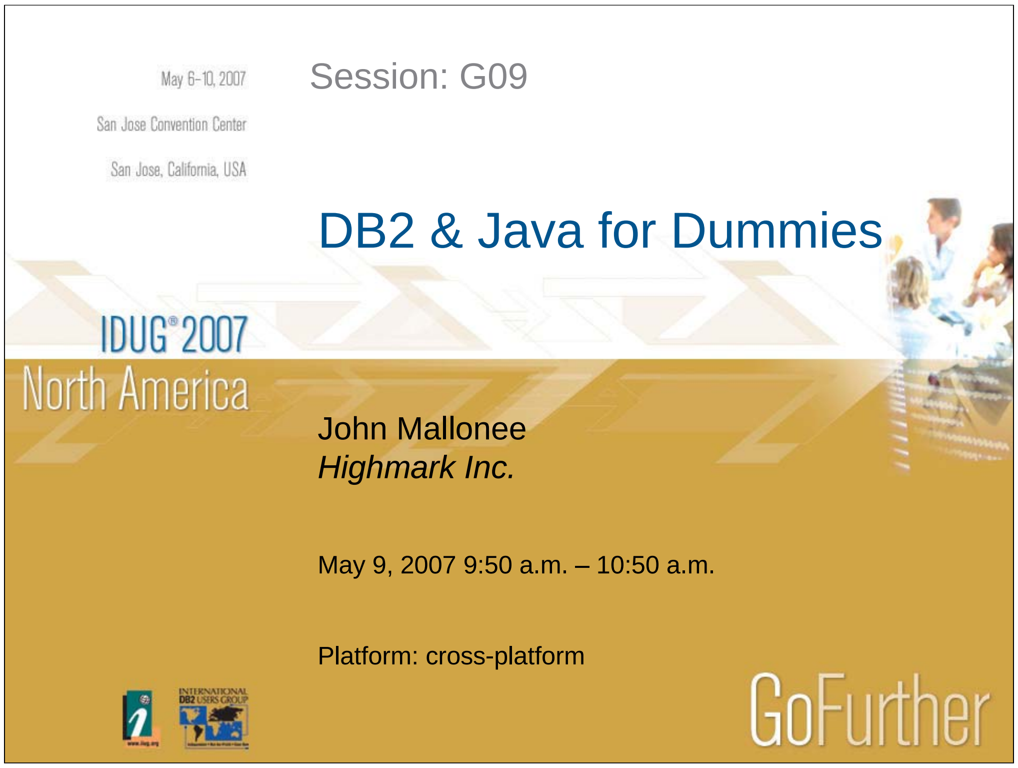


May 6–10, 2007

San Jose Convention Center

San Jose, California, USA

Session: G09

DB2 & Java for Dummies

IDUG® 2007 North America

John Mallonee
Highmark Inc.

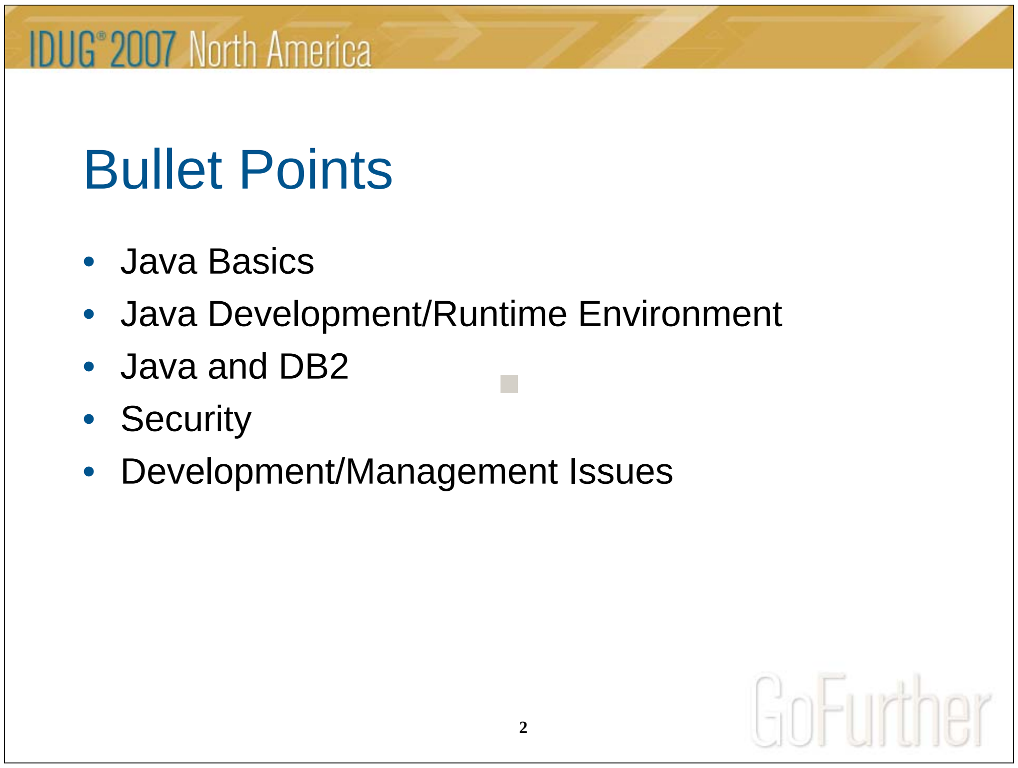
May 9, 2007 9:50 a.m. – 10:50 a.m.

Platform: cross-platform

GoFurther

DB2 & Java for Dummies

Even though the language has been around for a while now, not everyone has had a chance to get experience with Java. This presentation will try to demystify some of the basic terminology and concepts of Java, and how it works with DB2, including connectivity to the mainframe, security, and management issues.



IDUG® 2007 North America

Bullet Points

- Java Basics
- Java Development/Runtime Environment
- Java and DB2
- Security
- Development/Management Issues

2

GoFurther

I. Java basics

a. Java basics - .java, .class, compiler, JVM, byte code, JRE, Object-oriented

b. J2EE, J2SE, JDK

c. structure - web, batch, data access, business logic

II. Java Developoment/Runtime Environment

a. JAR's, EAR's, WAR's - how build's are done transferred to server

b. web apps - servlets, JSP's

c. server environment - property settings, JAR's, base JAR's, JDK version, etc.

d. bouncing application servers - property file changes

III. Security

a. datasource vs. property file

b. SQLJ vs. JDBC - package vs. individual, id or secondary auth id

c. third party application security

IV. Java and DB2 Connection

a. API's vs. drivers

b. JDBC vs. SQLJ

c. db drivers, datasource, and security

d. persistence - EJB's, open source, DAO's

e. desktop vs. server connections

V. Development/Management Issues

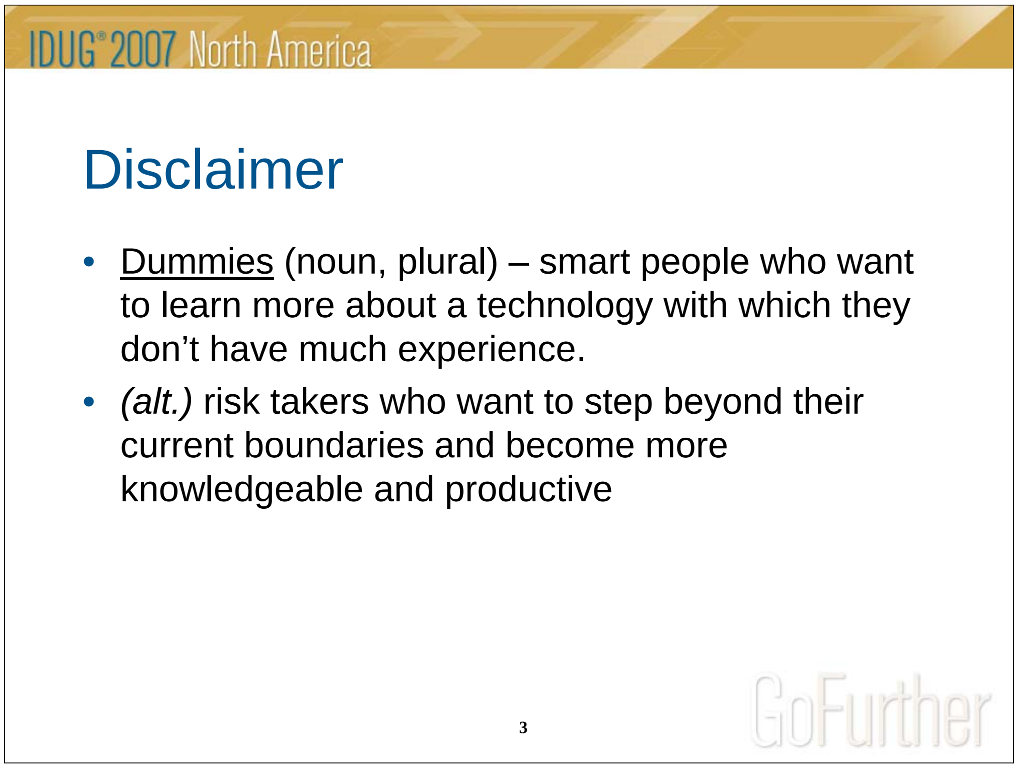
a. tracing - logging, remote threads, monitoring

b. compared to mainframe

c. desktop vs. server

d. what can go wrong?

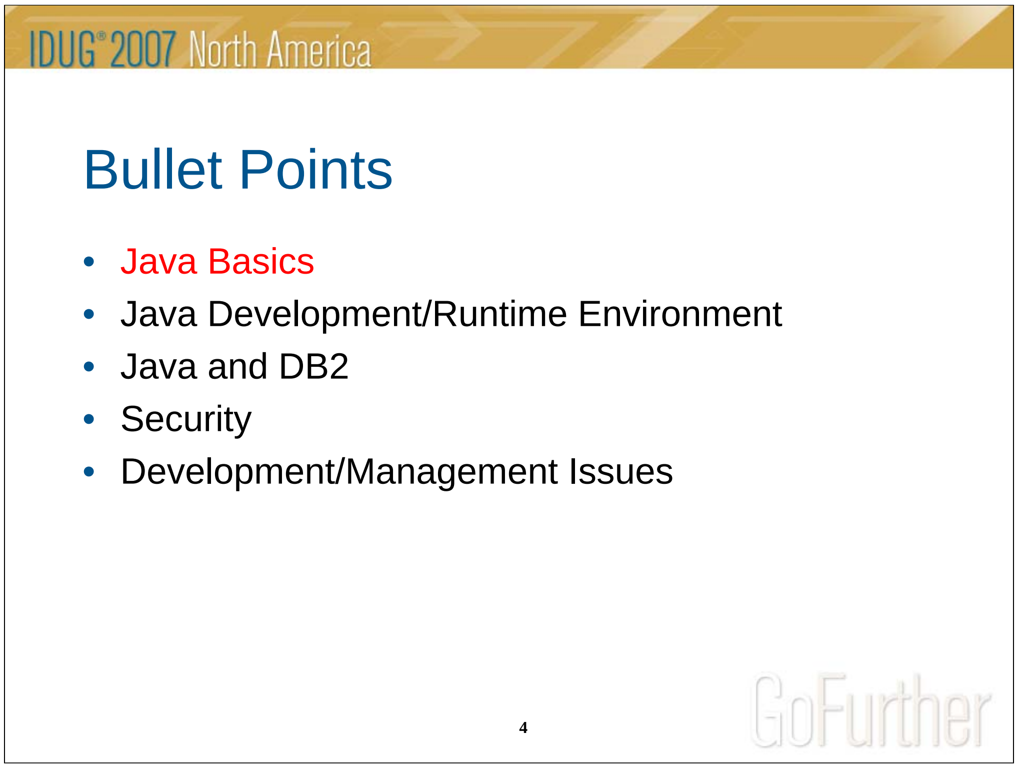
e. what can change?



Disclaimer

- Dummies (noun, plural) – smart people who want to learn more about a technology with which they don’t have much experience.
- *(alt.)* risk takers who want to step beyond their current boundaries and become more knowledgeable and productive

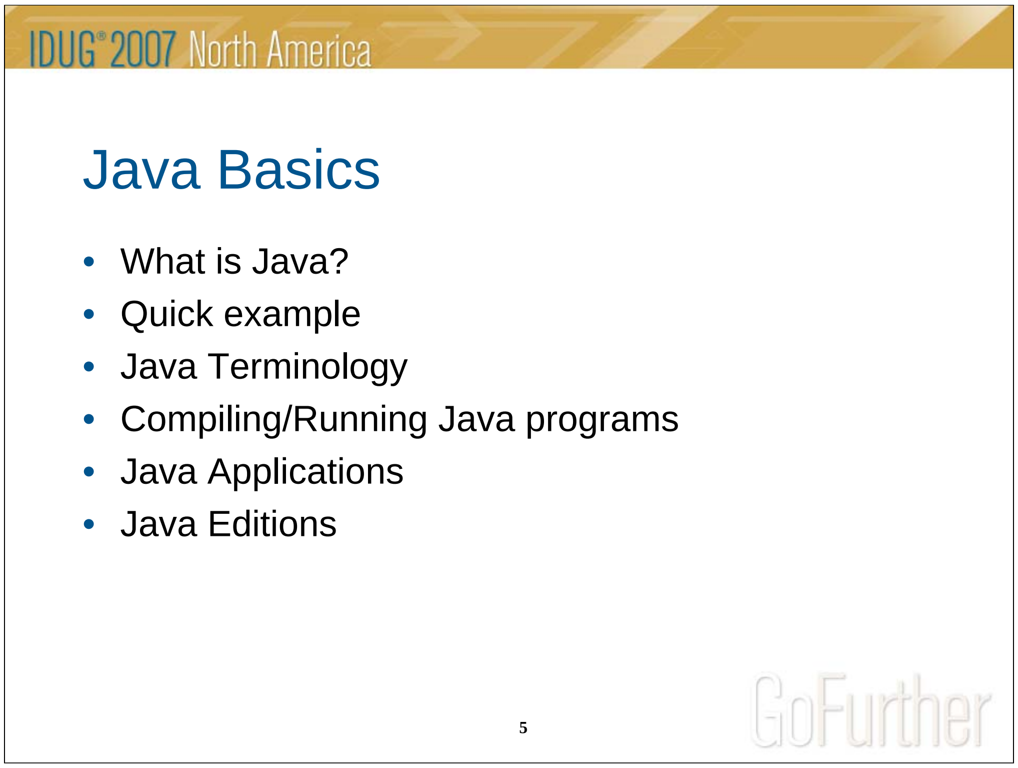
GoFurther



Bullet Points

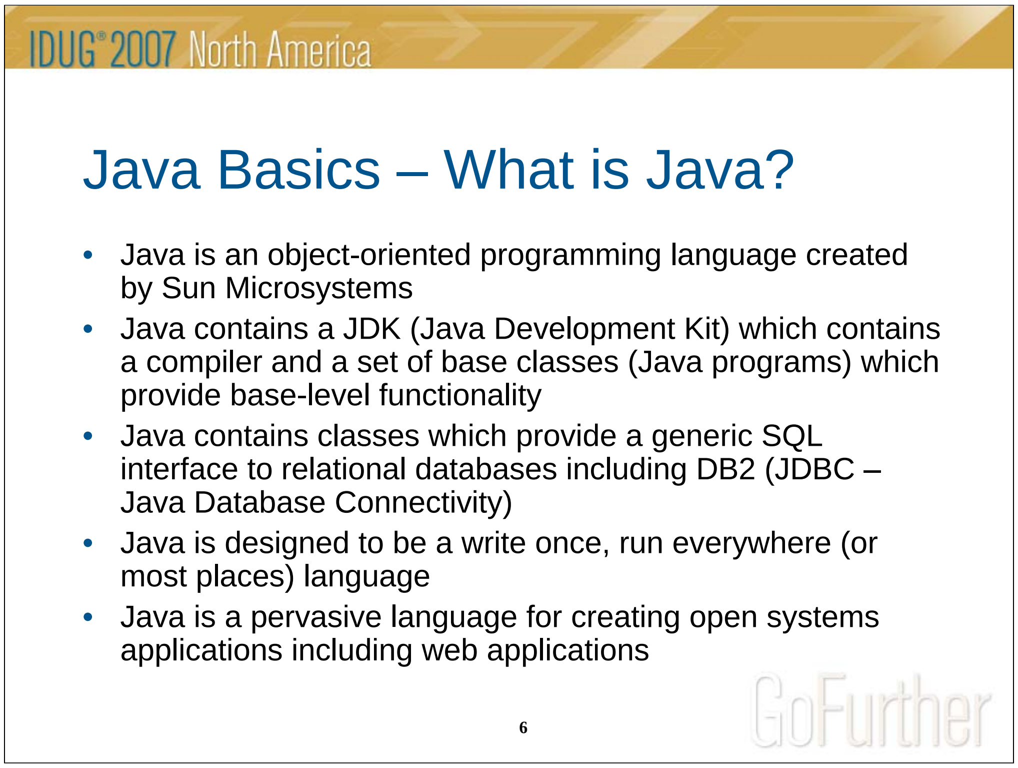
- Java Basics
- Java Development/Runtime Environment
- Java and DB2
- Security
- Development/Management Issues

GoFurther



Java Basics

- What is Java?
- Quick example
- Java Terminology
- Compiling/Running Java programs
- Java Applications
- Java Editions


IDUG® 2007 North America

Java Basics – What is Java?

- Java is an object-oriented programming language created by Sun Microsystems
- Java contains a JDK (Java Development Kit) which contains a compiler and a set of base classes (Java programs) which provide base-level functionality
- Java contains classes which provide a generic SQL interface to relational databases including DB2 (JDBC – Java Database Connectivity)
- Java is designed to be a write once, run everywhere (or most places) language
- Java is a pervasive language for creating open systems applications including web applications

6

GoFurther

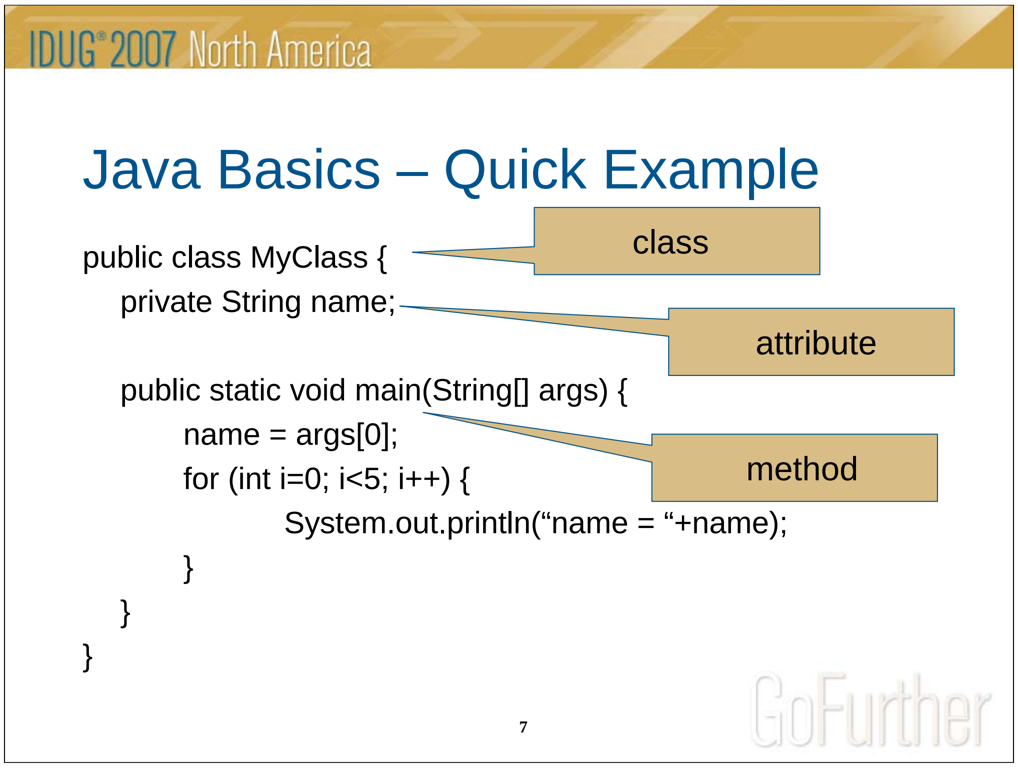
Java Basics – What is Java?

This slide says a lot, but one of the primary goals is to be able to code an application in one language that can be deployed and run on "any" platform. To a great deal this goal is met, but there are always exceptions to the rule.

As a result of meeting this goal, Java has become a natural fit for web applications since they can run on Windows or Unix or Linux or new and emerging operating systems.

Java is also more than just a language. There is base level functionality that comes in the JDK to perform standard functions. This base functionality is in the form of Java classes which are pre-coded.

Most importantly, Java supports business logic and connections to relational databases, including DB2. In fact, the interface from Java to DB2 is the same, irrespective of the version of DB2 (UDB or z/OS).


IDUG® 2007 North America
class
attribute
method
GoFurther

Java Basics – Quick Example

```
public class MyClass {
   private String name;

   public static void main(String[] args) {
         name = args[0];
         for (int i=0; i<5; i++) {
                  System.out.println(“name = “+name);
         }
   }
}
```

7

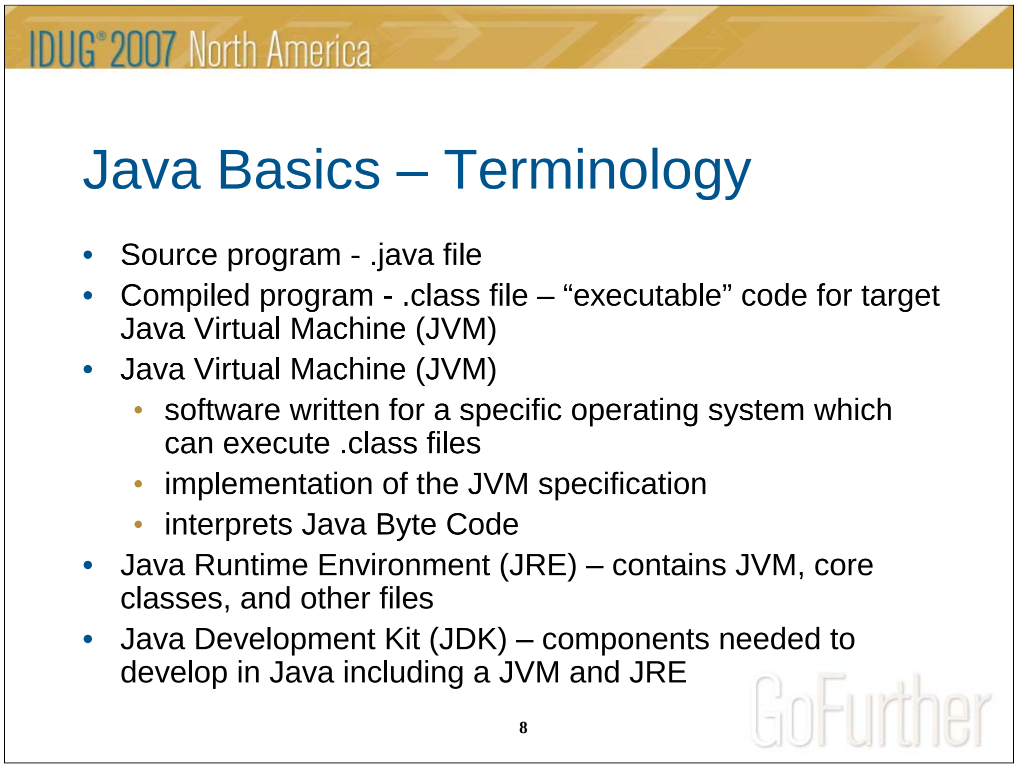
Java Basics – Quick Example

At a simple level, Java programs (or classes) are made up of 2 major components:

1. Attribute – an attribute is a defined property or field for a class. The field is embedded in the class and the class can be “instantiated”. Instantiation is the process of creating a separate and distinct copy of a class.

2. Method – a process or function related to this type of class. The process can reference attributes that belong to the class, or parameters that are passed to it.

There are many different classifications of classes, but we’ll discuss that later…


IDUG® 2007 North America

Java Basics – Terminology

- Source program - .java file
- Compiled program - .class file – “executable” code for target Java Virtual Machine (JVM)
- Java Virtual Machine (JVM)
  - software written for a specific operating system which can execute .class files
  - implementation of the JVM specification
  - interprets Java Byte Code
- Java Runtime Environment (JRE) – contains JVM, core classes, and other files
- Java Development Kit (JDK) – components needed to develop in Java including a JVM and JRE

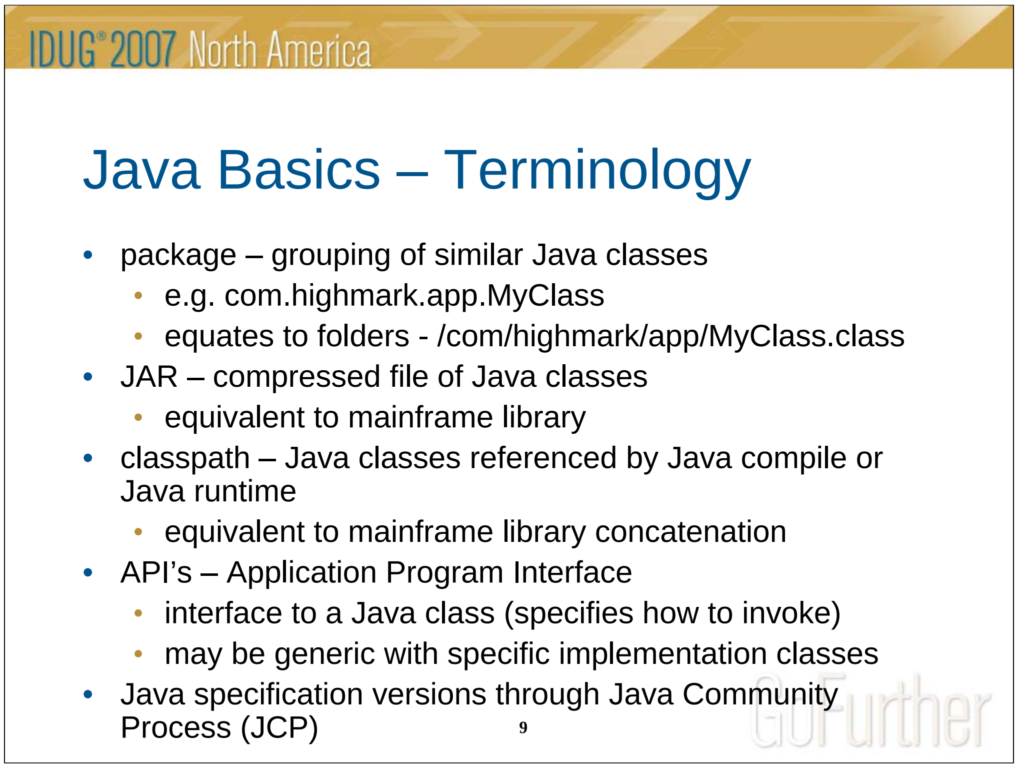
8

GoFurther

Java Basics – Terminology

To support cross-platform compatibility, “compiled” Java code is not directly executable. Java source code (.java files) are compiled to Java byte code (.class files) which are then executed with a Java Virtual Machine. The Java Virtual Machine is specific for a given platform/operating system and bridges the gap between the standard Java Byte Code and the specific internals of the runtime environment.

The Java Runtime Environment (JRE) consists of the JVM and core/base classes. These core classes provide the building blocks to do common tasks such as working with strings, performing math operations, reading from and writing to files, performing date calculations, etc.


IDUG® 2007 North America

Java Basics – Terminology

- package – grouping of similar Java classes
  - e.g. com.highmark.app.MyClass
  - equates to folders - /com/highmark/app/MyClass.class
- JAR – compressed file of Java classes
  - equivalent to mainframe library
- classpath – Java classes referenced by Java compile or Java runtime
  - equivalent to mainframe library concatenation
- API's – Application Program Interface
  - interface to a Java class (specifies how to invoke)
  - may be generic with specific implementation classes
- Java specification versions through Java Community Process (JCP)

9

GoFurther

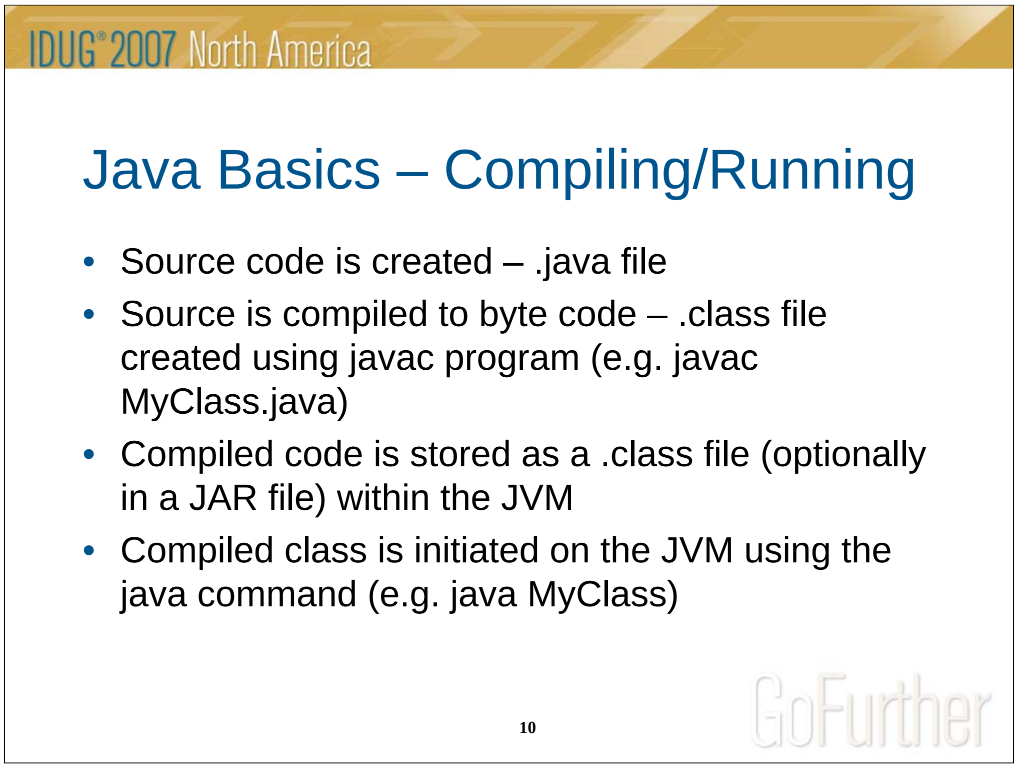
Java Basics – Terminology

Package – different from a DB2 package, a Java package is a way to combine similar classes together into one folder. The unique name for any given Java class is the combination of its package and class name. A given class name can exist in more than one package (and do different things).

JAR – this is nothing more than a zipped file of .class files.

Classpath – is a build (compile) or execution concatenation of Java classes and/or JAR files that are referenced by the class to be compiled.

API – is a generic term used to specify the way to interface with a Java class or classes. API can also refer to a set of related classes and defines the overall interface to their contained functionality.


IDUG® 2007 North America

Java Basics – Compiling/Running

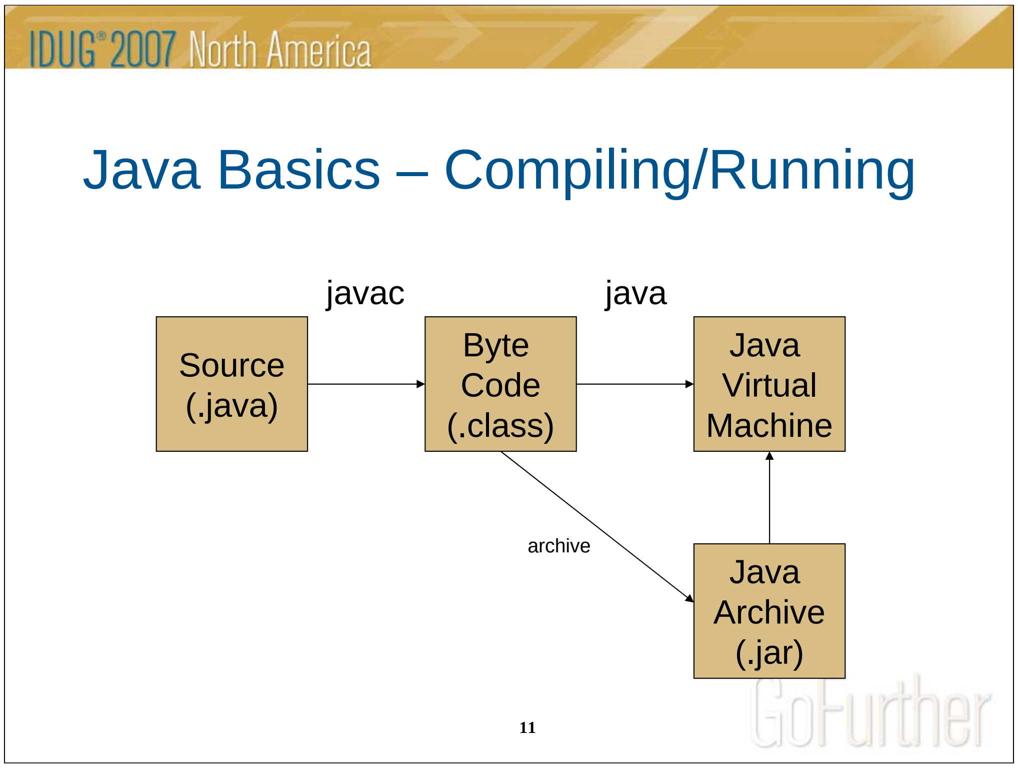
- Source code is created – .java file
- Source is compiled to byte code – .class file created using javac program (e.g. javac MyClass.java)
- Compiled code is stored as a .class file (optionally in a JAR file) within the JVM
- Compiled class is initiated on the JVM using the java command (e.g. java MyClass)

10

GoFurther

Java Basics – Compiling/Running

At a developer level, source code is created in a .java (text) file, irrespective of the platform where the source code is created. A Java compiler creates Java Byte Code to create a quasi-executable format of the class (.class file). Once compiled, the Java class is executed from within a Java Virtual Machine in the runtime environment on the target platform/operating system. Not all Java classes are directly executable, similar to how subprograms aren't directly executable in a mainframe environment. Some Java classes exist to represent and transport data (Java beans), some exist for common functionality, etc. We'll see more about the different types of Java classes later.



IDUG® 2007 North America

Java Basics – Compiling/Running

$$\begin{tikzcd}
\text{Source (.java)} \arrow[r, "\text{javac}"] & \text{Byte Code (.class)} \arrow[r, "\text{java"] \arrow[dr, "\text{archive}"'] & \text{Java Virtual Machine} \\
& & \text{Java Archive (.jar)} \arrow[u]
\end{tikzcd}$$

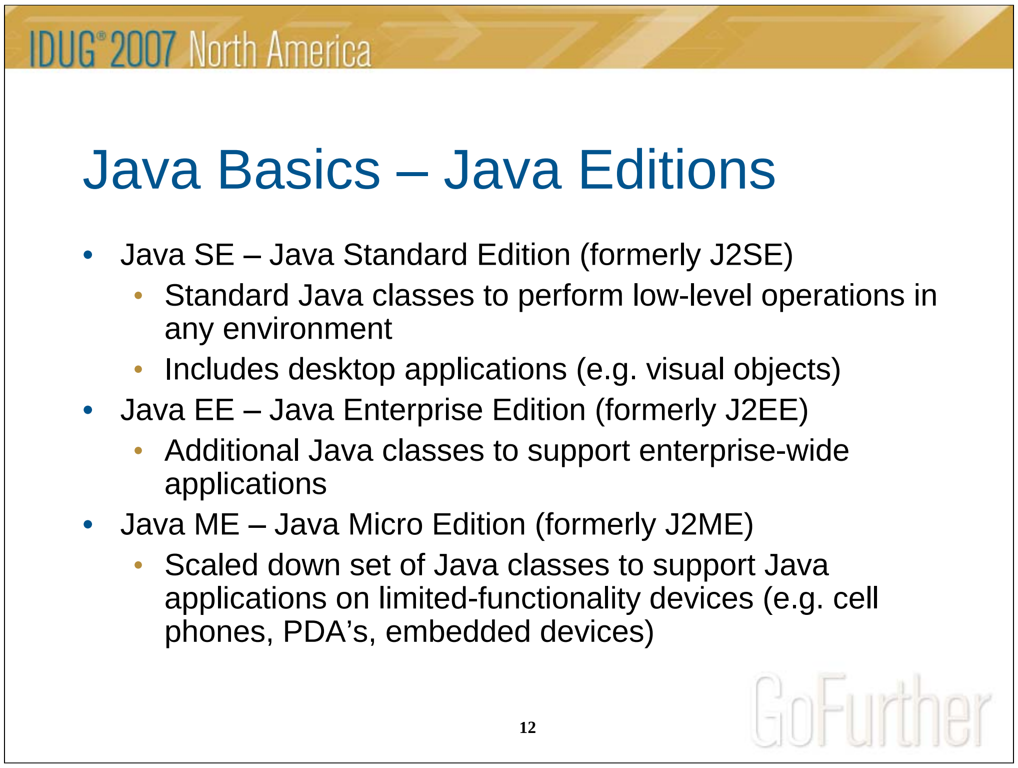
11

GoFurther

Java Basics – Compiling/Running

This slide graphically depicts the process from the previous slide. Note that after compilation that a .class file can be combined with other class files into a Java Archive (.jar) file.

JAR files are handy for distributing like classes together as a group. Compilers and JVM's recognize the JAR's and can read the .class files contained within them.


IDUG® 2007 North America

Java Basics – Java Editions

- Java SE – Java Standard Edition (formerly J2SE)
  - Standard Java classes to perform low-level operations in any environment
  - Includes desktop applications (e.g. visual objects)
- Java EE – Java Enterprise Edition (formerly J2EE)
  - Additional Java classes to support enterprise-wide applications
- Java ME – Java Micro Edition (formerly J2ME)
  - Scaled down set of Java classes to support Java applications on limited-functionality devices (e.g. cell phones, PDA's, embedded devices)

12

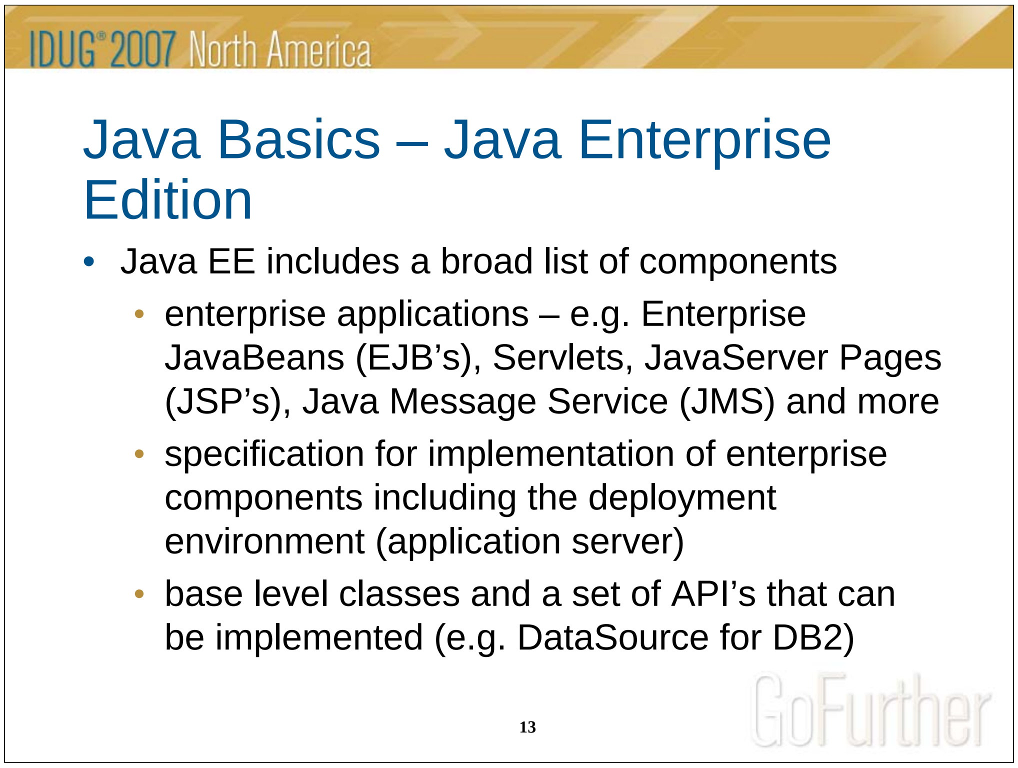
GoFurther

Java Basics – Java Editions

Java started out primarily as a set of base classes that supported core functionality and desktop applications (now J2SE). However, that's grown over time to support applications on an enterprise level (i.e. server-based applications serving many users).

Java EE is a newer term for J2EE for enterprise applications. This contains most of the functionality to build web and server-based applications including Servlets, JSP's, and message-based application components. We'll hear more about those shortly.

Java ME is a scaled down version of classes to enable Java applications to run on constrained devices such as cell phones, PDA's, and embedded devices. The concepts are the same, but the class files have to be smaller due to memory limitations on such devices.


IDUG® 2007 North America

Java Basics – Java Enterprise Edition

- Java EE includes a broad list of components
  - enterprise applications – e.g. Enterprise JavaBeans (EJB's), Servlets, JavaServer Pages (JSP's), Java Message Service (JMS) and more
  - specification for implementation of enterprise components including the deployment environment (application server)
  - base level classes and a set of API's that can be implemented (e.g. DataSource for DB2)

13

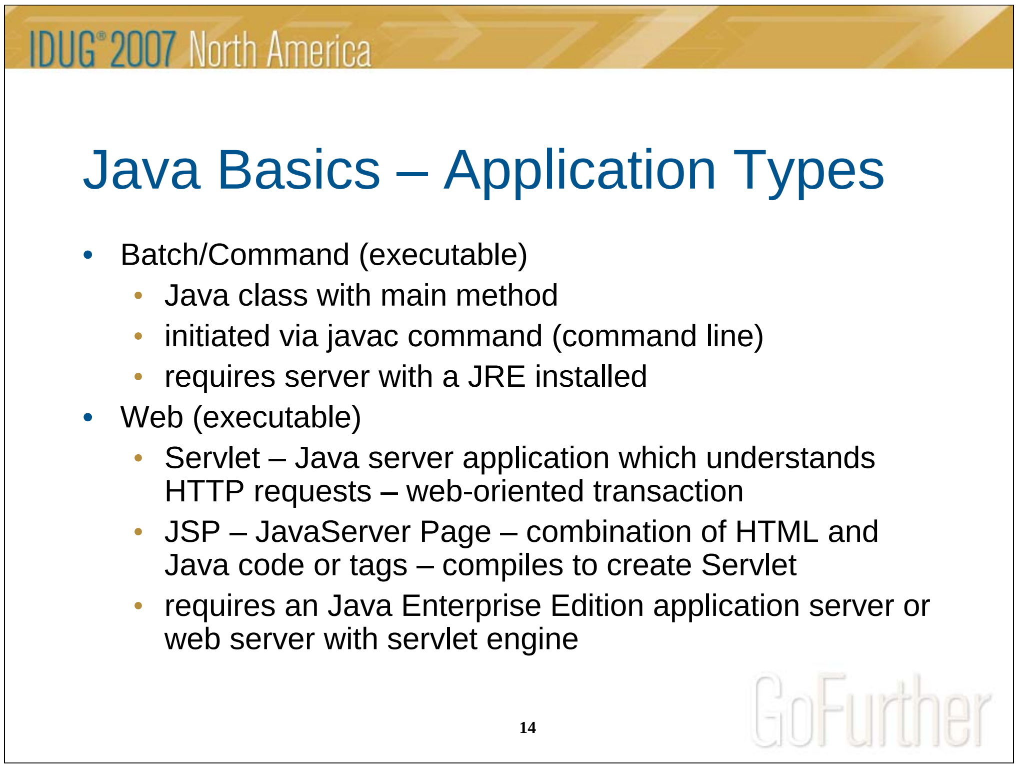
GoFurther

Java Basics – Java Enterprise Edition

Java EE is what really makes Java a usable language and environment for large scale business applications and what moved Java past the desktop.

EJB's provide the capability to manage data persistence, control message-based services, and transactions that are managed by the runtime environment instead of the applications. Additionally, EJB's can make reusable processes usable across servers and platforms.

Other web-based Java constructs are included in Java EE such as Servlets and JSP's which are web-invokable processes and dynamic HTML pages. The common API for connecting to relational databases (a DataSource class) is also included in Java EE.


IDUG® 2007 North America

Java Basics – Application Types

- Batch/Command (executable)
  - Java class with main method
  - initiated via javac command (command line)
  - requires server with a JRE installed
- Web (executable)
  - Servlet – Java server application which understands HTTP requests – web-oriented transaction
  - JSP – JavaServer Page – combination of HTML and Java code or tags – compiles to create Servlet
  - requires an Java Enterprise Edition application server or web server with servlet engine

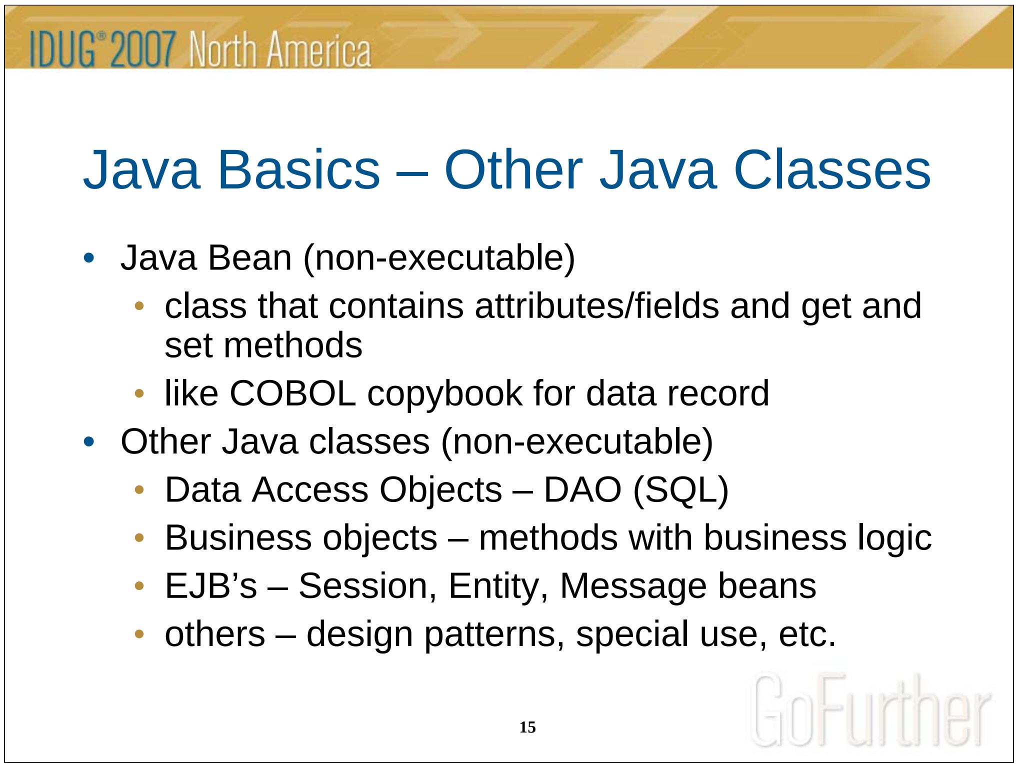
14

GoFurther

Java Basics – Application Types

There are many, many different classifications or types of Java classes. There are also many different types of applications or application components. Two very common applications are command-based applications and web applications:

1. Batch/Command – this is an automated process that is initiated by a command line interface command (java command). This is generally that "batch" applications are executed in Java.
2. Web – this is an application that is run from a web browser. A Servlet (Java class) is invoked by a URL or an action taken from an HTML form. The Servlet likely invokes other Java classes to perform business processes and generally forwards the results to a JSP page which renders the data as dynamic HTML. These applications require an application server (and associated web server) over and above a JVM to execute. More on web applications shortly.


IDUG® 2007 North America

Java Basics – Other Java Classes

- Java Bean (non-executable)
  - class that contains attributes/fields and get and set methods
  - like COBOL copybook for data record
- Other Java classes (non-executable)
  - Data Access Objects – DAO (SQL)
  - Business objects – methods with business logic
  - EJB's – Session, Entity, Message beans
  - others – design patterns, special use, etc.

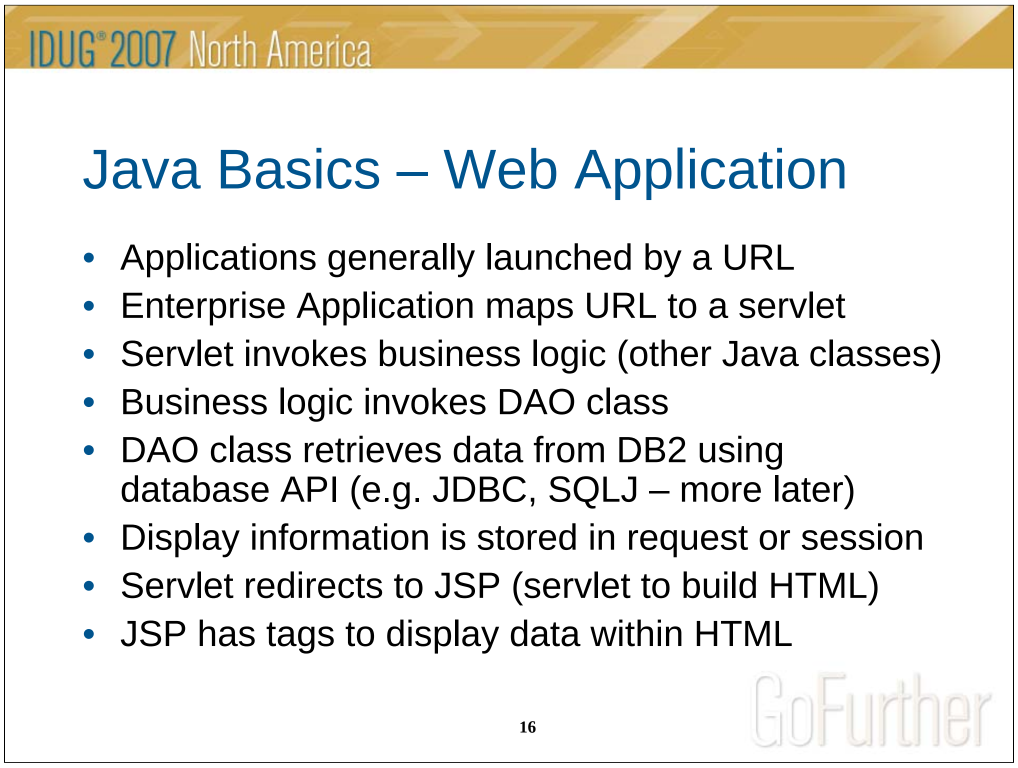
15

GoFurther

Java Basics – Other Java Classes

Java Bean – a Java bean is a Java class/object that contains data. It's similar to a COBOL copybook or record layout in a COBOL program. Each new instance (occurrence) of a bean has its own unique values for the fields contained. Beyond the fields/properties defined in the bean, the only methods are generally get and set methods (one each for each field/property). The fields/properties are private in that other classes cannot reference them directly, but must use the API (get and set methods) to read or update a value. Java beans cannot exist on their own, but are created by other Java classes (non-executable).

Others – there are numerous other classifications of Java classes. As an example, a Data Access Object (DAO) is a type of class that is created that contains methods to build and execute SQL through a generic database API (see Java and DB2 section). Business logic is contained in methods in a class sometimes referred to as a Business object. Entity beans are a Java EE specification for handling transactions, data persistence, and message brokers (more later). There are numerous other design patterns, but eventually, each class contains fields/properties (optional) and methods.



IDUG® 2007 North America

Java Basics – Web Application

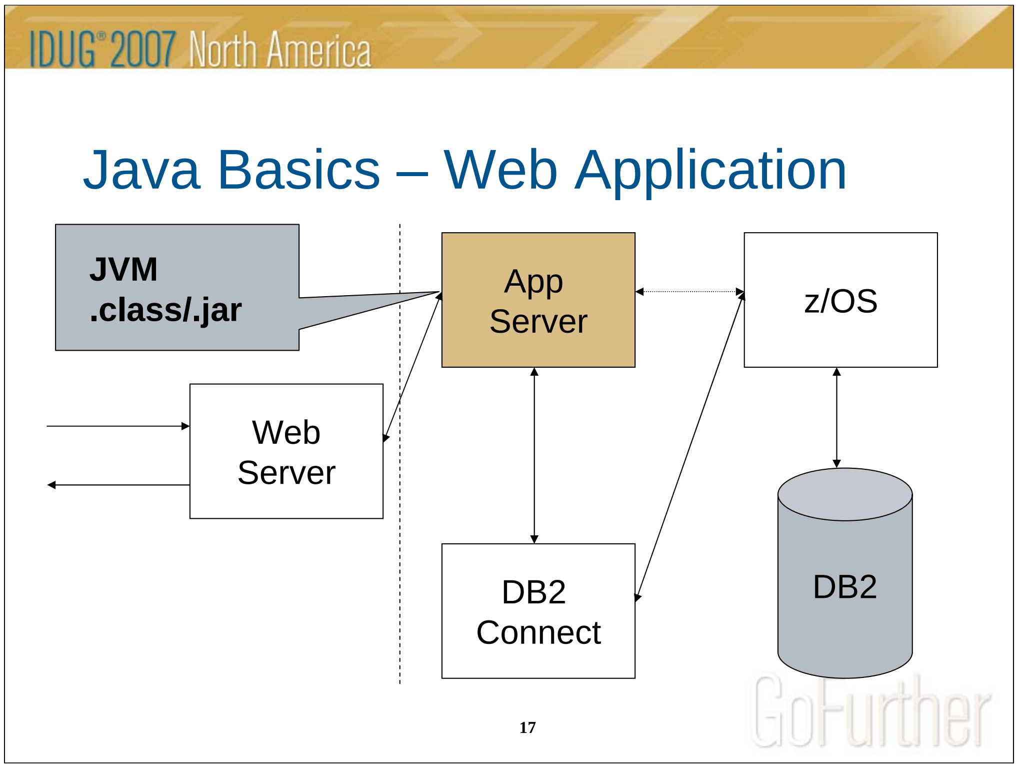
- Applications generally launched by a URL
- Enterprise Application maps URL to a servlet
- Servlet invokes business logic (other Java classes)
- Business logic invokes DAO class
- DAO class retrieves data from DB2 using database API (e.g. JDBC, SQLJ – more later)
- Display information is stored in request or session
- Servlet redirects to JSP (servlet to build HTML)
- JSP has tags to display data within HTML

16

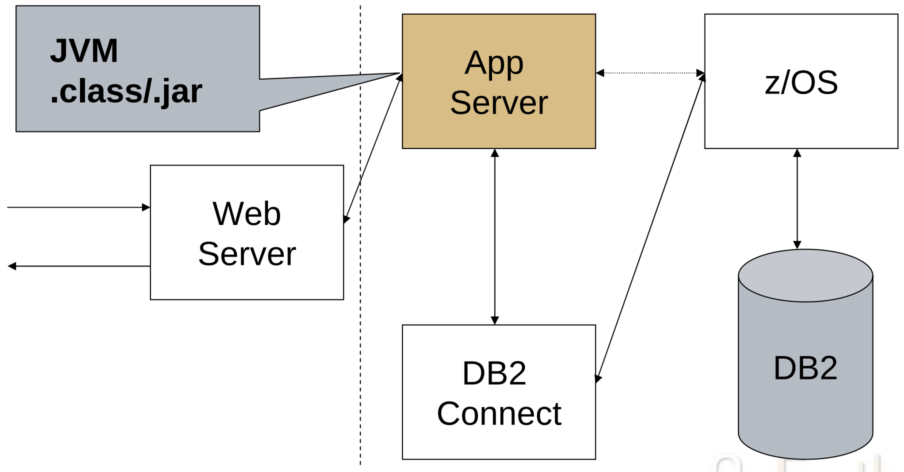
GoFurther

Java Basics – Web Application

The Java EE specification includes a lot of support for web applications (applications invoked through a web browser). These applications can be complex and include Java, HTML, images, documents, and more. A URL will ultimately trigger a Servlet (a special type of Java class that understands an HTTP (hypertext transfer protocol) or web request) which controls the process. The Servlet then invokes other classes to process the request from the browser, including processing validation, business logic, and database calls. When the processing logic is complete, the resulting data is inserted into an HTML stream (dynamic HTML) through a JavaServer Page (JSP) which is then returned as an HTML response to the web browser.



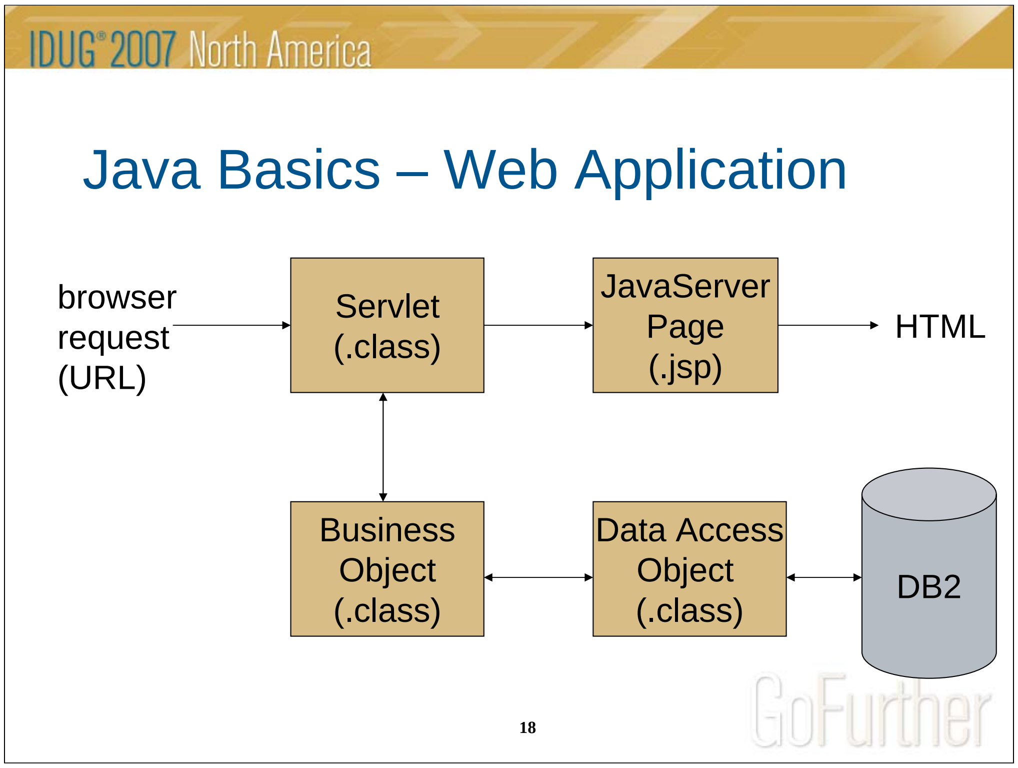
Java Basics – Web Application



Java Basics – Web Application

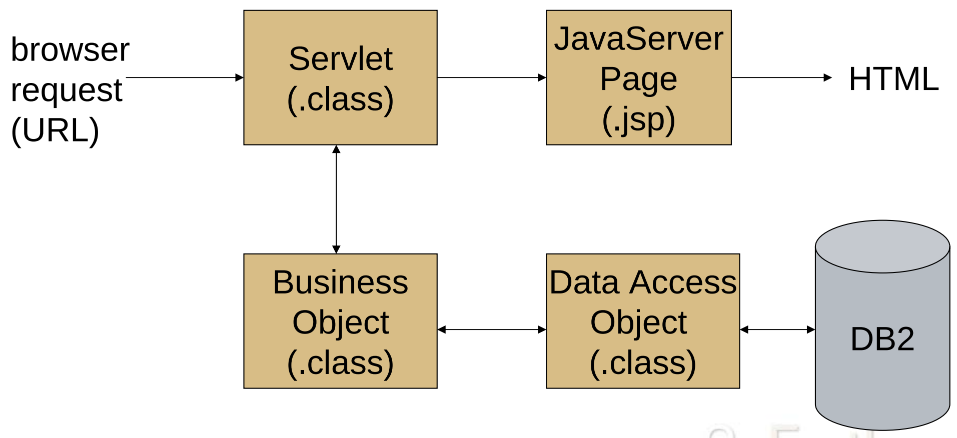
On a high level, web application requests in the form of HTTP requests and responses are funneled through a web server which inspects the URL and passes the request to a Java EE Application Server. The Application Server contains the web application including the Java code. The Application Server contains connections to the database server (in this example through DB2 Connect on a separate server) which contains the DB2 data. See the Java and DB2 section for more on the connection between the application and DB2.

NOTE: the dotted line between the App Server and z/OS represents an optional Type 4 (all Java, no middleware) driver. More later in the Java and DB2 section.



IDUG® 2007 North America

Java Basics – Web Application



18

GoFurther

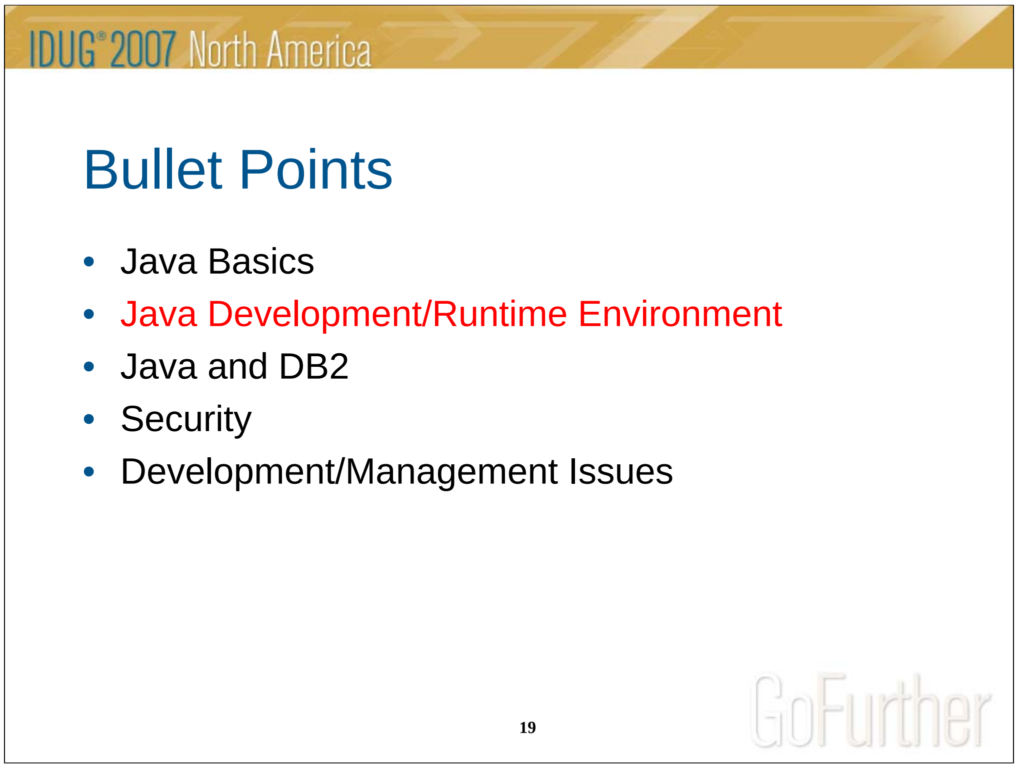
Java Basics – Web Application

This picture is a lower level view of the components of the web application that contained within the Application Server.

A request is received from a web browser either in the form of an entered address/URL, or a link or button is clicked on a displayed HTML page.

The Servlet receives the request including any data (URL parameters or HTML form fields) and invokes other classes to process the request through any business process.

The resulting business data is then saved in a Java object(s) in memory which is combined with a JSP page to display a dynamic HMTL page.

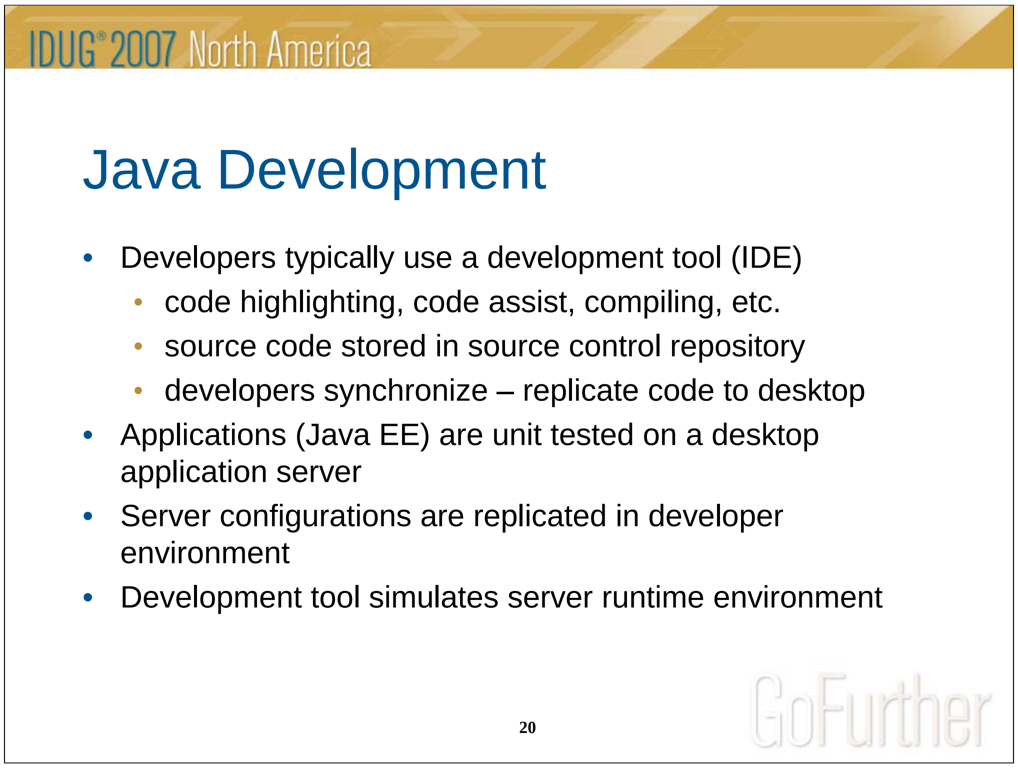

IDUG® 2007 North America

Bullet Points

- Java Basics
- Java Development/Runtime Environment
- Java and DB2
- Security
- Development/Management Issues

19

GoFurther


IDUG® 2007 North America

Java Development

- Developers typically use a development tool (IDE)
  - code highlighting, code assist, compiling, etc.
  - source code stored in source control repository
  - developers synchronize – replicate code to desktop
- Applications (Java EE) are unit tested on a desktop application server
- Server configurations are replicated in developer environment
- Development tool simulates server runtime environment

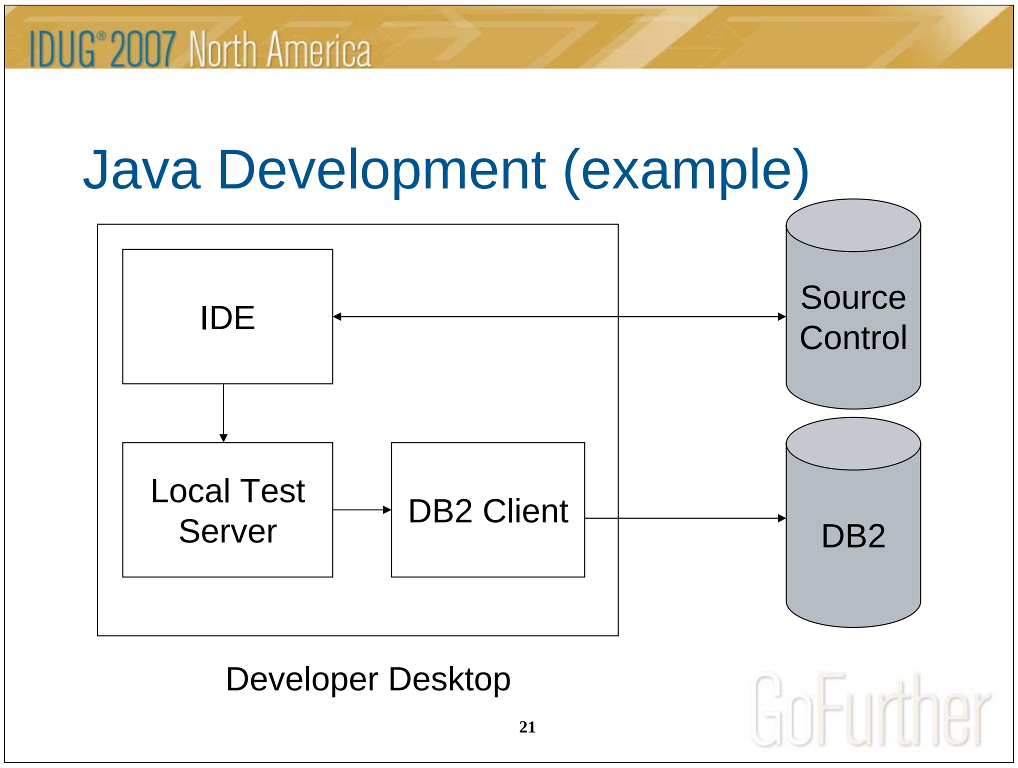
20

GoFurther

Java Development

Java development can be done in several ways, but is typically done using an Integrated Development Environment (IDE) tool. An IDE is software that provides for easier creating, editing, and managing of the components of an application. Java IDE’s typically provide accelerated source code creation, source code highlighting, and automatic compiling (creation of .class files). They can be configured to support a variety of development approaches. Additionally, they typically have integration with source code management software so that code can be versioned to a repository and shared with teams of developers.

In many cases, a developer’s desktop environment will include a test application server environment to simulate the running of the application on a test or production server. In this way, developers can unit test an entire application (even a web application) on their desktops before deploying to a test server. This highlights one of the values of Java in that it can run on multiple platforms (e.g. Windows on a developer desktop and Linux on a test or production server).



IDUG® 2007 North America

Java Development (example)

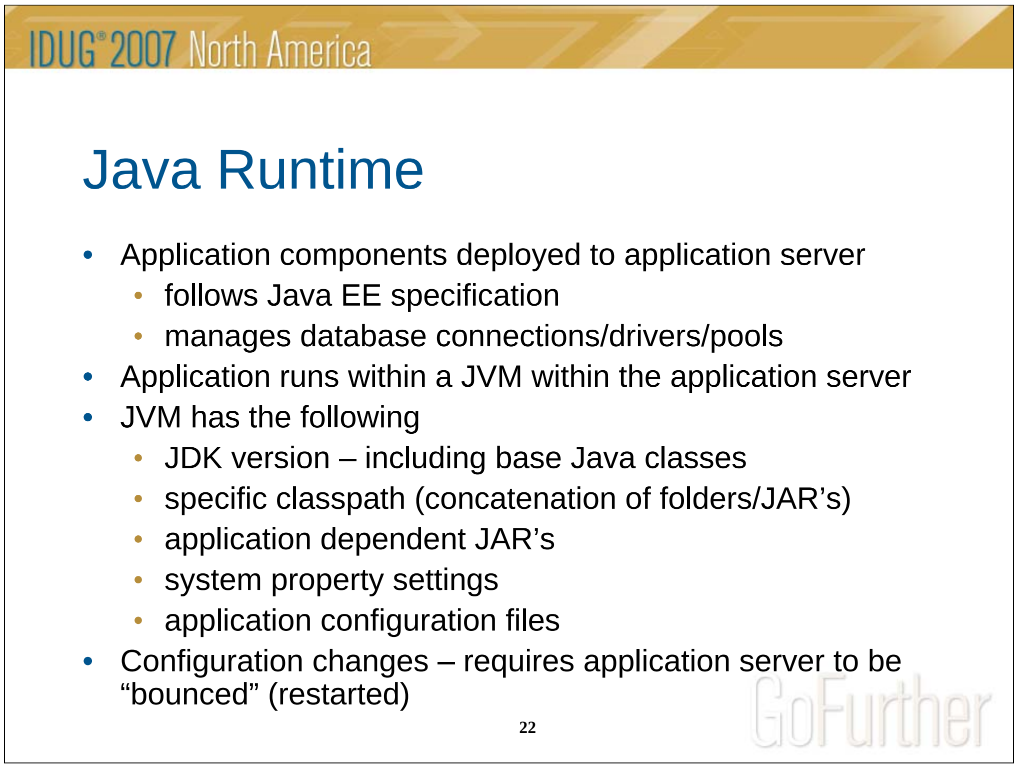
21

Java Development (example)

This chart shows a high level picture of a developer's desktop.

The IDE allows the developer to create and manage code and generally integrates with a local copy of a test application server (Local Test Server). The Local Test Server is then configured with a DataSource to use a local copy of a DB2 Client to communicate with the DB2 database server. DB2 can be local to the desktop or remote on a test server, or the DataSource can connect to another database local to the desktop such as Cloudscape or DB2 Express.

The IDE connects to a Source Control repository on a server to store, manage, and share application source code.


IDUG® 2007 North America

Java Runtime

- Application components deployed to application server
  - follows Java EE specification
  - manages database connections/drivers/pools
- Application runs within a JVM within the application server
- JVM has the following
  - JDK version – including base Java classes
  - specific classpath (concatenation of folders/JAR's)
  - application dependent JAR's
  - system property settings
  - application configuration files
- Configuration changes – requires application server to be "bounced" (restarted)

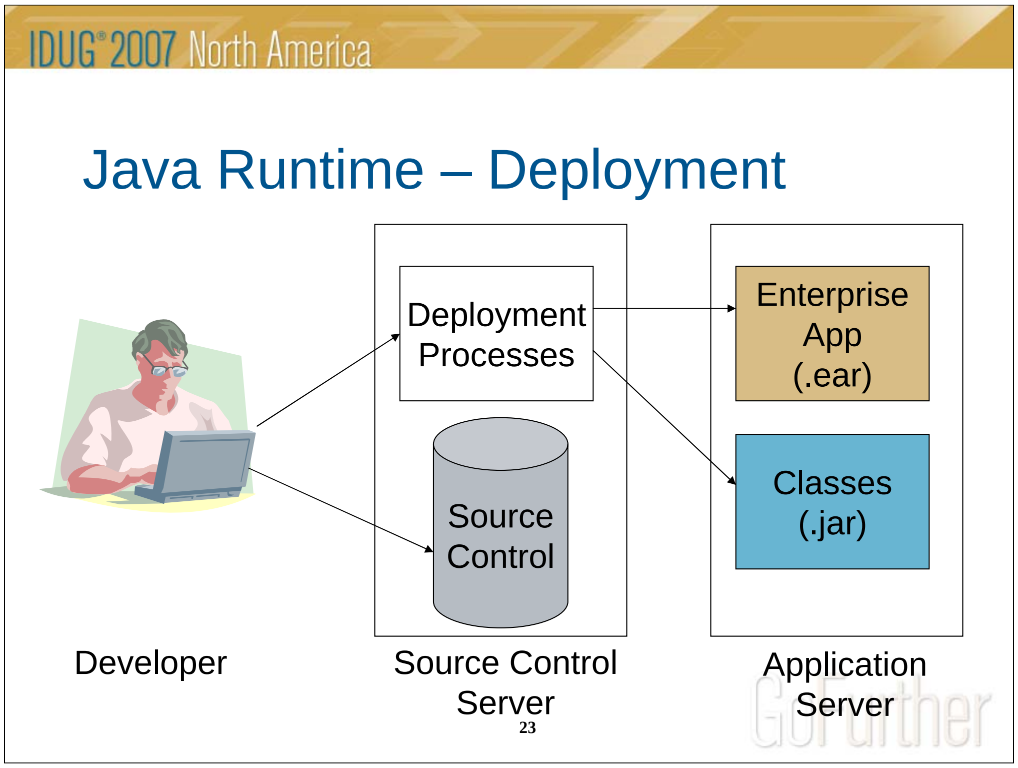
22

GoFurther

Java Runtime

Java applications require a Java Virtual Machine (JVM) to run, but the JVM is typically part of a larger application server. The Java EE specification outlines the requirements of a standard application server environment and many vendors and open source foundations provide application servers that meet the specification.

The application server environment has many configuration possibilities. These configurations can be more complicated than the application itself and can lead to many of the issues encountered by applications. This requires developers to know much more than Java to test and application and prepare it for execution on a test or production server.

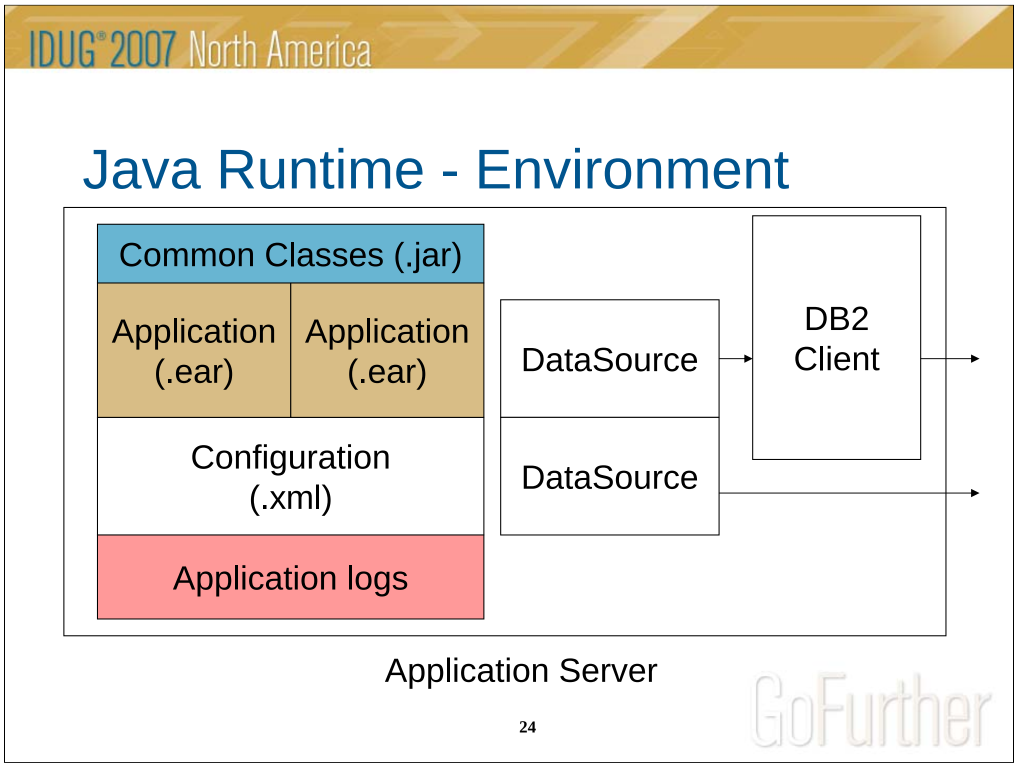


Java Runtime - Deployment

There are many ways to deploy applications to a test or production server. From an audit perspective, it is generally accepted that deployments are automated vs. manual.

In this scenario, a developer replicates source code to a Source Control repository. On the Source Control Server, there are automated processes which can be initiated by the developer to build (compile and assemble) the application, and then distribute and deploy it to the target application server.

Enterprise applications in Java EE are contained within .ear files. Additionally, shared or reusable code can be contained within a .jar file to be accessed by multiple applications.

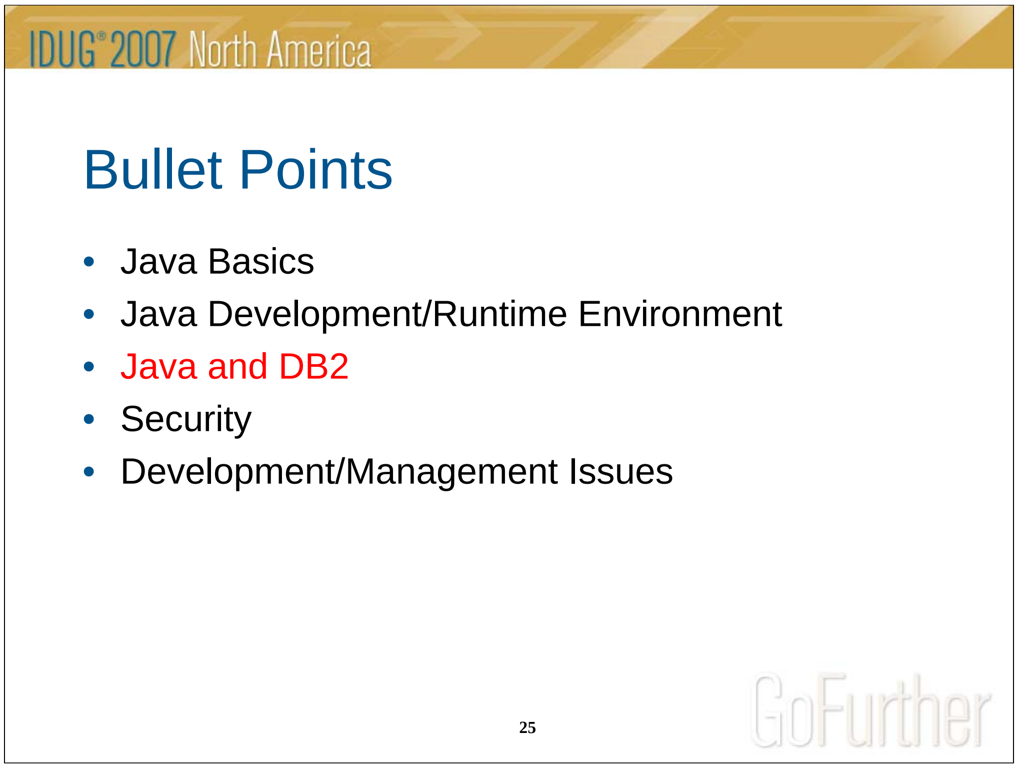


Java Runtime - Environment

Java Runtime – Environment

This chart depicts some of the major components within a Java EE Application Server. One or more enterprise applications are deployed as .ear files (which are expanded on the server) and configured through .xml files. Common code is stored in a folder on the server that is configured in the classpath of the application server's JVM settings to be accessible by applications. Application log files are stored in an appropriate server folder for the application(s) to write debug or error messages during execution.

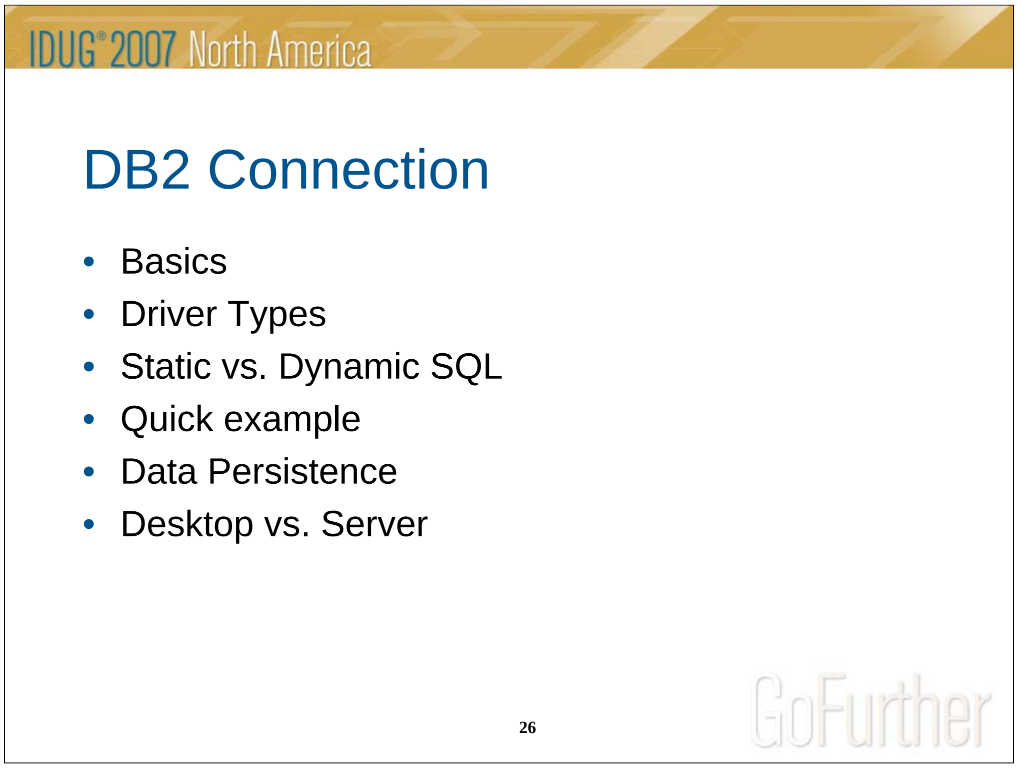
One or more DataSource is configured within the application server management to connect to the required relational database(s). In the case of DB2, DB2 Client (along with DB2 Connect) is sometimes used to bridge the network from the application server to the database server (e.g. DB2 on z/OS).



Bullet Points

- Java Basics
- Java Development/Runtime Environment
- Java and DB2
- Security
- Development/Management Issues

GoFurther



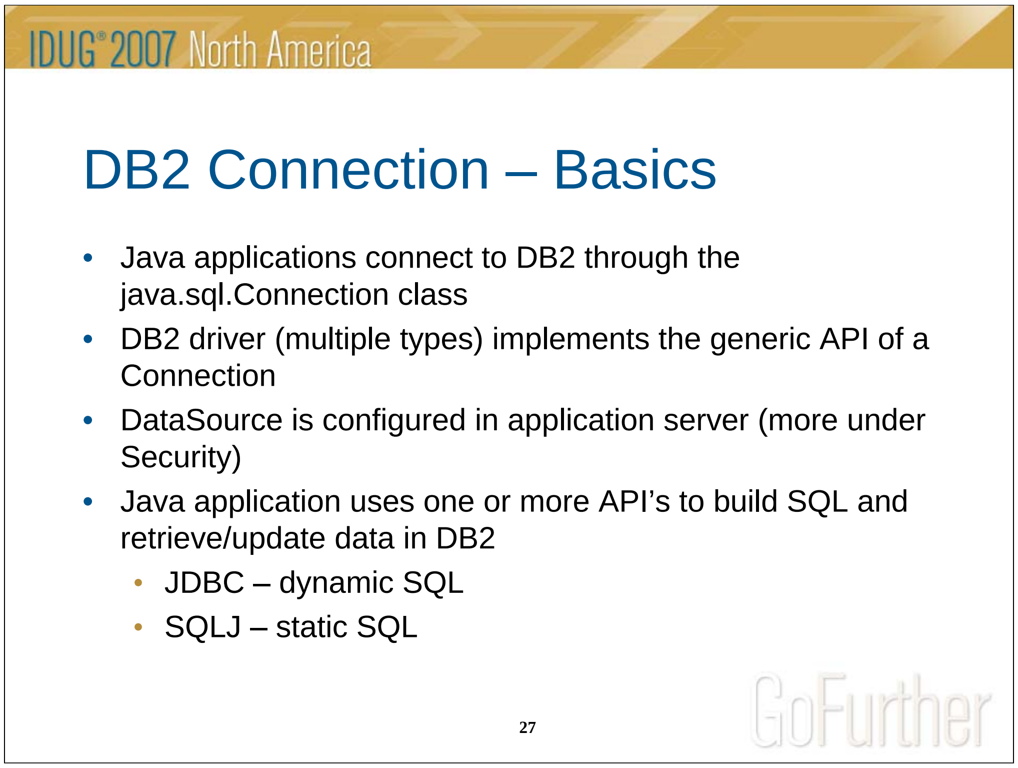
IDUG® 2007 North America

DB2 Connection

- Basics
- Driver Types
- Static vs. Dynamic SQL
- Quick example
- Data Persistence
- Desktop vs. Server

26

GoFurther


IDUG® 2007 North America

DB2 Connection – Basics

- Java applications connect to DB2 through the java.sql.Connection class
- DB2 driver (multiple types) implements the generic API of a Connection
- DataSource is configured in application server (more under Security)
- Java application uses one or more API's to build SQL and retrieve/update data in DB2
  - JDBC – dynamic SQL
  - SQLJ – static SQL

27

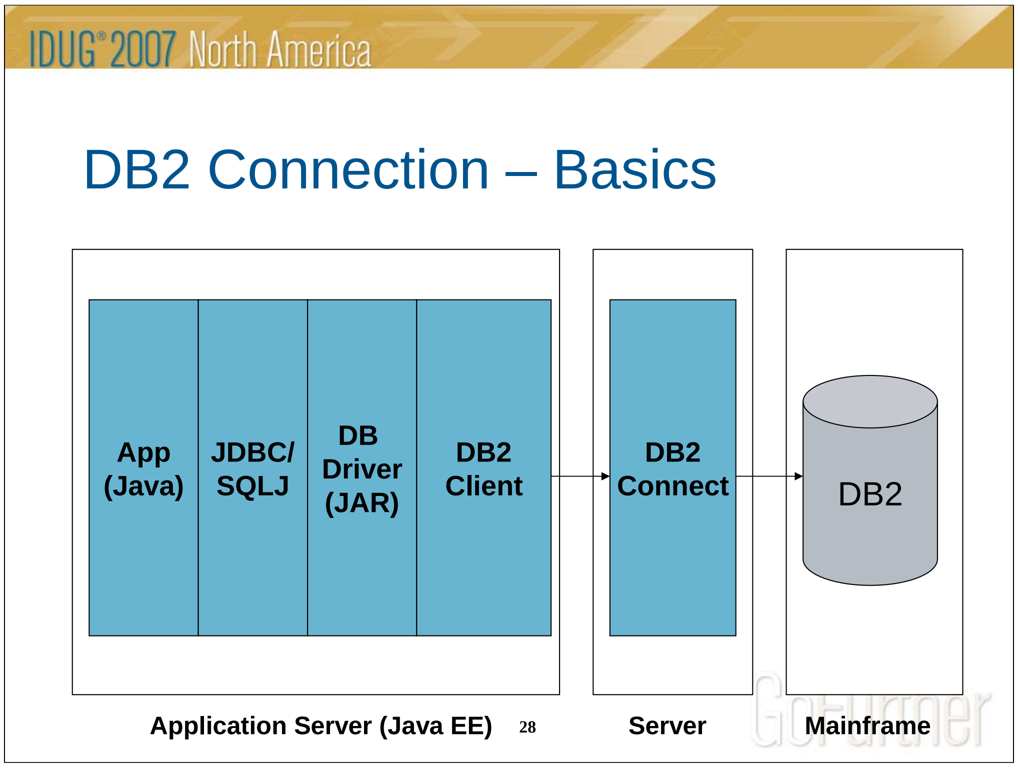
GoFurther

DB2 Connection – Basics

The JDBC API is a set of Java base classes (included in Java SE) which abstract the interaction with a relational database such as DB2 or Oracle. The Connection class represents a Java application's connection to the database and provides an interface for authenticating to it. Typically, in an application server, the specific application server provides a mechanism for obtaining a Connection through a DataSource which is part of the Java EE. The DataSource is configured in the application server and is a mechanism for obtaining Connections.

Beyond the abstraction or generic interface to the database is a specific implementation of the interface by the database vendor. In other words, the vendor builds Java classes specific to their database which make up what's known as a database driver. The Connection and DataSource are known as *Interfaces* which need to be implemented and can't function without that implementation (i.e. the database vendor classes).

Finally, the SQL is sent to the database and returned data handled through one of two API's (sets of Java classes) for either dynamic or static SQL.


IDUG® 2007 North America

DB2 Connection – Basics

28

DB2 Connection – Basics

Thie picture depicts the different levels of Java classes and infrastructure between a Java application and DB2.

App – the business application. This includes code to obtain a DataSource and a Connection through it.

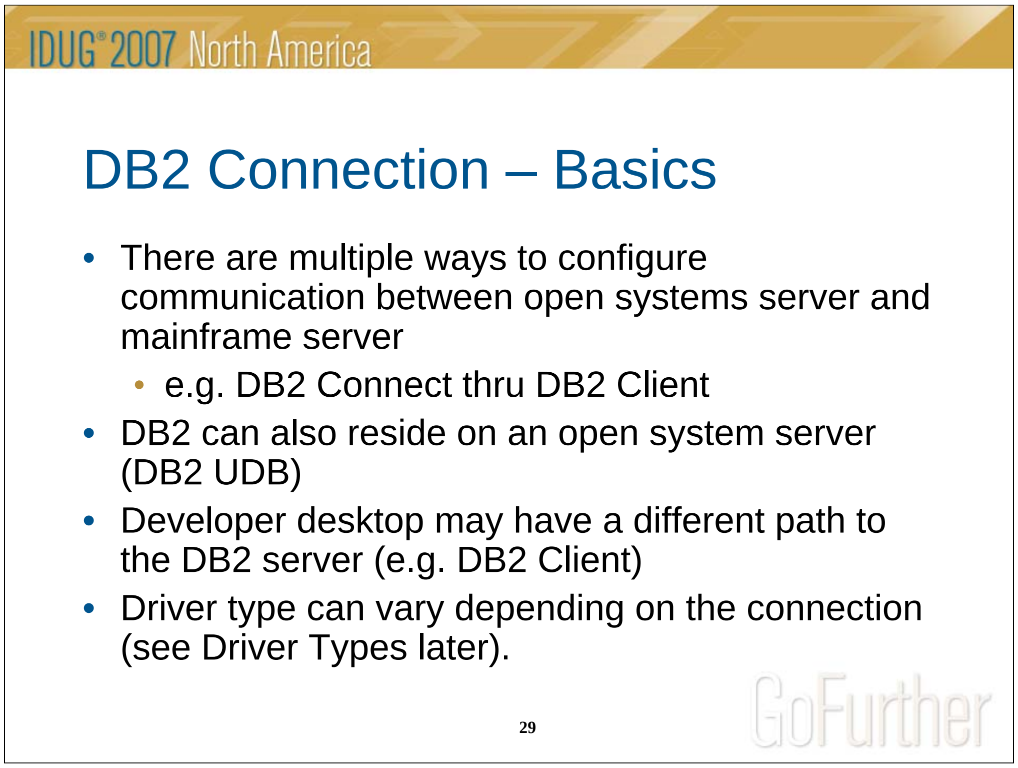
JDBC/SQLJ – the Java API classes for handling dynamic or static SQL

DB Driver – this is the database-specific Java code to interact with the database

DB2 Client – in this example, a way to communicate between the application server and DB2 Connect on another server

DB2 Connect – middleware server to connect the application server to the database server

DB2 – in this example, DB2 running in a mainframe environment with remote threads to handle requests from the application server through the DB2 Connect server



IDUG® 2007 North America

DB2 Connection – Basics

- There are multiple ways to configure communication between open systems server and mainframe server
  - e.g. DB2 Connect thru DB2 Client
- DB2 can also reside on an open system server (DB2 UDB)
- Developer desktop may have a different path to the DB2 server (e.g. DB2 Client)
- Driver type can vary depending on the connection (see Driver Types later).

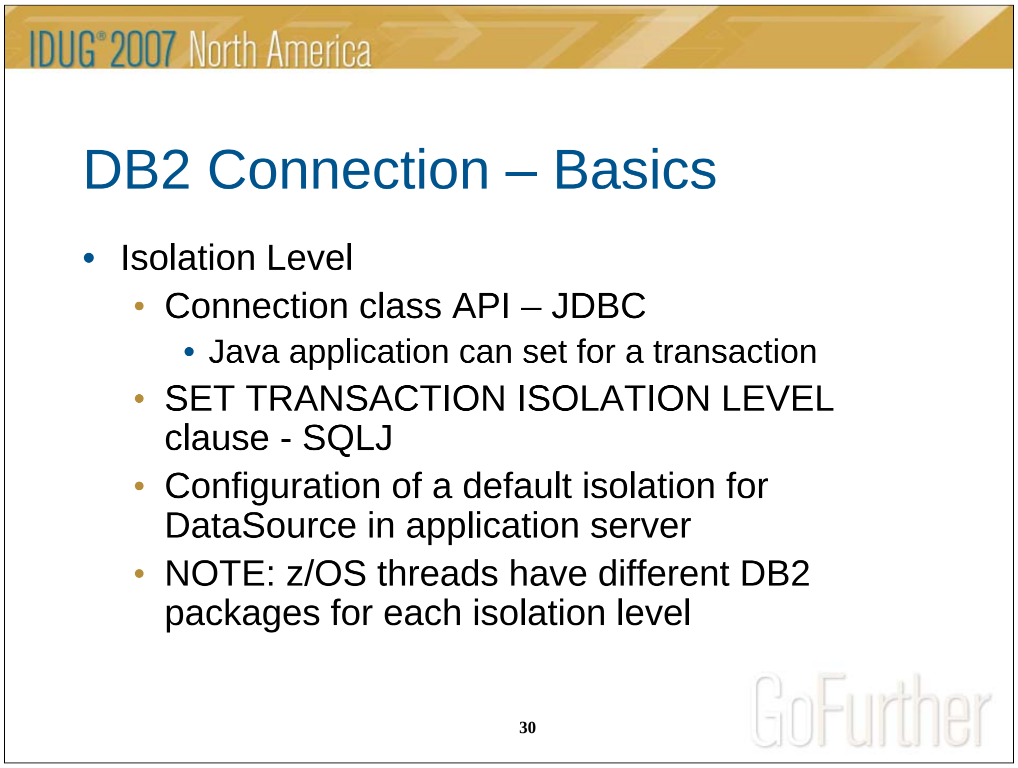
29

GoFurther

DB2 Connection – Basics

There are different ways for an application server (where the Java code runs) to communicate with the database server. One method for an application server on an open systems platform to communicate with DB2 on a mainframe is through a connection through DB2 Connect. A DB2 Client can provide connectivity from the application server to DB2 Connect and finally to DB2 on the mainframe (middleware option). DB2 also has an all-Java database driver which can connect directly to DB2 on the mainframe.

If DB2 is already on an open systems platform, the connection may be direct through the all-Java driver. Developers on their desktops (in Windows) would probably have a DB2 Client that communicates to the mainframe through DB2 Connect.



IDUG® 2007 North America

DB2 Connection – Basics

- Isolation Level
  - Connection class API – JDBC
    - Java application can set for a transaction
  - SET TRANSACTION ISOLATION LEVEL clause - SQLJ
  - Configuration of a default isolation for DataSource in application server
  - NOTE: z/OS threads have different DB2 packages for each isolation level

30

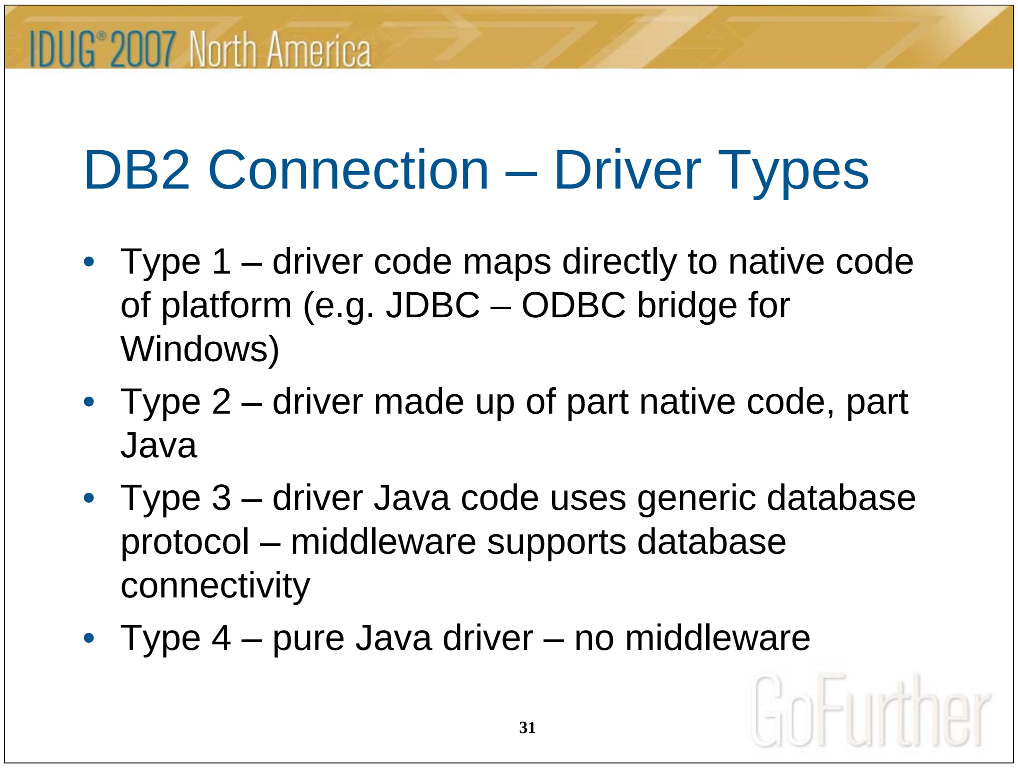
GoFurther

DB2 Connection – Basics

In mainframe applications, the isolation level is generally controlled at development time through the DB2 binds. However, in Java, since the SQL is dynamic, the isolation level is controlled at execution time. The Connection class has a method which can be used to set the level from within the Java code when using the JDBC API for dynamic SQL. When using the SQLJ API for static SQL, there is a SQLJ clause which controls the level.

A default isolation level can be set for all Connections generated through a DataSource defined in the application server.

On z/OS, there are separate packages for each isolation level. Each package then handles the remote Java threads to DB2 that have specified its associated isolation level.


IDUG® 2007 North America

DB2 Connection – Driver Types

- Type 1 – driver code maps directly to native code of platform (e.g. JDBC – ODBC bridge for Windows)
- Type 2 – driver made up of part native code, part Java
- Type 3 – driver Java code uses generic database protocol – middleware supports database connectivity
- Type 4 – pure Java driver – no middleware

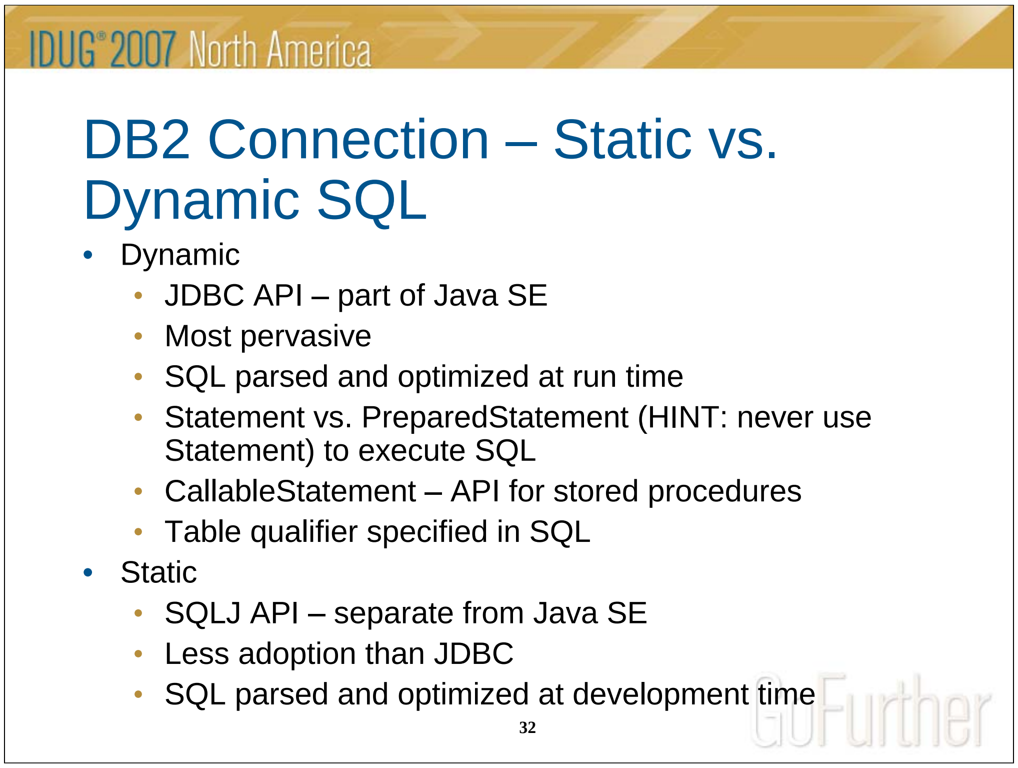
31

GoFurther

DB2 Connection – Driver Types

The database driver is a Java class(es) that implements the generic Java API for connectivity from a Java application to the specific relational database. There are 4 levels of driver types to support different means of providing the connectivity. Type 1 is the least portable as it depends more on specific operating system code to work. Type 4 is the most portable in that the driver is written entirely in Java and requires no middleware to work. Levels 2 and 3 are somewhere in the middle.

NOTE: DB2 does provide an all-Java or Type 4 driver which can run on multiple platforms and which requires no middleware.


IDUG® 2007 North America

DB2 Connection – Static vs. Dynamic SQL

- Dynamic
  - JDBC API – part of Java SE
  - Most pervasive
  - SQL parsed and optimized at run time
  - Statement vs. PreparedStatement (HINT: never use Statement) to execute SQL
  - CallableStatement – API for stored procedures
  - Table qualifier specified in SQL
- Static
  - SQLJ API – separate from Java SE
  - Less adoption than JDBC
  - SQL parsed and optimized at development time

32

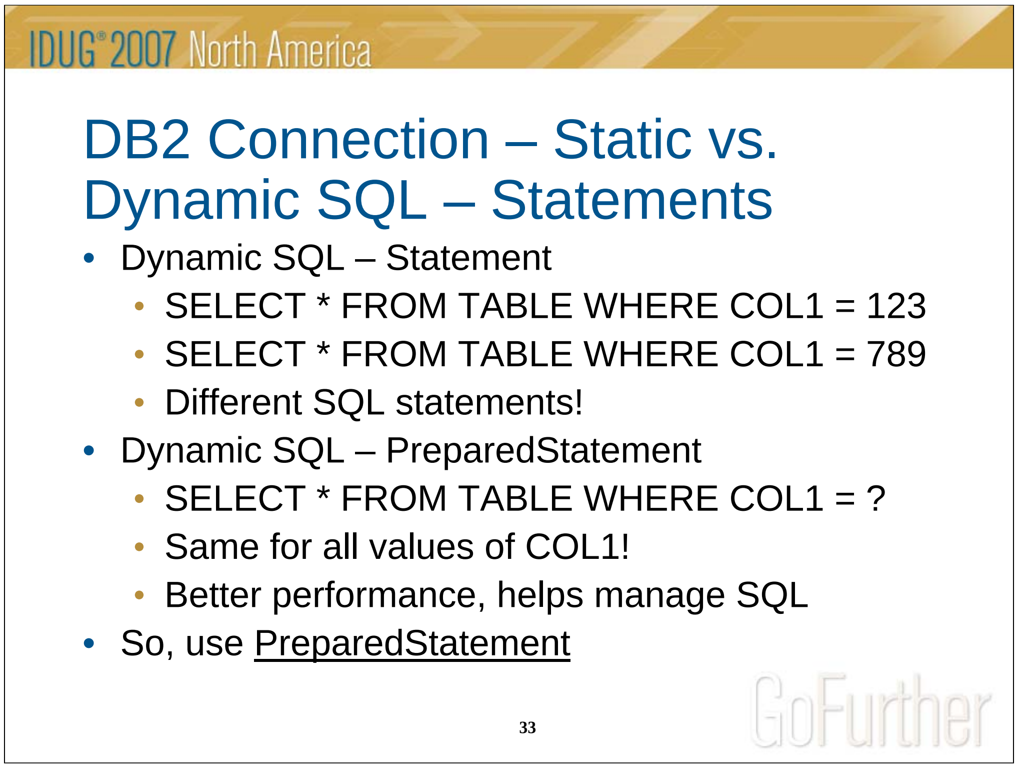
GoFurther

DB2 Connection – Static vs. Dynamic SQL

Most mainframe/COBOL programmers are familiar with static SQL. In other words, the SQL is determined at development time, and a package is bound to DB2. At that time, the SQL is parsed and optimized and the SQL cannot change at execution time. Dynamic SQL on the other hand isn't parsed or optimized until run time. Also, the SQL statement might not be fully known until run time since the Java application can dynamically build it through code (i.e. SQL string is strung together and can vary).

Java has two primary API's (sets of classes) which provide the ability to send SQL to DB2. JDBC is the API which supports dynamic SQL and is the most pervasive among Java developers. The Java class which abstracts the SQL statement is the PreparedStatement class (don't ever use the Statement class). For calling stored procedures, the CallableStatement class from the JDBC API is used.

SQLJ is a standard API for building static SQL (including DB2 binds at development time). It's not used as widely as JDBC and isn't part of the base JDK (the API is an ISO standard, not a Java standard). However, it does provide for the use of static SQL which may perform better at run time.


IDUG® 2007 North America

DB2 Connection – Static vs. Dynamic SQL – Statements

- Dynamic SQL – Statement
  - SELECT * FROM TABLE WHERE COL1 = 123
  - SELECT * FROM TABLE WHERE COL1 = 789
  - Different SQL statements!
- Dynamic SQL – PreparedStatement
  - SELECT * FROM TABLE WHERE COL1 = ?
  - Same for all values of COL1!
  - Better performance, helps manage SQL
- So, use PreparedStatement

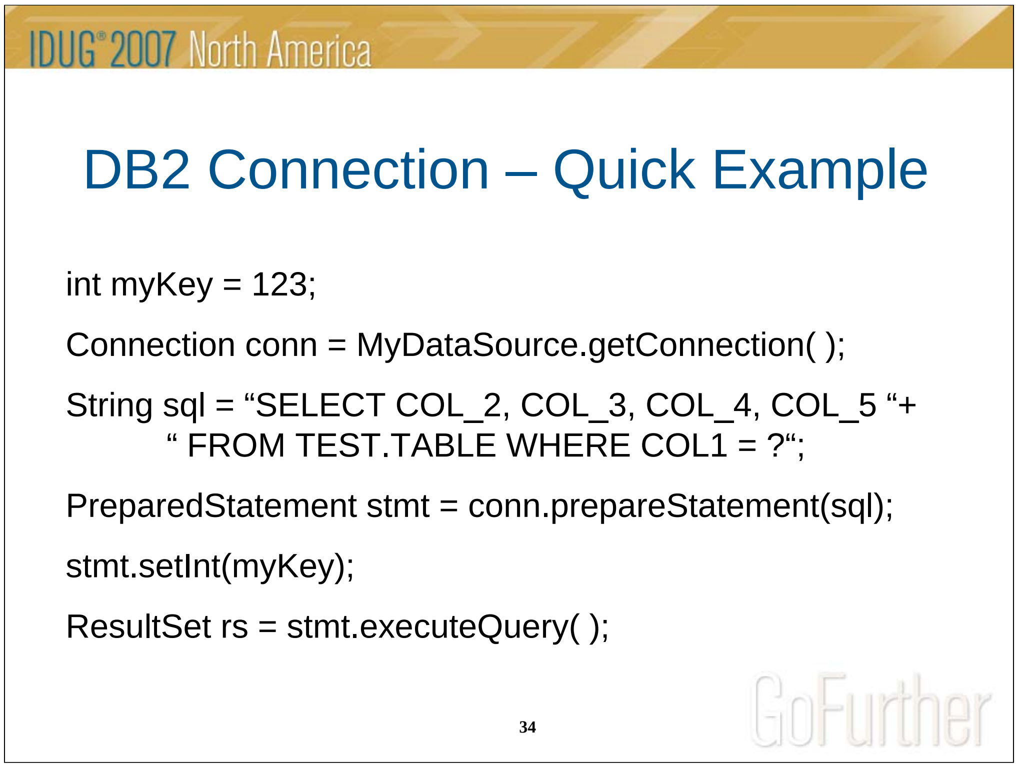
33

GoFurther

DB2 Connection – Static vs. Dynamic SQL – Statements

The Statement Java class (part of the JDBC API) is used to build very dynamic SQL statements in that any variables are actually literals within the SQL. The application dynamically builds the SQL string and inserts the value of any variable. This results in a separate SQL statement for each combination of variable values and limits the statement caching that DB2 can do to improve performance.

The PreparedStatement Java class uses parameter markers (question marks) to represent the placement of host variables. The Java application then binds values to the markers at run time which greatly improves statement caching in DB2. As an added benefit, this better enables the managing of SQL statements such as through tools that capture the SQL since there are fewer different statements.



IDUG® 2007 North America

DB2 Connection – Quick Example

```
int myKey = 123;
Connection conn = MyDataSource.getConnection( );
String sql = “SELECT COL_2, COL_3, COL_4, COL_5 “+
        “ FROM TEST.TABLE WHERE COL1 = ?“;
PreparedStatement stmt = conn.prepareStatement(sql);
stmt.setInt(myKey);
ResultSet rs = stmt.executeQuery( );
```

34

GoFurther

DB2 Connection – Quick Example

This example shows a code snippet in Java to get a Connection to DB2 and issue an SQL statement:

•set up a key value for the database

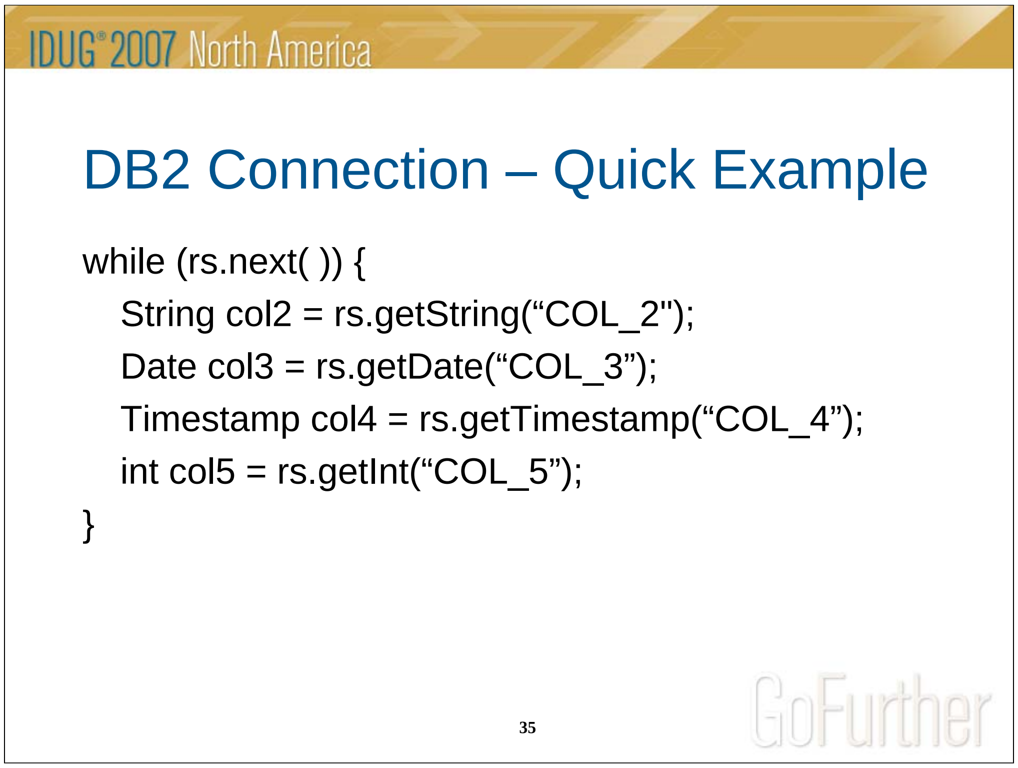
•invoke a method from another Java class which obtains a Connection to the database

•dynamically builds an SQL string (uses a ? as a parameter marker for the host variable)

•prepares the SQL before execution

•binds a Java variable to the parameter marker in the SQL string

•executes the query and returns a ResultSet which points to a returned DB2 cursor


IDUG® 2007 North America

DB2 Connection – Quick Example

```
while (rs.next( )) {
   String col2 = rs.getString(“COL_2");
   Date col3 = rs.getDate(“COL_3”);
   Timestamp col4 = rs.getTimestamp(“COL_4”);
   int col5 = rs.getInt(“COL_5”);
}
```

35

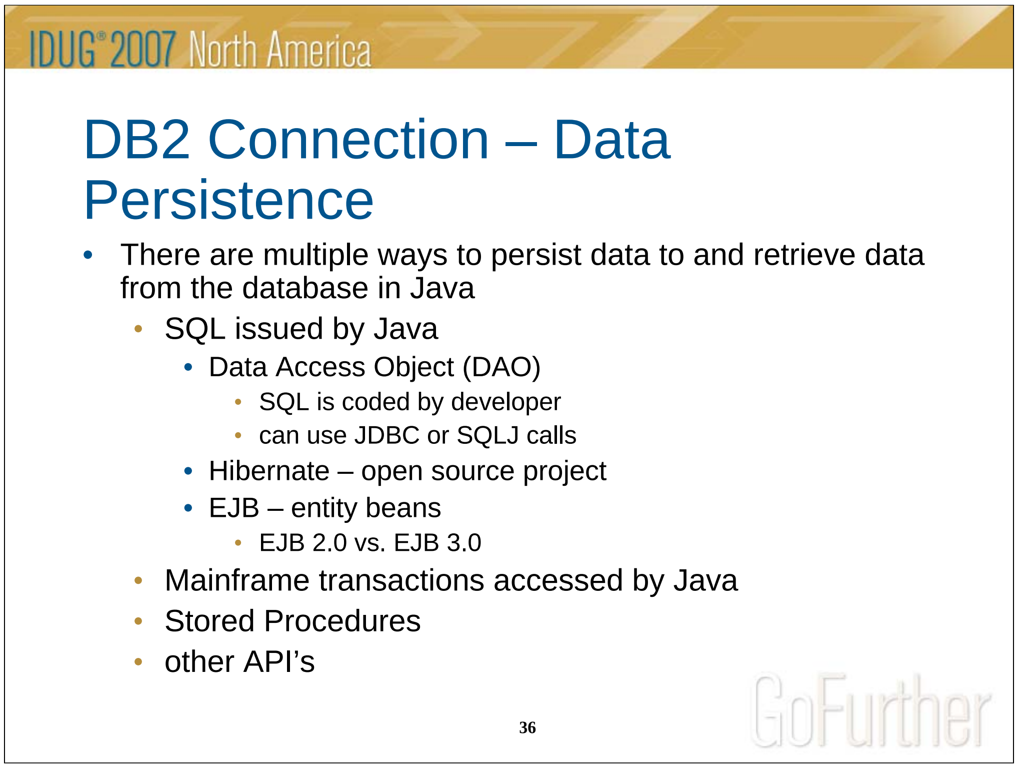
GoFurther

DB2 Connection – Quick Example

This slide completes the example from the previous page. After the ResultSet was obtained, the following steps occur:

•loop is executed for every row in the result set (fetches to the DB2 cursor) using the next( ) method

•column values are obtained from the row using the appropriate get( ) method (based on column type)



IDUG® 2007 North America

DB2 Connection – Data Persistence

- There are multiple ways to persist data to and retrieve data from the database in Java
  - SQL issued by Java
    - Data Access Object (DAO)
      - SQL is coded by developer
      - can use JDBC or SQLJ calls
    - Hibernate – open source project
    - EJB – entity beans
      - EJB 2.0 vs. EJB 3.0
  - Mainframe transactions accessed by Java
  - Stored Procedures
  - other API's

36

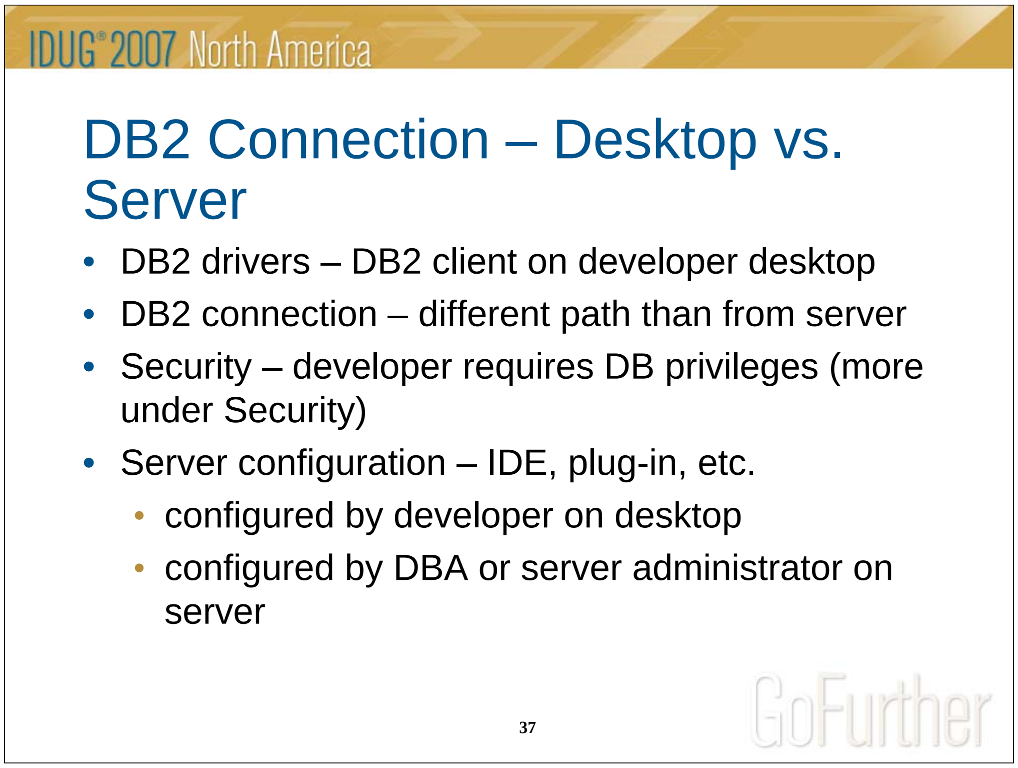
GoFurther

DB2 Connection – Data Persistence

There are choices for application design to store data to DB2 (or other relational database) from a Java application.

SQL can be issued directly from the Java application using various methods such as JDBC, SQLJ, Hibernate, and EJB entity beans. Developers have varying degrees of influence over the generated SQL when using EJB's and Hibernate.

Additionally, the SQL can be issued from the database server, including the mainframe for DB2 on z/OS. In this case, mainframe transactions or stored procedures may execute on the database server and return data to the invoking Java application. Details on these various approaches are beyond the scope of this presentation.



IDUG® 2007 North America

DB2 Connection – Desktop vs. Server

- DB2 drivers – DB2 client on developer desktop
- DB2 connection – different path than from server
- Security – developer requires DB privileges (more under Security)
- Server configuration – IDE, plug-in, etc.
  - configured by developer on desktop
  - configured by DBA or server administrator on server

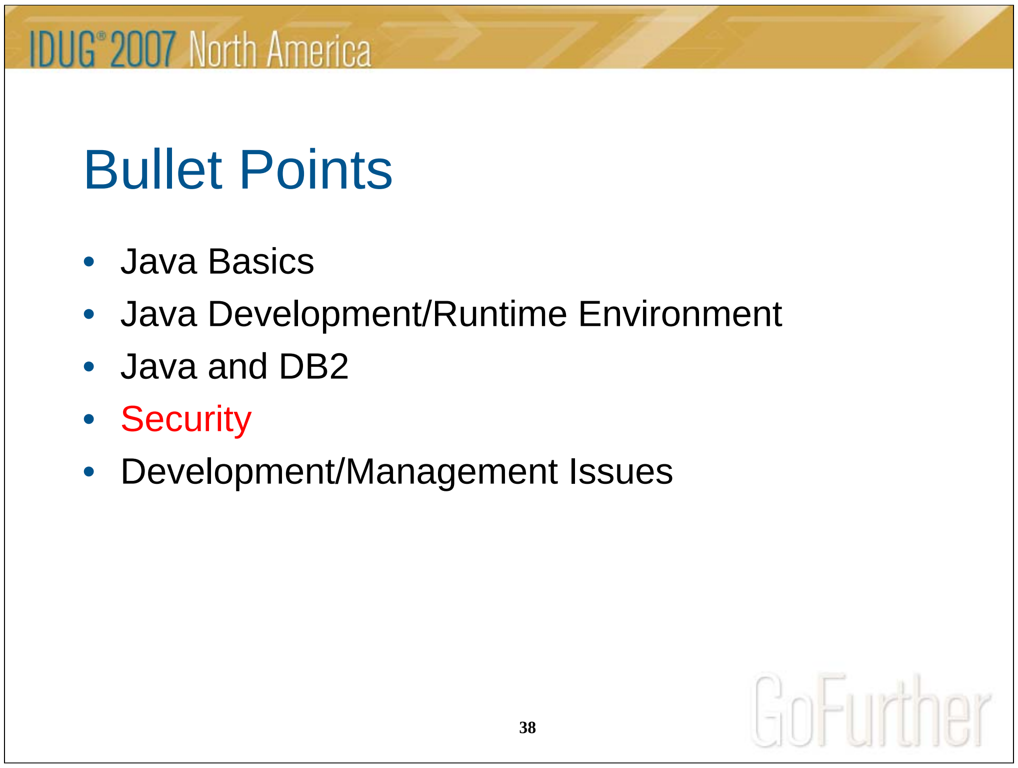
37

GoFurther

DB2 Connection – Desktop vs. Server

It's helpful for developers and DBA's to realize that the path to DB2 will usually vary between a developer's desktop and the application server. Also, if the developer's desktop is on a different operating system from the application server (again, this is likely), the software will vary. However, if a type 4 driver is used, the Java code of the driver should be the same. Of course, the closer the developer's desktop testing environment is to that of the servers, the less issues should arrive when moving from unit testing (developer desktop) to quality assurance testing and production (application server). Any configuration of the connection (e.g. DataSource) will need to be done by the developer on the desktop vs. the administrator of the server in the application server environment.

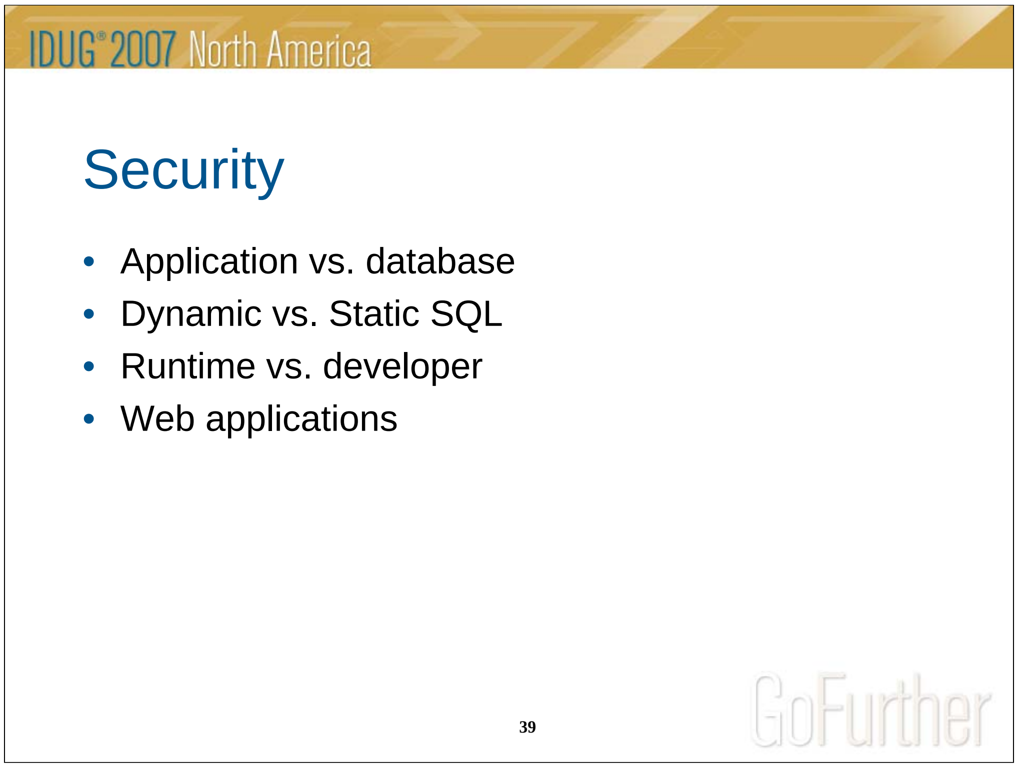
Security is another configuration which can change from developer desktop to server. Generally, the application server will authenticate to the database with a single application id for all users (see more under Security). However, the developer will probably need to use his or her own user id when authenticating to the database during desktop testing.



Bullet Points

- Java Basics
- Java Development/Runtime Environment
- Java and DB2
- Security
- Development/Management Issues

GoFurther


IDUG® 2007 North America

Security

- Application vs. database
- Dynamic vs. Static SQL
- Runtime vs. developer
- Web applications

39

GoFurther

Security

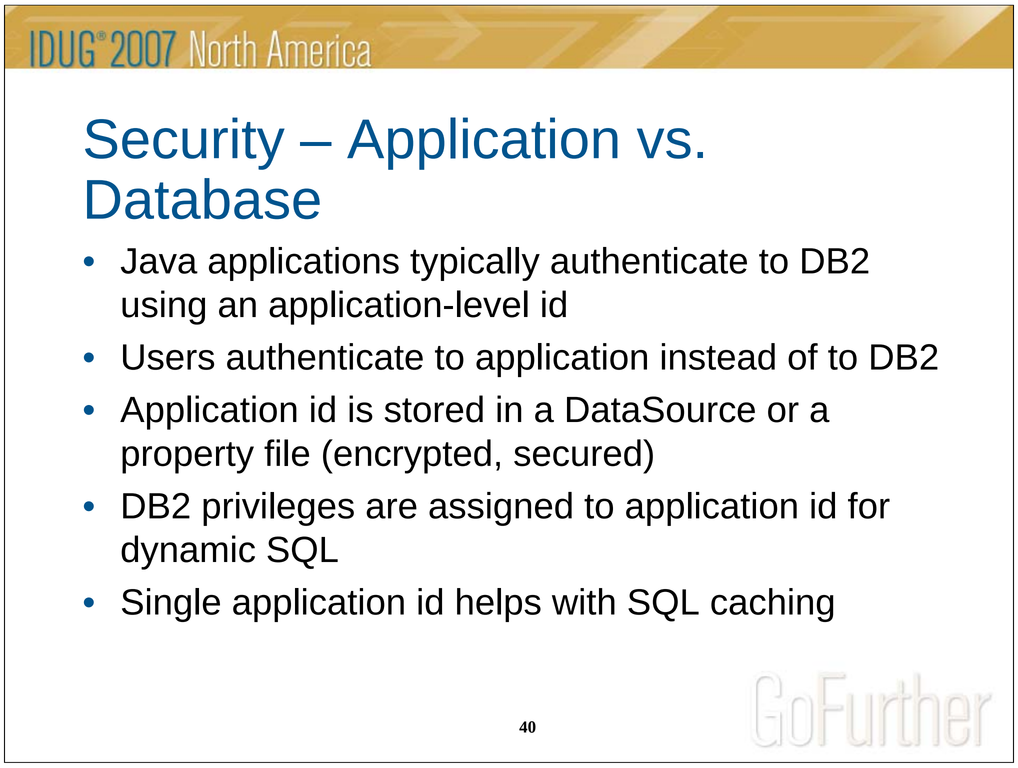
This section will address the the following security aspects of a Java application:

•Application vs. database – authenticating to an application as compared to the application authenticating to the database

•The differences in database authentication for applications using dynamic vs. static SQL

•The differences in security configuration for developers doing unit testing vs. that of the deployed application on a test or production application server

•The levels of security for a web application


IDUG® 2007 North America

Security – Application vs. Database

- Java applications typically authenticate to DB2 using an application-level id
- Users authenticate to application instead of to DB2
- Application id is stored in a DataSource or a property file (encrypted, secured)
- DB2 privileges are assigned to application id for dynamic SQL
- Single application id helps with SQL caching

40

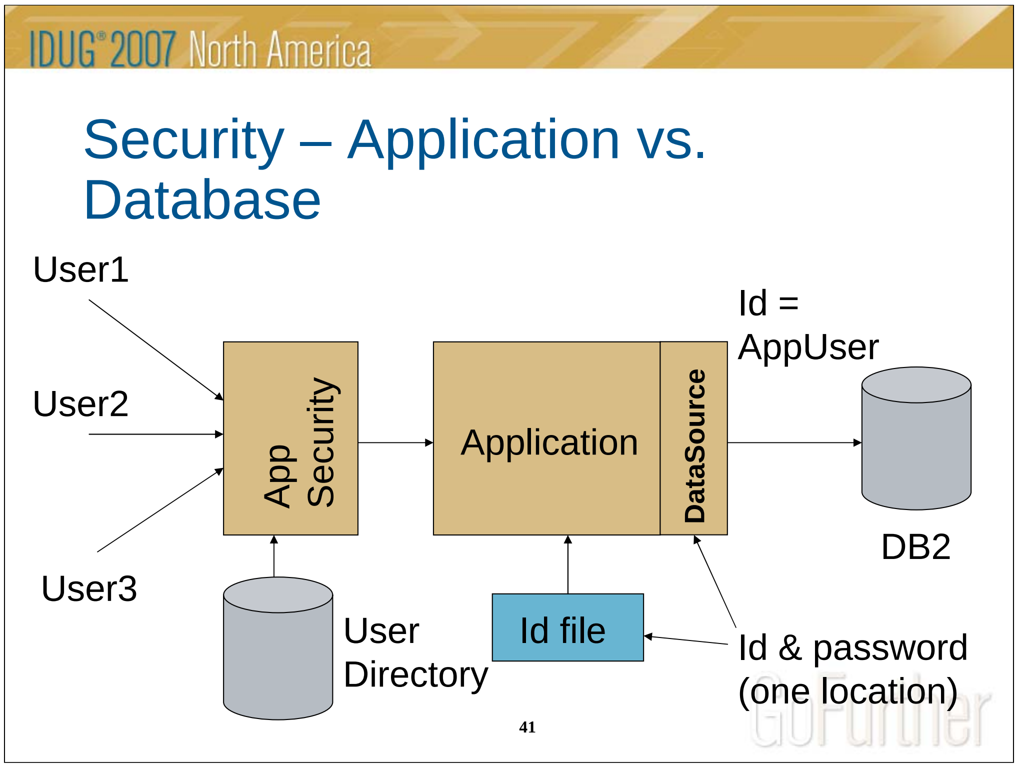
GoFurther

Security – Application vs. Database

Similar to the security paradigm in mainframe transactional applications where users have access to transactions and the transaction has access to the database, Java application users have access to the application and the application has access to the database. This is accomplished typically through an application-level id which has privileges to the database.

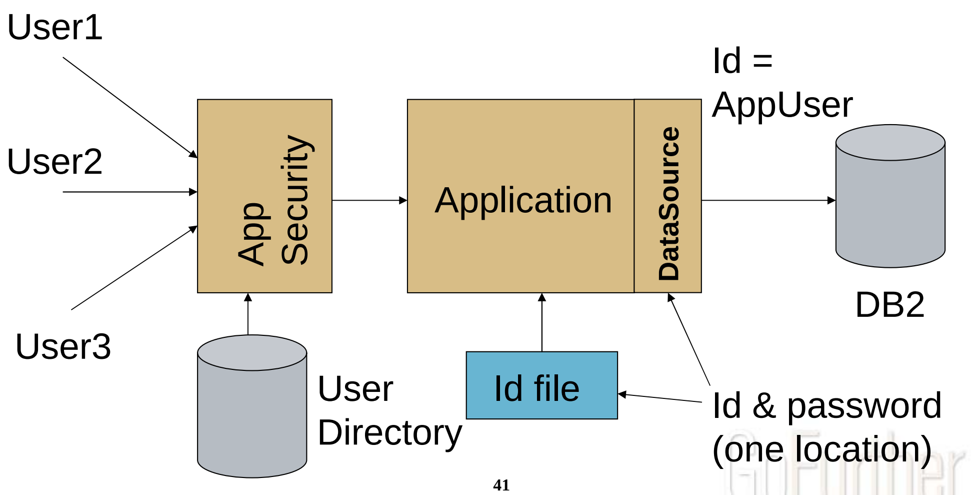
To secure the application id and password, DataSource's can be configured in an application server. The application id and password can be stored here and are generally encrypted with limited access for administration. Alternatively, the id and encrypted password can be stored in a protected server file folder. This approach requires base-level application code to read the file at database connection time.

An added benefit of a single application id is improved caching of SQL statements by DB2. DB2 looks at the contents of the SQL as well as the database user id to identify same statements that can be accessed from cache.



IDUG® 2007 North America

Security – Application vs. Database



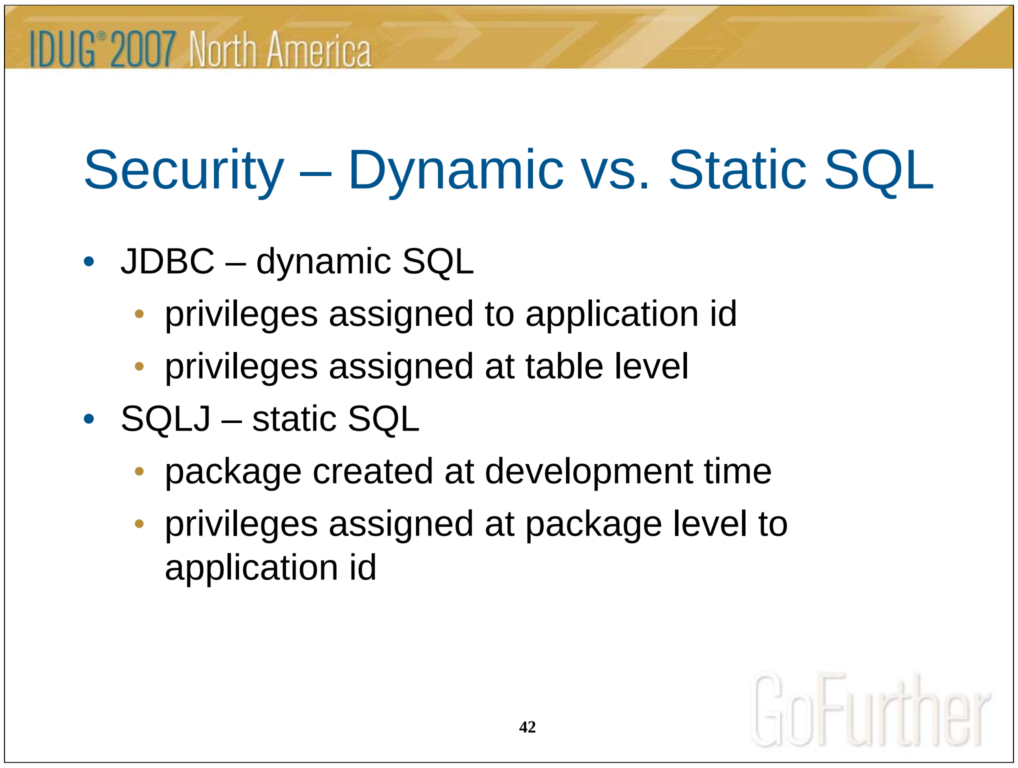
41

GoFurther

Security – Application Authentication

Users are first prompted for access to the application through application-level security. Once past the initial authentication, the user then has access to the application. The application then locates the application id and password from either a protected file or a DataSource configuration. The application id and password are used to connect to the DB2 database through the JDBC API.

NOTE: for table updates, the individual id of User1, User2, and User3 is recorded in user id column for audit/change tracking.


IDUG® 2007 North America

Security – Dynamic vs. Static SQL

- JDBC – dynamic SQL
  - privileges assigned to application id
  - privileges assigned at table level
- SQLJ – static SQL
  - package created at development time
  - privileges assigned at package level to application id

42

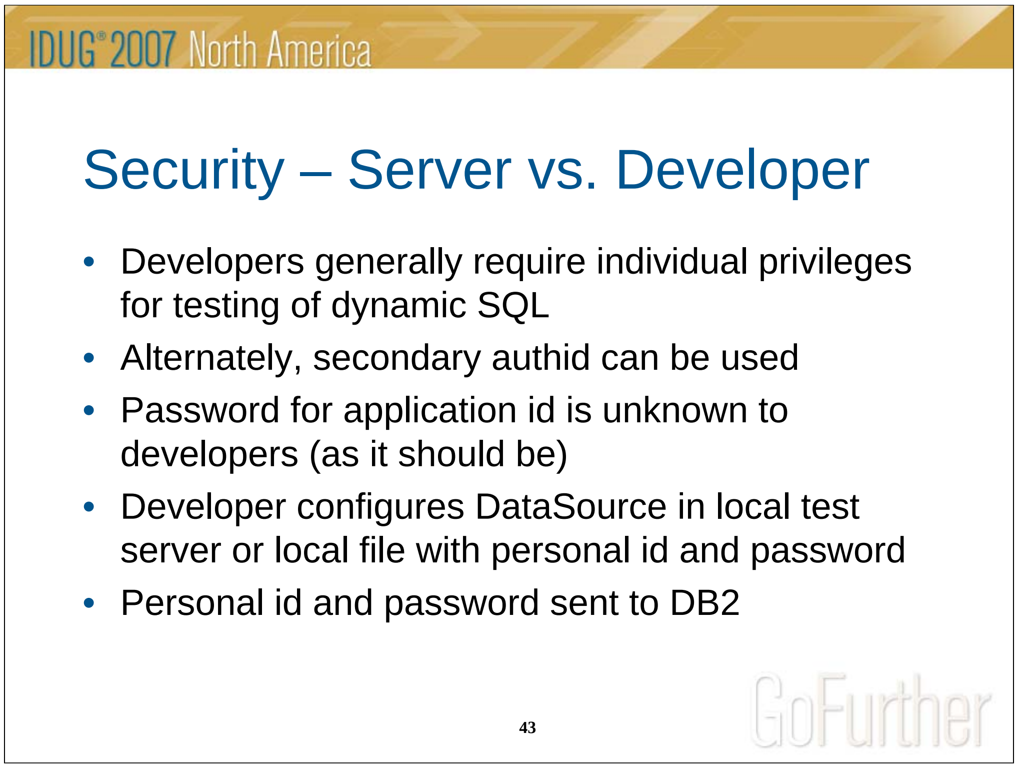
GoFurther

Security – Dynamic vs. Static SQL

Security configuration for applications using dynamic SQL is different from applications using static SQL.

Dynamic SQL (applications using the JDBC API) implies that privileges to individual tables need to be assigned to the id used to connect to DB2. Generally, this is an application level id. Of course, for desktop testing, the developer id's will also need to have access to the tables (this may occur through a secondary authid).

Static SQL (applications using the SQLJ API) use the same security approach as COBOL/mainframe applications. DB2 packages are bound at development time and privileges are assigned at the package level, not the id level.


IDUG® 2007 North America

Security – Server vs. Developer

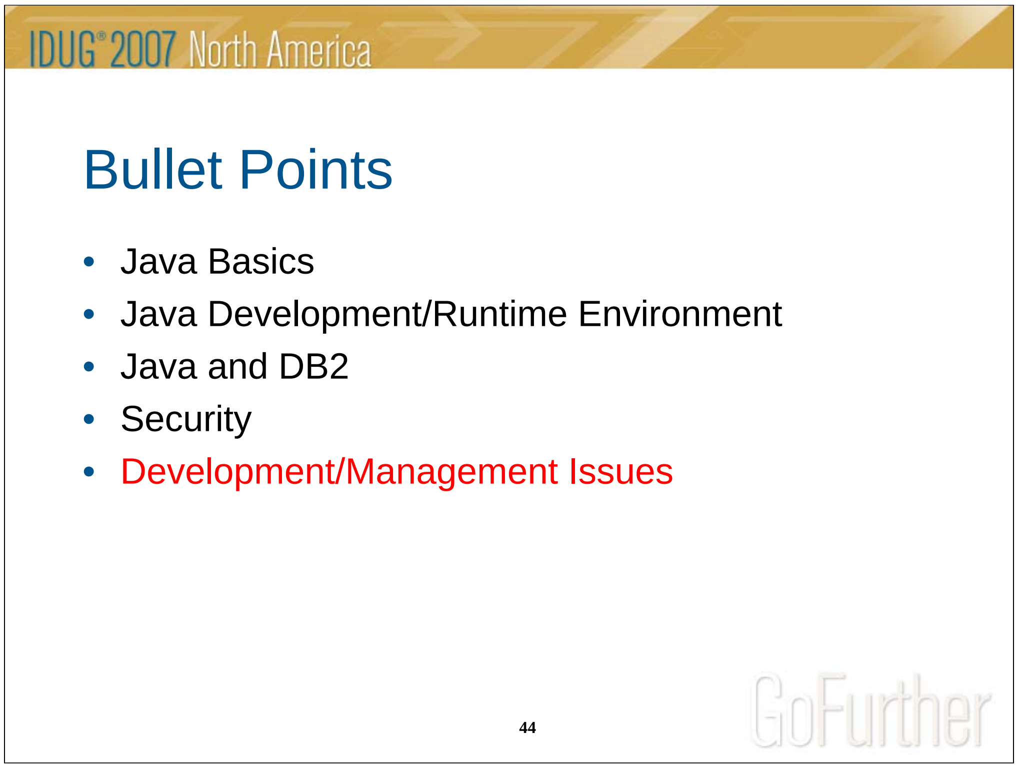
- Developers generally require individual privileges for testing of dynamic SQL
- Alternately, secondary authid can be used
- Password for application id is unknown to developers (as it should be)
- Developer configures DataSource in local test server or local file with personal id and password
- Personal id and password sent to DB2

43

GoFurther

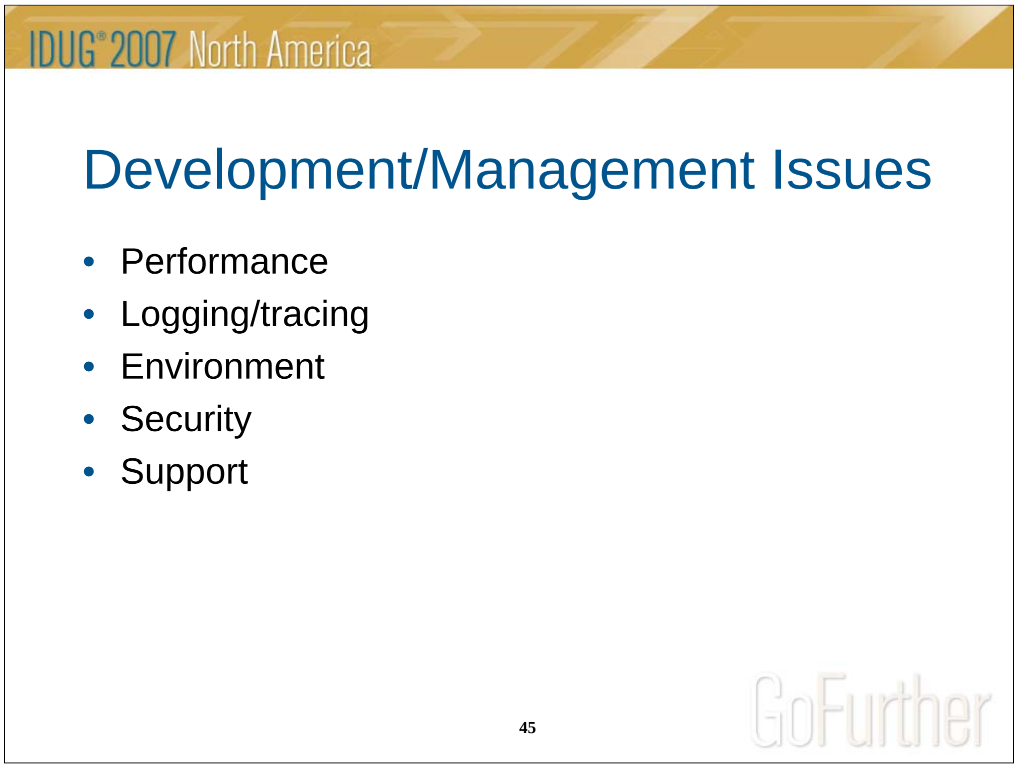
Security – Server vs. Developer

Because developers typically use their own id’s to authenticate to DB2 for desktop testing, they will require privileges to the tables for dynamic SQL or alternatively to a package for static SQL. This can certainly be accomplished by adding the developer id to a secondary authid (the same as the application id). The developer id is used to avoid revealing the application password to the developer.



Bullet Points

- Java Basics
- Java Development/Runtime Environment
- Java and DB2
- Security
- Development/Management Issues


IDUG® 2007 North America

Development/Management Issues

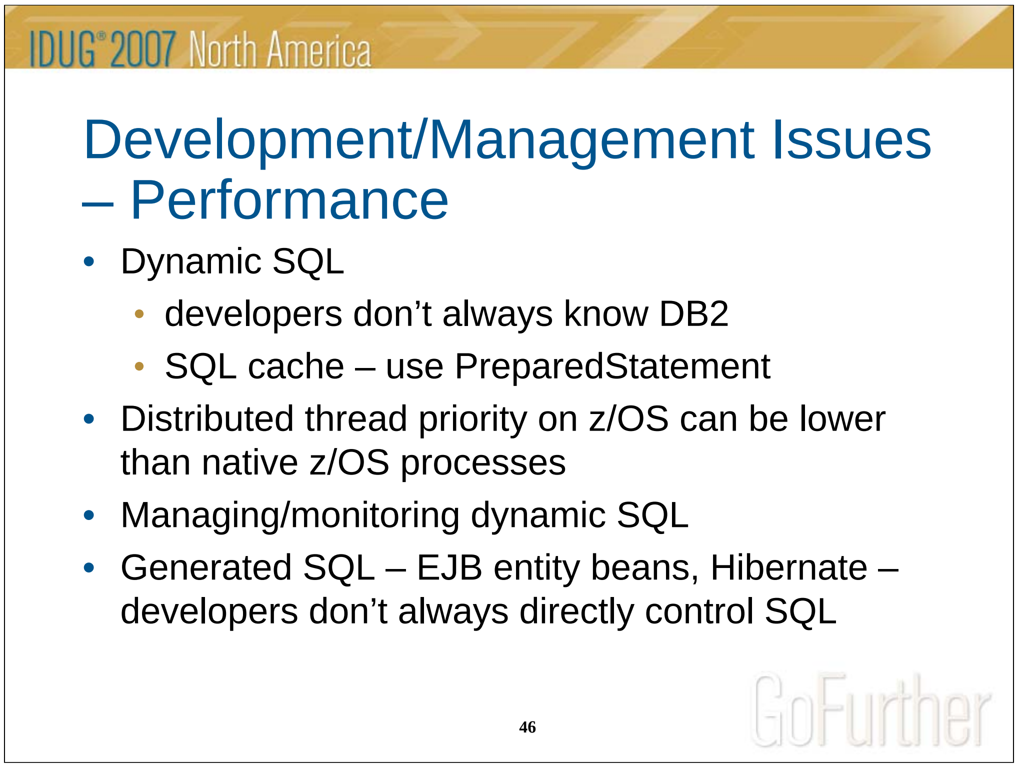
- Performance
- Logging/tracing
- Environment
- Security
- Support

45

GoFurther

Development/Management Issues

This section will look at just some of the issues related to developing and running Java applications since it can be quite different than for mainframe applications.


IDUG® 2007 North America

Development/Management Issues – Performance

- Dynamic SQL
  - developers don't always know DB2
  - SQL cache – use PreparedStatement
- Distributed thread priority on z/OS can be lower than native z/OS processes
- Managing/monitoring dynamic SQL
- Generated SQL – EJB entity beans, Hibernate – developers don't always directly control SQL

46

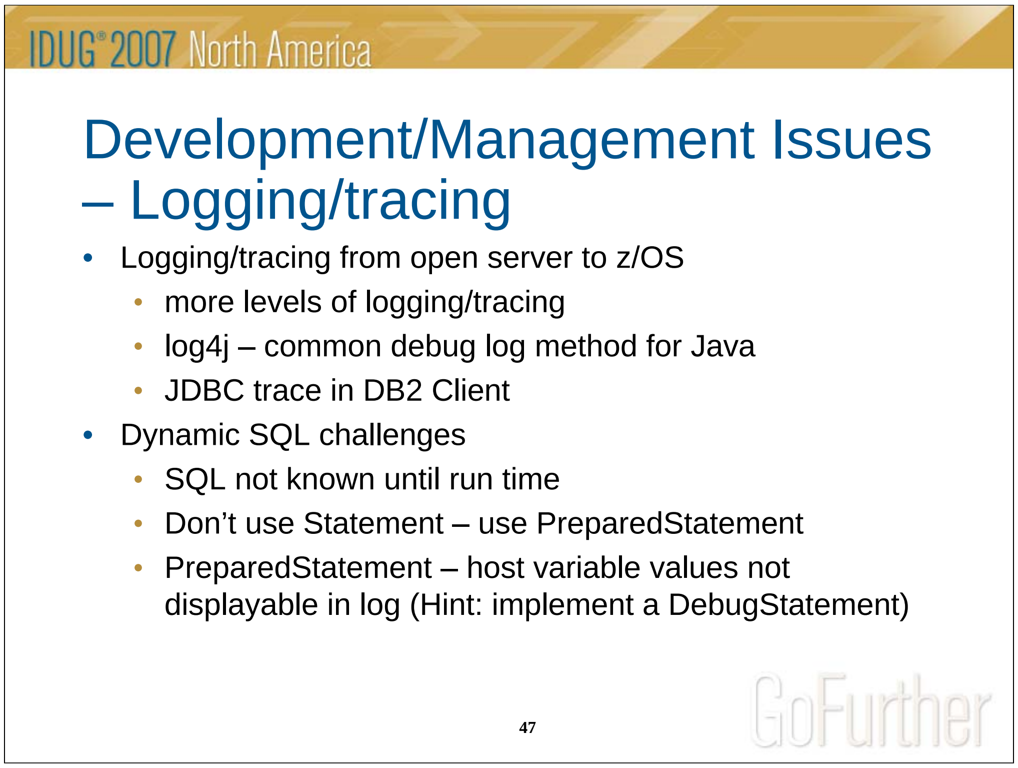
GoFurther

Development/Management Issues – Performance

Dynamic SQL can be a big change for developers and DBA's used to working with static SQL in COBOL programs. However, many developers who have "grown up" with Java aren't familiar with static SQL. In fact, they might not be familiar with the mainframe at all which can be a challenge to DBA's who support mainframe DB2. It's possible that these developers don't recognize the benefits of the PreparedStatement (although most do) for caching SQL statements. The very fact that dynamic SQL is parsed and optimized at run time typically means that it performs worse than static SQL. Managing dynamic SQL is also problematic since the SQL isn't even known until run time. One way to make this easier is to have a tool to organize the statements, but it's still always "after the fact".

Java requests to DB2 on the mainframe are handled by separate threads from other mainframe processes (as remote requests). It's possible or maybe even probable that these remote threads have lower priority on the mainframe than other processes which can slow down their response.

Depending on the option used by the development team for accessing DB2 from Java, the developer may not even explicitly know the SQL. Persistence methods such as EJB entity beans and Hibernate can build the SQL behind the scenes and the performance may not be optimal. Additionally, it may be difficult or impossible to influence the generation of the SQL enough to make it perform well.



IDUG® 2007 North America

Development/Management Issues – Logging/tracing

- Logging/tracing from open server to z/OS
  - more levels of logging/tracing
  - log4j – common debug log method for Java
  - JDBC trace in DB2 Client
- Dynamic SQL challenges
  - SQL not known until run time
  - Don't use Statement – use PreparedStatement
  - PreparedStatement – host variable values not displayable in log (Hint: implement a DebugStatement)

47

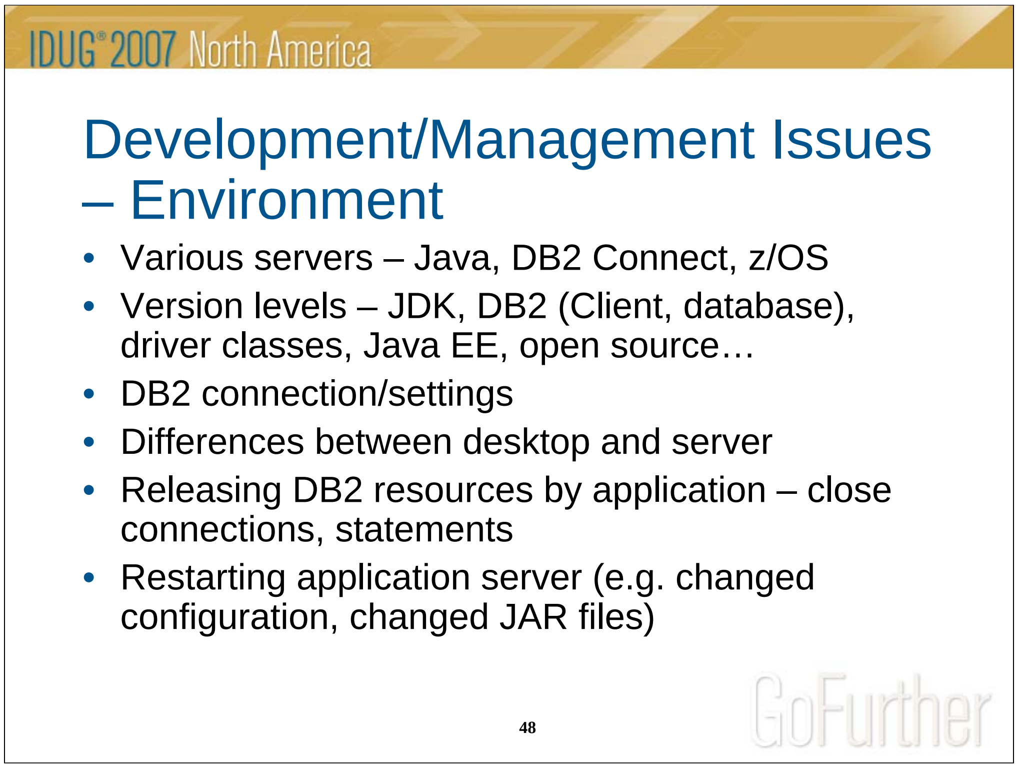
GoFurther

Development/Management Issues – Logging/tracing

Certainly, adding additional servers and environment types to the mix can make it more challenging to track down problems and monitor the health of applications. If DB2 runs on a different server from the Java code, there are two places to look at logging and tracing, and probably two different formats as well.

From the application server side, Java applications typically use some Java API for logging. A popular one is log4j which is an open source API from the Apache Group. Additionally, some tracing for the JDBC API is available through the DB2 Client on the application server.

If you haven't gotten the idea by now, dynamic SQL poses unique challenges with performance and management. Consistent use of the PreparedStatement class (using one application id for all users) can help make the SQL more manageable. Extending the PreparedStatement with debug capability for host variable values can make debugging easier for developers.


IDUG® 2007 North America

Development/Management Issues – Environment

- Various servers – Java, DB2 Connect, z/OS
- Version levels – JDK, DB2 (Client, database), driver classes, Java EE, open source…
- DB2 connection/settings
- Differences between desktop and server
- Releasing DB2 resources by application – close connections, statements
- Restarting application server (e.g. changed configuration, changed JAR files)

48

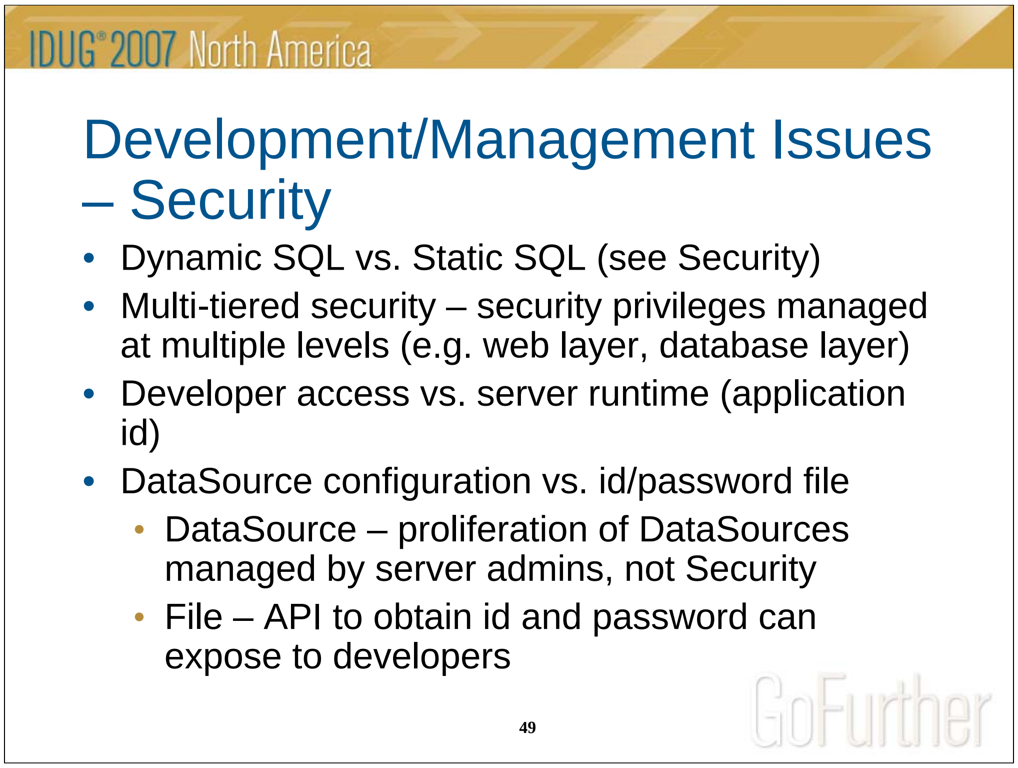
GoFurther

Development/Management Issues – Environment

Overall, there just seem to be more and more moving parts in modern applications, of which, Java is a part. More than likely Java will run on a different server from DB2 and DB2 might have more than one server for connectivity (DB2 Connect). The Java runtime environment adds additional variables for version management for the Java JDK, the version of Java EE (the vendor level for the application server), the version of the DB2 Client which connects to the database, the version of the Java drivers for DB2, possible and likely version of open source software, and more.

Additionally in the environment are configurations for how DB2 connections are managed from the application server and how connections are managed through the Java application code. An additional testing environment and its configuration are introduced for desktop testing and the connectivity of the desktop to DB2.

Availability of applications is also different for Java applications from that of mainframe applications. There are methods for mitigating outages, such as clustered application servers, but recycling of servers is required for many types of application or application configuration changes.



IDUG® 2007 North America

Development/Management Issues – Security

- Dynamic SQL vs. Static SQL (see Security)
- Multi-tiered security – security privileges managed at multiple levels (e.g. web layer, database layer)
- Developer access vs. server runtime (application id)
- DataSource configuration vs. id/password file
  - DataSource – proliferation of DataSources managed by server admins, not Security
  - File – API to obtain id and password can expose to developers

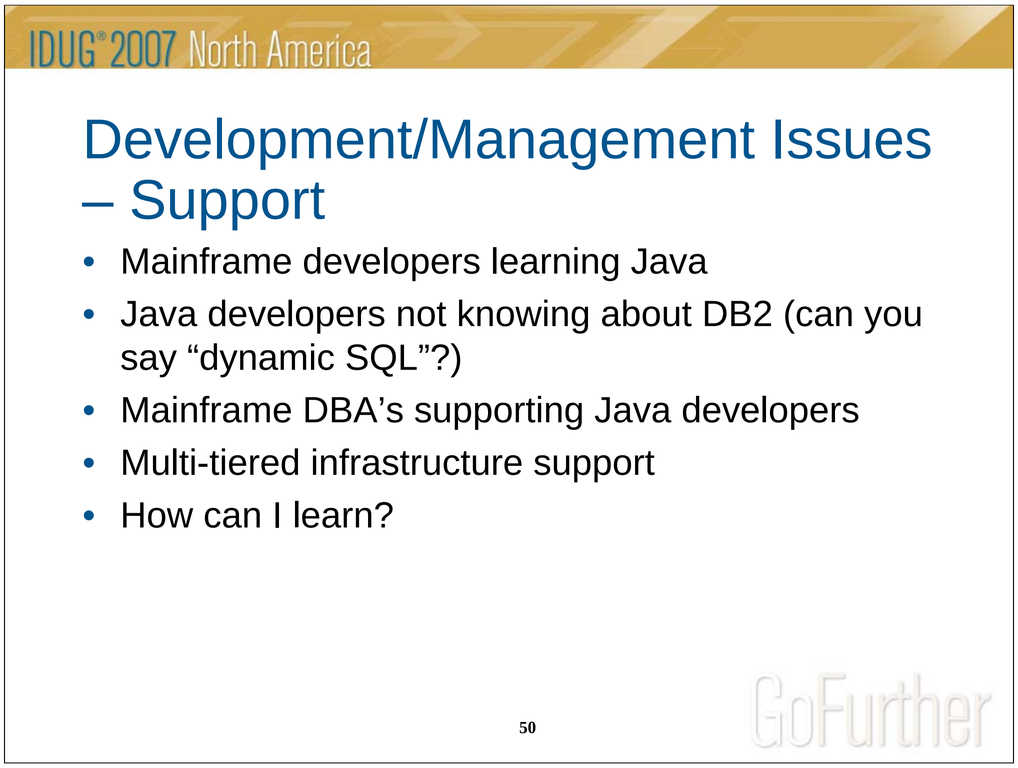
49

GoFurther

Development/Management Issues – Security

Using dynamic SQL may not be a security issue per se, but it's at least different from security for static SQL. As was seen in the Security section, web applications require security to managed for different sets of ids (user accounts vs. the application id) at different levels (authentication to the application and authentication to the database). The maintenance of security is further complicated when developer privileges are required in test to individual DB2 tables in addition to the application id access to the same tables.

As discussed in the Security section, the application id and password can be controlled by the application server administrator in a DataSource. However, there are tradeoffs since it's likely that each application would have a separate id indicating a separate DataSource. This could lead to a proliferation of DataSource configurations and additional administrator overhead. The id and password can be managed in a file, but this choice requires an application API to access the files and presents potential concerns for developers obtaining the credentials.


IDUG® 2007 North America

Development/Management Issues – Support

- Mainframe developers learning Java
- Java developers not knowing about DB2 (can you say “dynamic SQL”?)
- Mainframe DBA’s supporting Java developers
- Multi-tiered infrastructure support
- How can I learn?

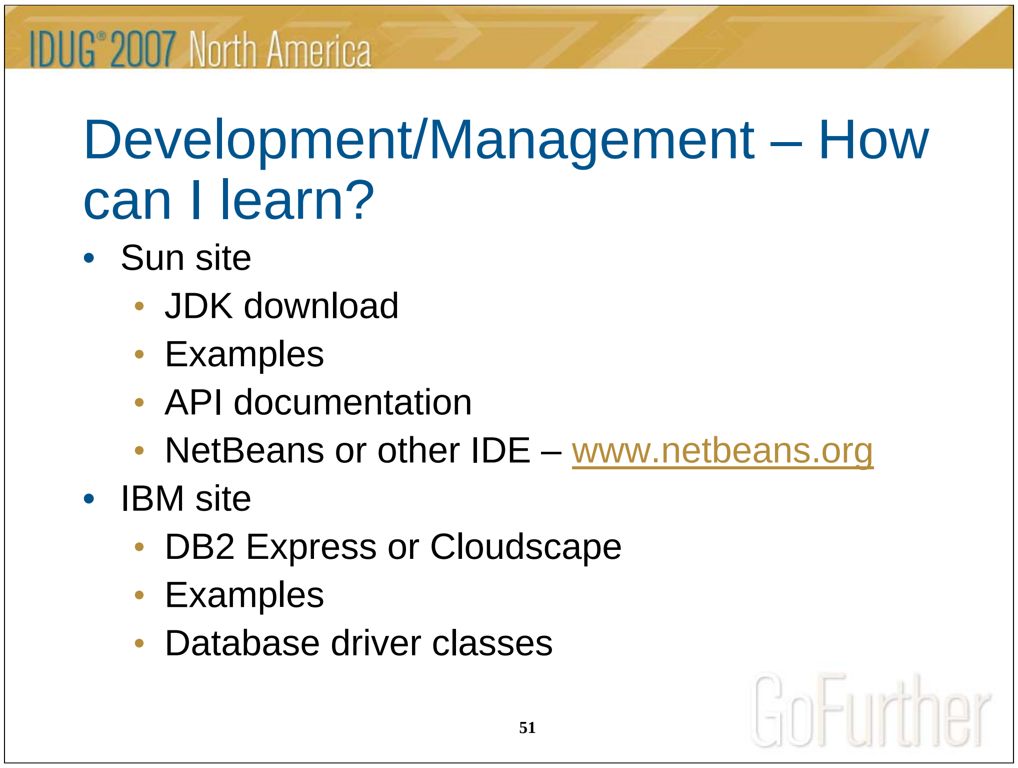
50

GoFurther

Development/Management Issues – Support

It’s obvious that an open systems environment using Java and DB2 can be complex which introduces many issues for support, including the ability for DBA’s and developers to communicate effectively.

One way to mitigate the complexity is to develop a better understanding of the environment and the issues…


IDUG® 2007 North America

Development/Management – How can I learn?

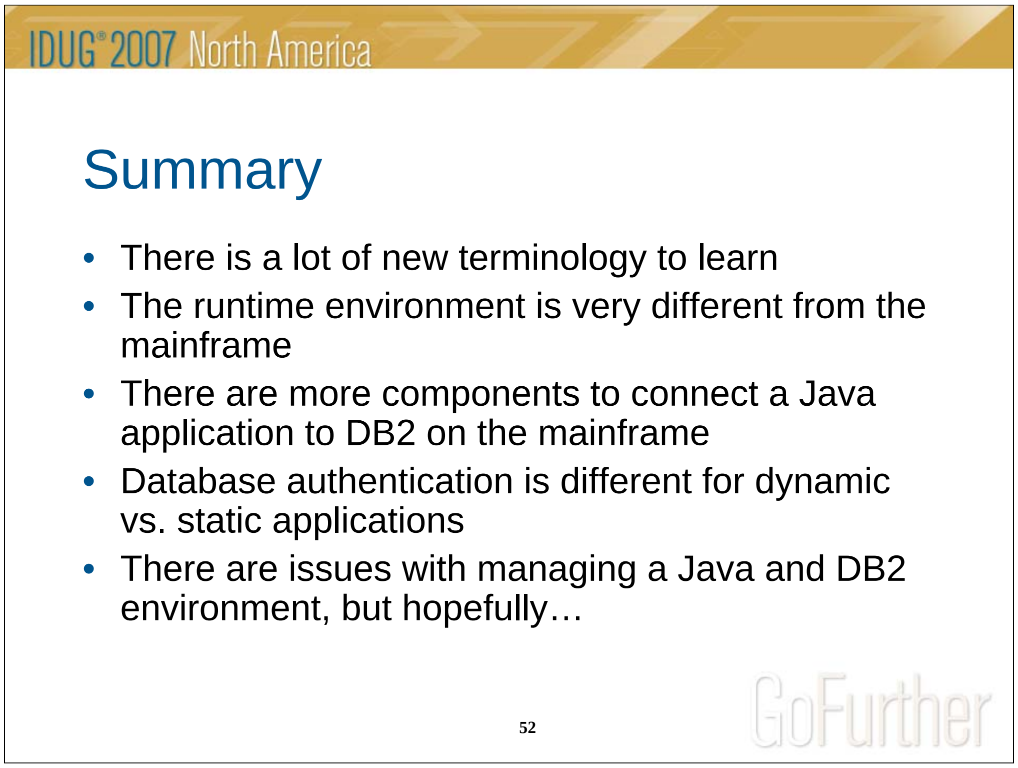
- Sun site
  - JDK download
  - Examples
  - API documentation
  - NetBeans or other IDE – www.netbeans.org
- IBM site
  - DB2 Express or Cloudscape
  - Examples
  - Database driver classes

51

GoFurther

Development/Management Issues – How can I learn?

Hopefully, this presentation has demystified Java somewhat, but there is always opportunity to learn more. For those who are adventurous and want to learn on their own (probably personal time), there is a wealth of support on the web to download Java and DB2 to build small sample applications. Additionally, there are “free” tools available to do so including development tools and even personal databases to tinker with. And, you’re not the first person to do this, so there’s plenty of documentation and forums to provide some of the answers.


IDUG® 2007 North America

Summary

- There is a lot of new terminology to learn
- The runtime environment is very different from the mainframe
- There are more components to connect a Java application to DB2 on the mainframe
- Database authentication is different for dynamic vs. static applications
- There are issues with managing a Java and DB2 environment, but hopefully…

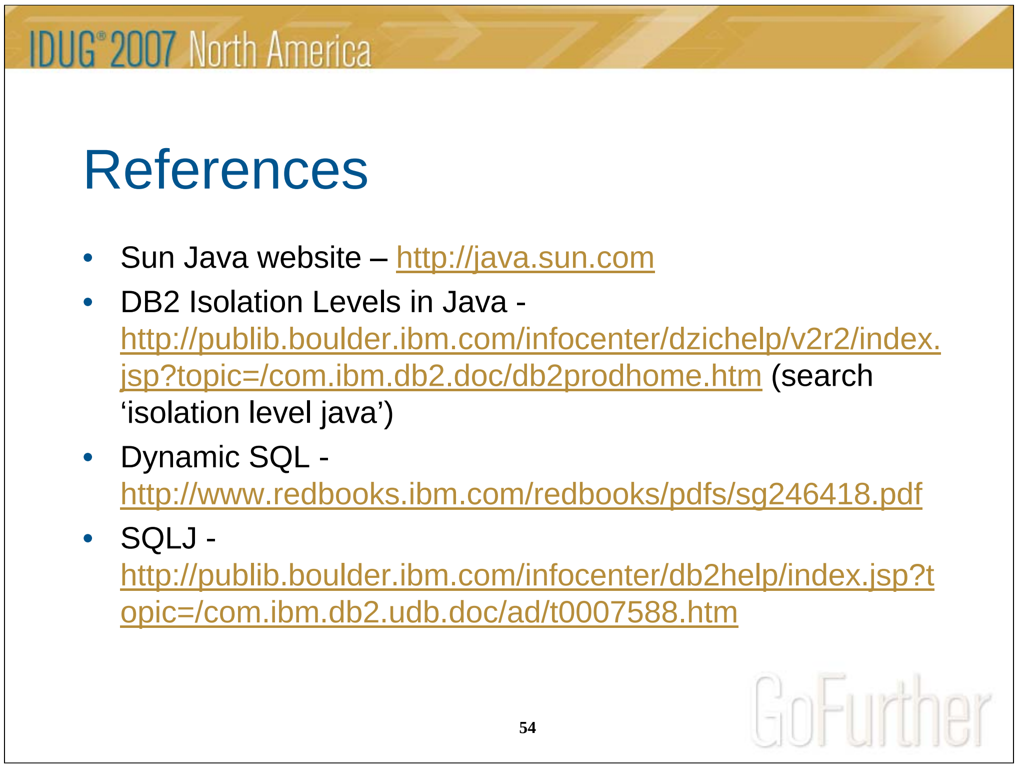
52

GoFurther

Summary

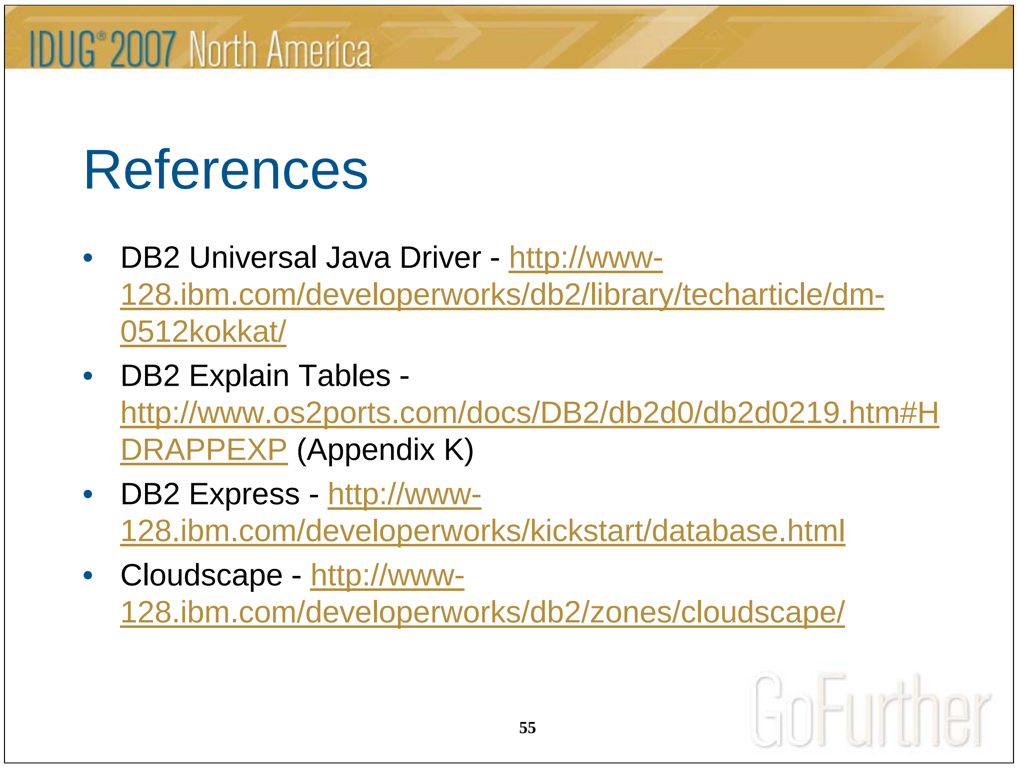
This slide says it all. There are many new concepts which can differ greatly from legacy mainframe applications. This presentation hopefully provided some beginning context to help shed some light on the language, its concepts, its relationship to DB2, and issues related to its use.

You’ve learned something!



References

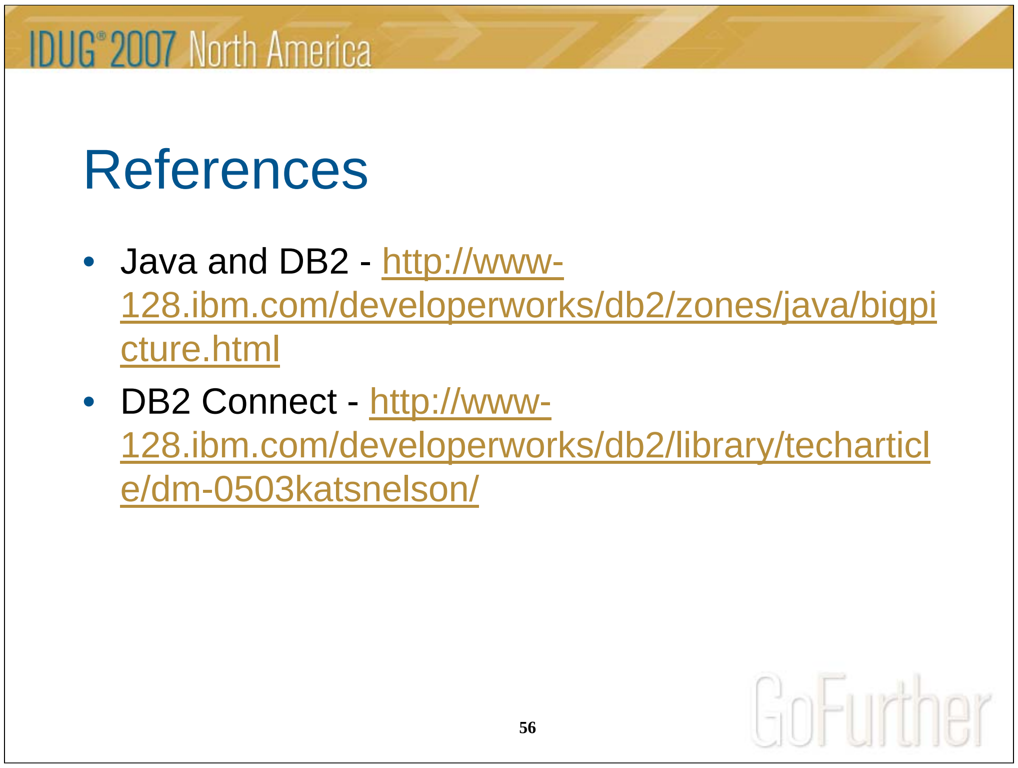
- Sun Java website – http://java.sun.com
- DB2 Isolation Levels in Java - http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=/com.ibm.db2.doc/db2prodhome.htm (search 'isolation level java')
- Dynamic SQL - http://www.redbooks.ibm.com/redbooks/pdfs/sg246418.pdf
- SQLJ - http://publib.boulder.ibm.com/infocenter/db2help/index.jsp?topic=/com.ibm.db2.udb.doc/ad/t0007588.htm



References

- DB2 Universal Java Driver - http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0512kokkat/
- DB2 Explain Tables - http://www.os2ports.com/docs/DB2/db2d0/db2d0219.htm#HDRAPPEXP (Appendix K)
- DB2 Express - http://www-128.ibm.com/developerworks/kickstart/database.html
- Cloudscape - http://www-128.ibm.com/developerworks/db2/zones/cloudscape/

GoFurther



References

- Java and DB2 - http://www-128.ibm.com/developerworks/db2/zones/java/bigpicture.html
- DB2 Connect - http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0503katsnelson/

IDUG® 2007 North America

Session: G09

DB2 & Java for Dummies

John Mallonee

Highmark Inc.

john.mallonee@highmark.com

57

GoFurther