=A,KEYLEN=4,PCBNAME=AUTOPCB

Define IMS Database Classes...






```
01 Dealer_Segment
    02 Dealer_ID  PIC X(4).
    02 Dealer_Name PIC X(20).
    02 Dealer_Address PIC X(30).
```

```
DBD NAME=AUTODBD,ACCESS=DEDB
SEGM NAME=DEALER,PARENT=0,BYTES=54,
FIELD NAME=(DLRNO,SEQ,U),BYTES=4,START=1,TYPE=C
FIELD NAME=DLRNAME,BYTES=20,START=5,TYPE=C
```

DLISegment subclass: Defines the type and layout of fields in a segment.

```
public class Dealer extends DLISegment {
static DLITypeInfo[] segmentInfo = {
 new DLITypeInfo("DealerID", DLITypeInfo.INT,   1, 4,  "DLRNO"),
 new DLITypeInfo("DealerName",   DLITypeInfo.CHAR,  5, 20, "DLRNAME"),
 new DLITypeInfo("DealerAddress",DLITypeInfo.CHAR, 25, 30)};

public Dealer() {
     super("DEALER", segmentInfo, 54);
}
} // end Dealer
```

What is JDBC API?

Common database programming interface for Java

- **Consists of two sets of interfaces**
  - JDBC API for application developers
    - Establish and open connection to database
    - Execute request to obtain results from database
    - Process the results
  - JDBC Driver API for database vendors
    - Function to support database connection
    - Function to support sending request to database
    - Function to support results processing

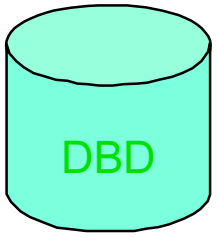
JDBC and SQL Statements

```
// Load the JDBC driver for DL/I data
Class.forName("jdbc:dli:DLIDriver");

// Establish a connection to the database
Connection con=DriverManager.getconnection("dealer.DealerDatabaseView");

// Create a statement object from the connection
Statement statement=con.createStatement();

// Execute a query against the statement
String query="SELECT DealerID, DealerName,"+
             "FROM  Dealer"+
             "WHERE DealerID > 300";
ResultSet results=statement.executeQuery(query);

// Examine the results of the query
while (results.next()) {
   String dealerID=results.getString("DealerID");
   String dealerName=results.getString("DealerName");
}
```

JDBC IMS Database Access

DLIDatabaseView subclass: Defines the hierarchy of DLISegments

```
public class DealerDatabaseView extends DLIDatabaseView {
static DLISegmentInfo[] segments =  {
   new DLISegmentInfo(new Dealer(),DLIDatabaseView.ROOT),
   new DLISegmentInfo(new Model(),0),
   new DLISegmentInfo(new Order(),1),
   new DLISegmentInfo(new Sales(),1),
   new DLISegmentInfo(new Stock(),1),
   new DLISegmentInfo(new BackLot(),4)
   new DLISegmentInfo(new Salesperson(),0),
   new DLISegmentInfo(new Salesinfo(),6)
  };
public DealerDatabaseView() {
     super("AUTOPCB", segments);
}
} // end DealerDatabaseView
```

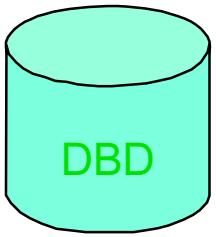


DBD NAME=AUTODBD,ACCESS=DEDB
SEGM NAME=DEALER,PARENT=0
SEGM NAME=MODEL,PARENT=DEALER
SEGM NAME=ORDER,PARENT=MODEL
SEGM NAME=SALES,PARENT=MODEL
SEGM NAME=STOCK,PARENT=MODEL
SEGM NAME=BACKLOT,PARENT=STOCK
SEGM NAME=SALESPERSON,PARENT=DEALER
SEGM NAME=SALESINFO,PARENT=SALESPERSON

```
Connection con=DriverManager.getconnection("dealer.DealerDatabaseView");
```

JDBC IMS Database Access...

```
public class Dealer extends DLISegment {
static DLITypeInfo[] segmentInfo = {
 new DLITypeInfo("DealerID", DLITypeInfo.INT,   1, 4,  "DLRNO"),
 new DLITypeInfo("DealerName",   DLITypeInfo.CHAR,  5, 20, "DLRNAME"),
 new DLITypeInfo("DealerAddress",DLITypeInfo.CHAR, 25, 30)};

public Dealer() {
        super("DEALER", segmentInfo, 54);
}
} // end Dealer
```




```
DBD NAME=AUTODBD,ACCESS=DEDB
SEGM NAME=DEALER,PARENT=0,BYTES=54,
FIELD NAME=(DLRNO,SEQ,U),BYTES=4,START=1,TYPE=C
FIELD NAME=DLRNAME,BYTES=20,START=5,TYPE=C
```

```
String query="SELECT DealerID, DealerName,"+
             "FROM  Dealer"+
             "WHERE DealerID > 300";
```

ssa=DEALERbb(DLRNObbb>b300)

DLIConnection IMS Database Access

Create a connection using the DLIDatabaseView subclass:

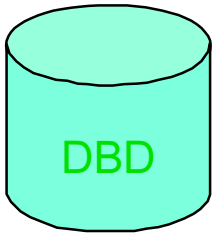
```
// Create a connection with the database
DealerDatabaseView dbView = new DealerDatabaseView();
DLIConnection connection =  new DLIConnection(dbView);

// Retrieve a segment from the database
Order order = new Order();
connection.getUniqueSegment(order, ssaList);

// Access the data in the segment object

// Return the OrderNunber and Price
String orderNumber = order.getString(1);
int    orderNumber = order.getInt("OrderNumber");
double price       = order.getdouble("Price");
```

Note: Data conversion functions provided for JDBC access are still available for DLIConnection access:

DLIConnection IMS Database Access

DLIDatabaseView subclass: Defines the hierarchy of DLISegments

```
public class DealerDatabaseView extends DLIDatabaseView {
static DLISegmentInfo[] segments =  {
   new DLISegmentInfo(new Dealer(),DLIDatabaseView.ROOT),
   new DLISegmentInfo(new Model(),0),
   new DLISegmentInfo(new Order(),1),
   new DLISegmentInfo(new Sales(),1),
   new DLISegmentInfo(new Stock(),1),
   new DLISegmentInfo(new BackLot(),4)
   new DLISegmentInfo(new Salesperson(),0),
   new DLISegmentInfo(new Salesinfo(),6)
  };
public DealerDatabaseView() {
     super("AUTOPCB", segments);
}
} // end DealerDatabaseView
```



DBD NAME=AUTODBD,ACCESS=DEDB
SEGM NAME=DEALER,PARENT=0
SEGM NAME=MODEL,PARENT=DEALER
SEGM NAME=ORDER,PARENT=MODEL
SEGM NAME=SALES,PARENT=MODEL
SEGM NAME=STOCK,PARENT=MODEL
SEGM NAME=BACKLOT,PARENT=STOCK
SEGM NAME=SALESPERSON,PARENT=DEALER
SEGM NAME=SALESINFO,PARENT=SALESPERSON

```
DealerDatabaseView dbView = new DealerDatabaseView();
DLIConnection connection =  new DLIConnection(dbView);
```

DLIConnection IMS Database Access

Note use the same DLISegment and DLIDatabaseView classes built for JDBC, but allows complete control over building SSAs and traversing the database.

Construct an SSAList object

```
SSAList ssaList = new SSAList();
```

Construct each SSA

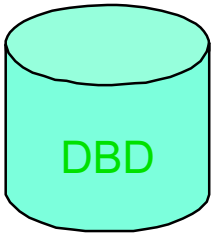
```
// Construct an unqualified SSA for the Root Segment
SSA dealerSSA = new SSA("Dealer"
                        "DealerID",SSA.EQUAL_TO,"300");

// Construct a qualified SSA
SSA orderSSA = new SSA("Order",
                       "Price", SSA.GREATER_THAN, "10000");

// Add an additional qualification statement
orderSSA.addQualification(SSA.OR,
                       "DeliverDate", SSA.LESS_THAN, "11212000");
```

Add SSAs to the SSAList object

```
ssaList.addSSA(dealerSSA);
 ssaList.addSSA(modelSSA);
ssaList.addSSA(orderSSA);
```

DLIConnection IMS Database Access

```
public class Dealer extends DLISegment {
static DLITypeInfo[] segmentInfo = {
 new DLITypeInfo("DealerID", DLITypeInfo.INT,   1, 4,  "DLRNO"),
 new DLITypeInfo("DealerName",   DLITypeInfo.CHAR,  5, 20, "DLRNAME"),
 new DLITypeInfo("DealerAddress",DLITypeInfo.CHAR, 25, 30)};

public Dealer() {
        super("DEALER", segmentInfo, 54);
}
} // end Dealer
```



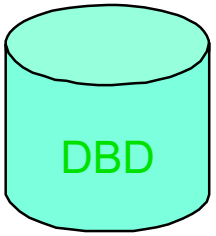

```
DBD NAME=AUTODBD,ACCESS=DEDB
SEGM NAME=DEALER,PARENT=0,BYTES=54,
FIELD NAME=(DLRNO,SEQ,U),BYTES=4,START=1,TYPE=C
FIELD NAME=DLRNAME,BYTES=20,START=5,TYPE=C
SEGM NAME=MODEL,PARENT=DEALER,BYTES=94,
FIELD NAME=(MODTYPE,SEQ,U),BYTES=2,START=1,TYPE=C
SEGM NAME=ORDER,PARENT=MODEL,BYTES=91,
FIELD NAME=(ORDNBR,SEQ,U),BYTES=6,START=1,TYPE=C
FIELD NAME=PRICE,BYTES=8,START=37,TYPE=C
```

```
SSA dealerSSA = new SSA("Dealer",
                    "DealerID",SSA.EQUAL_TO,"300");
SSA orderSSA = new SSA("Order",
                  "Price", SSA.GREATER_THAN, "10000");
```

```
public class Order extends com.ibm.ims.db.DLISegment {

   static DLITypeInfo[] typeInfo = {
       new DLITypeInfo("OrderNumber",   DLITypeInfo.CHAR,   1,  6, "ORDNO"),
       new DLITypeInfo("Price",         DLITypeInfo.DOUBLE, 37, 8, "PRICE"),

   };
   public Order() {
     super("ORDER", typeInfo, 91);
   }
} //end Order
```

DLIConnection IMS Database Access

ssa1=DEALERbb(DLRNObbb=b300)
ssa2=ORDERbbb(PRICEbbb>b10000)




```
Order order = new Order();
connection.getUniqueSegment(order, ssaList);

String orderNumber = order.getString(1);
int orderNumber = order.getInt("OrderNumber");

double price = dealer.getdouble("Price");
```

```
DBD NAME=AUTODBD,ACCESS=DEDB
SEGM NAME=DEALER,PARENT=0,BYTES=54,
FIELD NAME=(DLRNO,SEQ,U),BYTES=4,START=1,TYPE=C
FIELD NAME=DLRNAME,BYTES=30,START=5,TYPE=C
SEGM NAME=MODEL,PARENT=DEALER,BYTES=94,
FIELD NAME=(MODTYPE,SEQ,U),BYTES=2,START=1,TYPE=C
SEGM NAME=ORDER,PARENT=MODEL,BYTES=91,
FIELD NAME=(ORDNBR,SEQ,U),BYTES=6,START=1,TYPE=C
FIELD NAME=PRICE,BYTES=8,START=37,TYPE=C
```

```
public class Order extends com.ibm.ims.db.DLISegment {

  static DLITypeInfo[] typeInfo = {
     new DLITypeInfo("OrderNumber",  DLITypeInfo.CHAR,   1,  6, "ORDNO"),
     new DLITypeInfo("Price",        DLITypeInfo.DOUBLE, 37, 8. "PRICE"),

                          .

  };
  public Order() {
    super("ORDER", typeInfo, 91);
  }
} //end Order
```

Note when an SSA for one of the levels in a GU call that has multiple SSAs is not included, IMS assumes an SSA for that level.

Programing using JavaToDLI

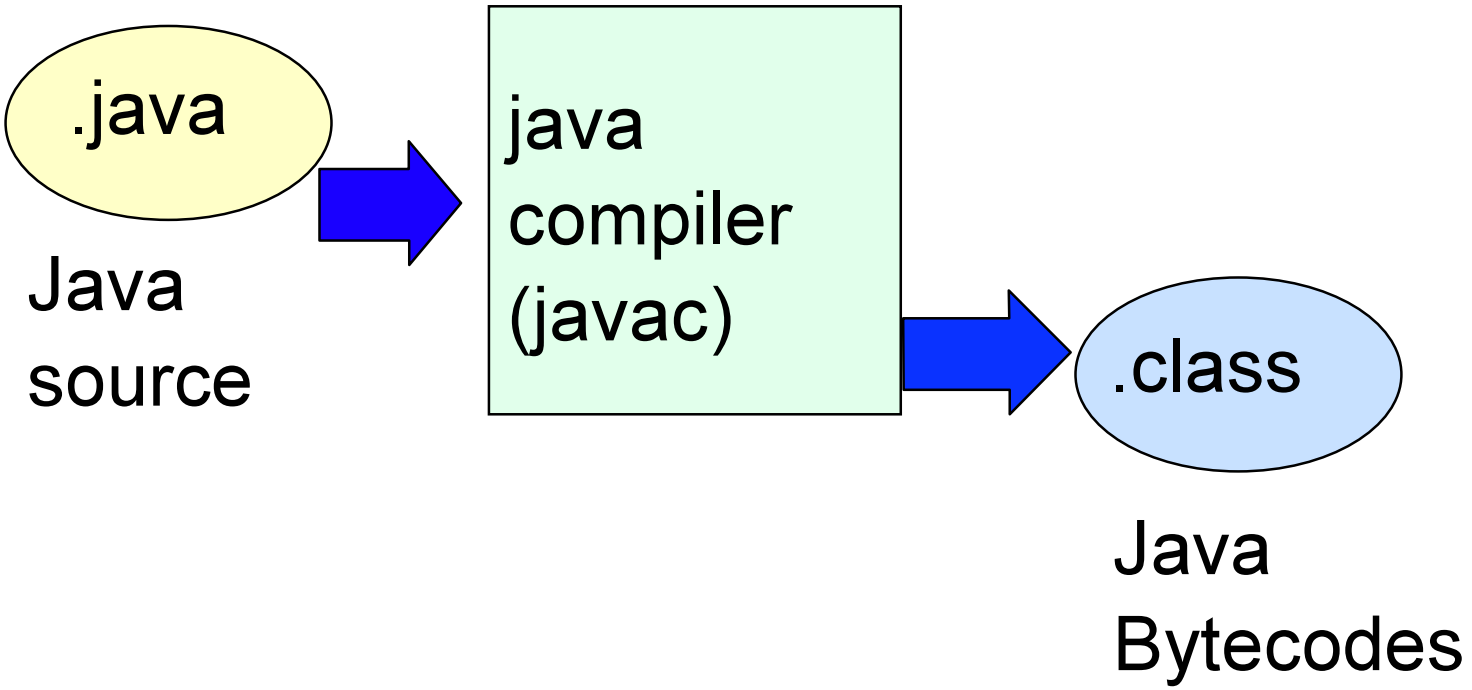
- **Package consists of**
  - JavaToDLI, AIB, DBPCB, IOPCB, and IMSInfo
  - IMSException
  - IMSTrace
  - DLIBaseSegment and DLITypeInfo
- **IMS access via CEETDLI Interface using JavaToDLI.execute() functions**
- **Application is responsible for conversion between UNICODE and EBCDIC**

JavaToDLI.execute("GU", ioaib, ioArea);

```
public class SampleMPP extends IMSApplication
{
  static AIB ioaib = new AIB("IOPCB", 100);  // AIB for messages
  static byte [] ioArea = new byte[100];     // IO area for DB/msg
  static int tranCodeLength = 8;             // Length of tran name

  public static void main( String argv[]) {
     SampleMPP sample = new SampleMPP();
     sample.begin();

  public void doBegin( ) throws IMSException  {
    JavaToDLI.execute("GU", ioaib, ioArea);

    String inputMessage
                 = new String(ioArea, tranCodeLength + 4).trim();

    IMSTransaction.getTransaction().commit();
  }
     ....
} // end SampleMPP
```

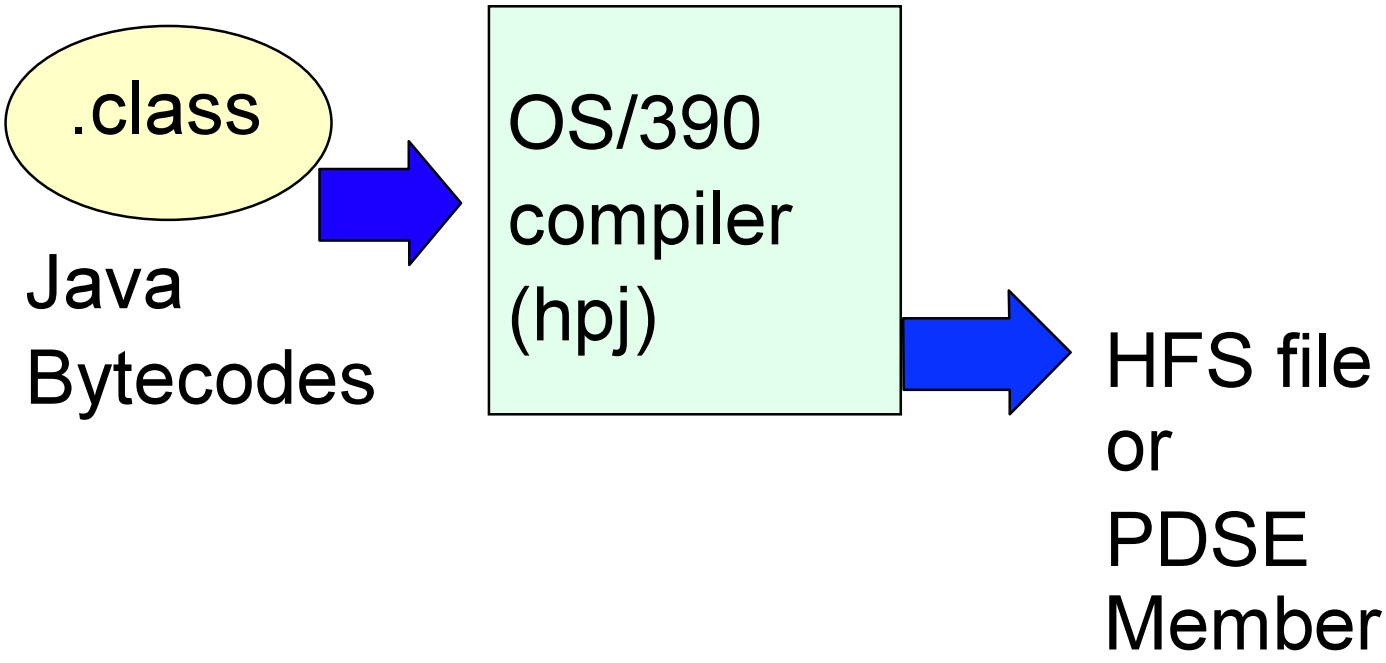
JAVA Compiler

- javac



High Performance JAVA

- OS/390 HPJ



Preparing the Program for Execution

- Using a UNIX shell
  - Compile the Java source files

    Ex. **javac** dealership/application/*.java
    javac dealership/database/*.java
    where "dealership/application/" provides the path to the source

  - Bind the class files into an executable member of a PDSE

    Ex. **hpj** -make dealership.application.IMSAuto
    -o"//'IMS.JAVA.PGMLIB(IMSAUTO)'"
    -alias=IMSAUTO
    -target=IMS
    -lerunopts="ENVAR('CLASSPATH=/usr/lpp/hpj/lib:/imsjava/imsjava.jll ',
    'IBMHPJ_HOME=/usr/lpp/hpj ')"
    -exclude=com.ibm.ims.db.*
    -exclude=com.ibm.ims.application.*
    -exclude=com.ibm.ims.base.*

NOTE: class files can be bound into HFS files but PDSEs can be put into ELPA which can improve performance.

Preparing the Program for Execution...

- Prepare the dependent region environment
  - Include the PDSE containing the Java executables in the STEPLIB concatenation

```
//STEPLIB DD DSN=IMS.JAVA.PGMLIB,DISP=SHR
```

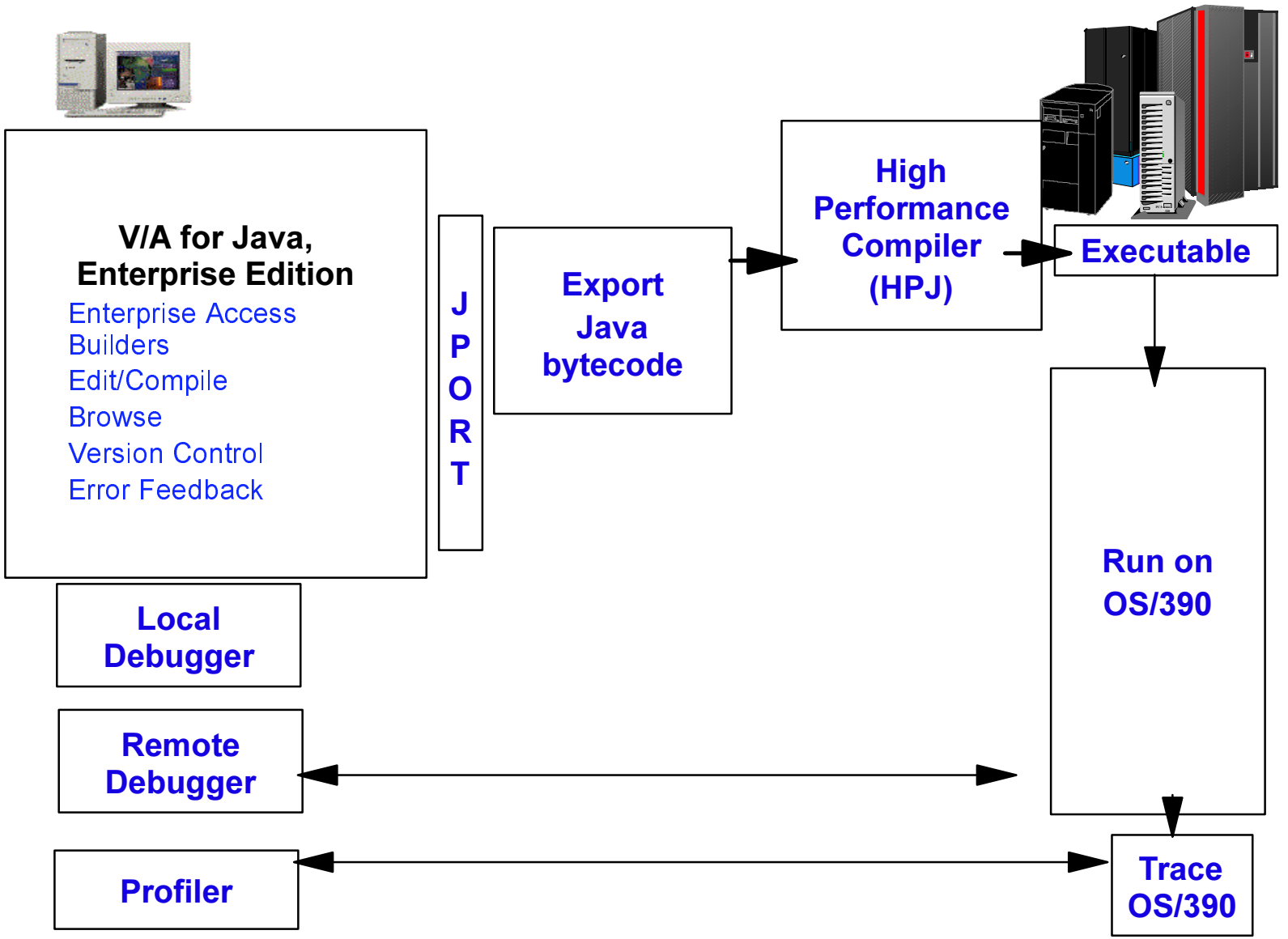
where IMS.JAVA.PGMLIB is the PDSE that contains the DLL that was bound using HPJ

  - Also include the datasets for the IMS Java class libraries and HPJ class libraries in the STEPLIB:

```
//          DD DSN=HLQ.SDFSJLIB,DISP=SHR
//          DD DSN=VAJAVA.SHPJMOD,DISP=SHR
//          DD DSN=VAJAVA.SHPOMOD,DISP=SHR
```

where The HLQ.SDFSJLIB is the IMS Java library specified during installation and VAJAVA.SHPxxxx are the HPJ libraries

IMS Java Development Scenario



Considerations

- Use of AIB requires PCBNAMES in PSB
- Cannot call COBOL subroutines
- Must be single-threaded
- Use PDSE vs HFS file for executable
- WHERE clause usage in JDBC
  - only fields defined in DBD
- Inserting database record

Debugging

- IMSExceptions
  - Extends java.lang.Exception
- IMSTrace
  - Constructor
  - Methods
  - Data

Abends

- U118
  - Application did not issue explicit commit
- U475
  - Application not supported in IMS Batch
- U778
  - ROLLBACK failure

Getting Started - Reference Materials

- **Java for OS/390**
  - Program Directory (GI10-0614)
  - information on "http://www/s390.ibm.com/Java"
- **VisualAge for Java, Enterprise Edition for OS/390 (5655-JAV)**
  - Program Directory (GI10-4949)
  - VisualAge for Java ET/390 Reference
- **IMS Java**
  - IMS Java User's Guide (ZES1-2233)

What is IMS Java? -- Summary

- **Function that enables the use of Java for IMS application programming**
  - Requires only basic IMS knowledge
  - Supports IMS application services
  - Provides a foundation for the visual tools
- **IMS Java classes**
  - Support conversational and non-conversational transactions
  - Support MFS options
  - Run in IMS dependent regions
  - Provide JDBC access to IMS DB
  - Support JDBC/SQLJ access to DB2
  - HPJ complied support
    - JVM support later

RETURN TO INDEX