Java and Databases

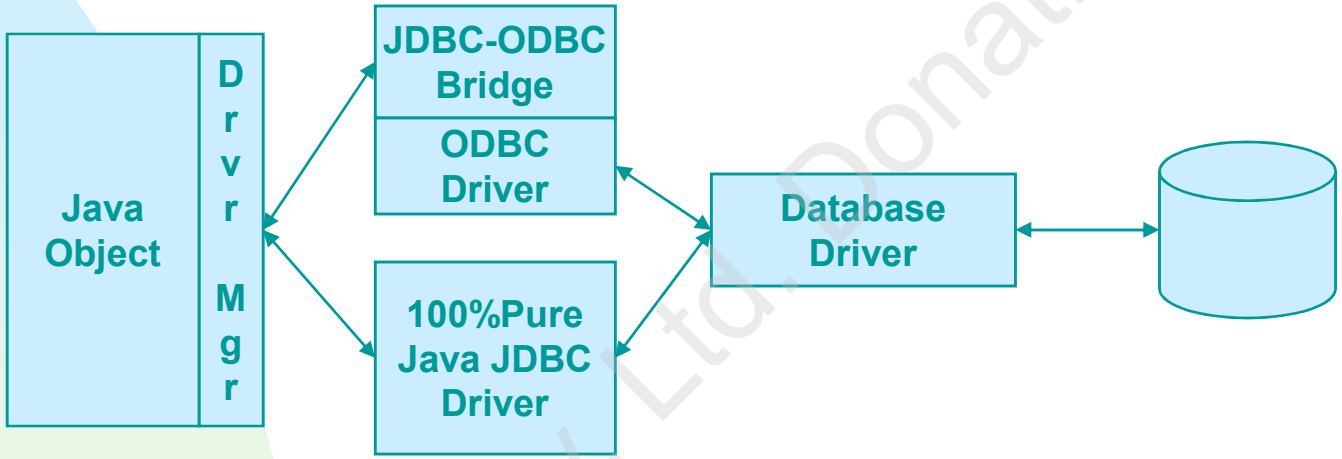
"…a core set of APIs that enables Java applications to connect to industry standard and proprietary database management systems. Using JDBC, your applications can retrieve and store information using Structured Query Language statements…"

Java and Databases

- Impetus
  - Data already in DBMS: don't want to duplicate in other domain
  - Utilise robust DBMS deployment and management tools
  - A modern DBMS can additionally hold:
    - HTML Pages, Images, Sounds, Video, Java...
  - Benefits
    - Recovery, integrity, security, mgmt tools, integration with advanced facilities

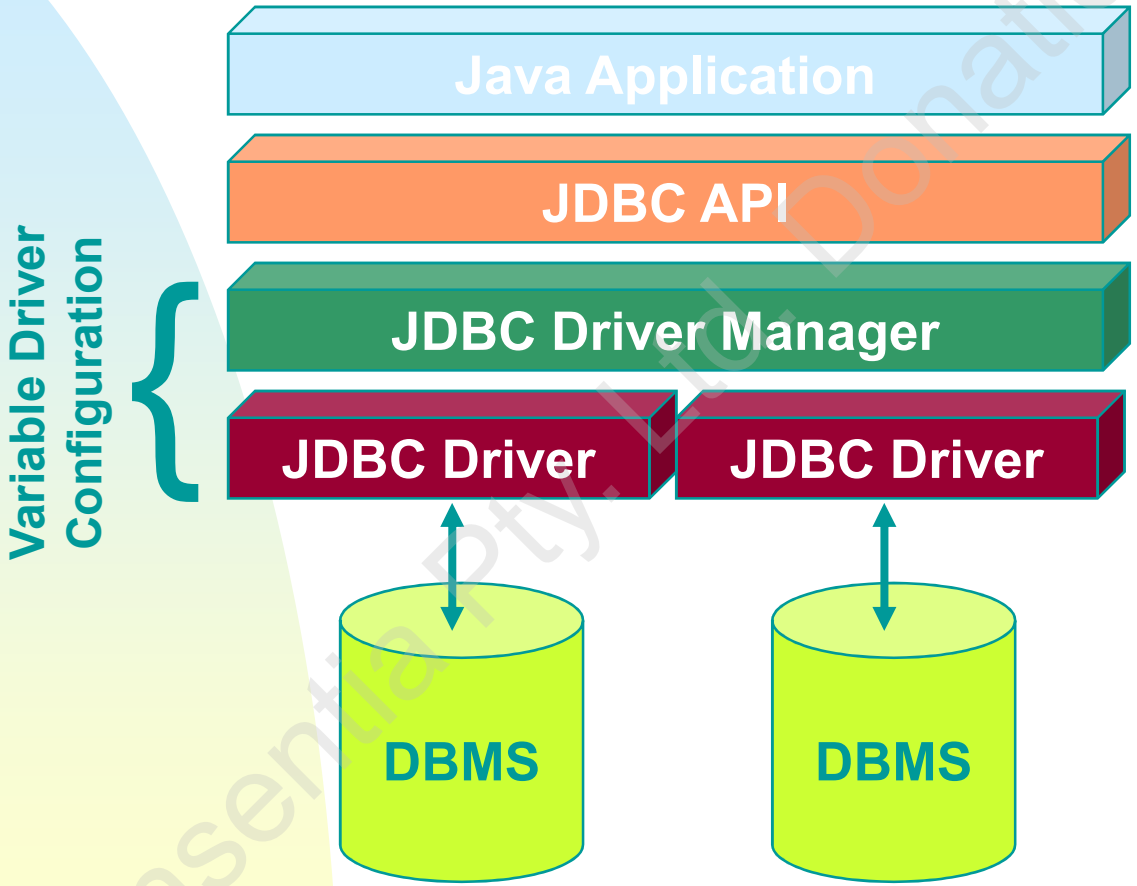
Java and Databases

- Overview



  - based (grudgingly!) on Microsoft's Open Database Connectivity (ODBC)
  - in the java.sql package

Java and Databases



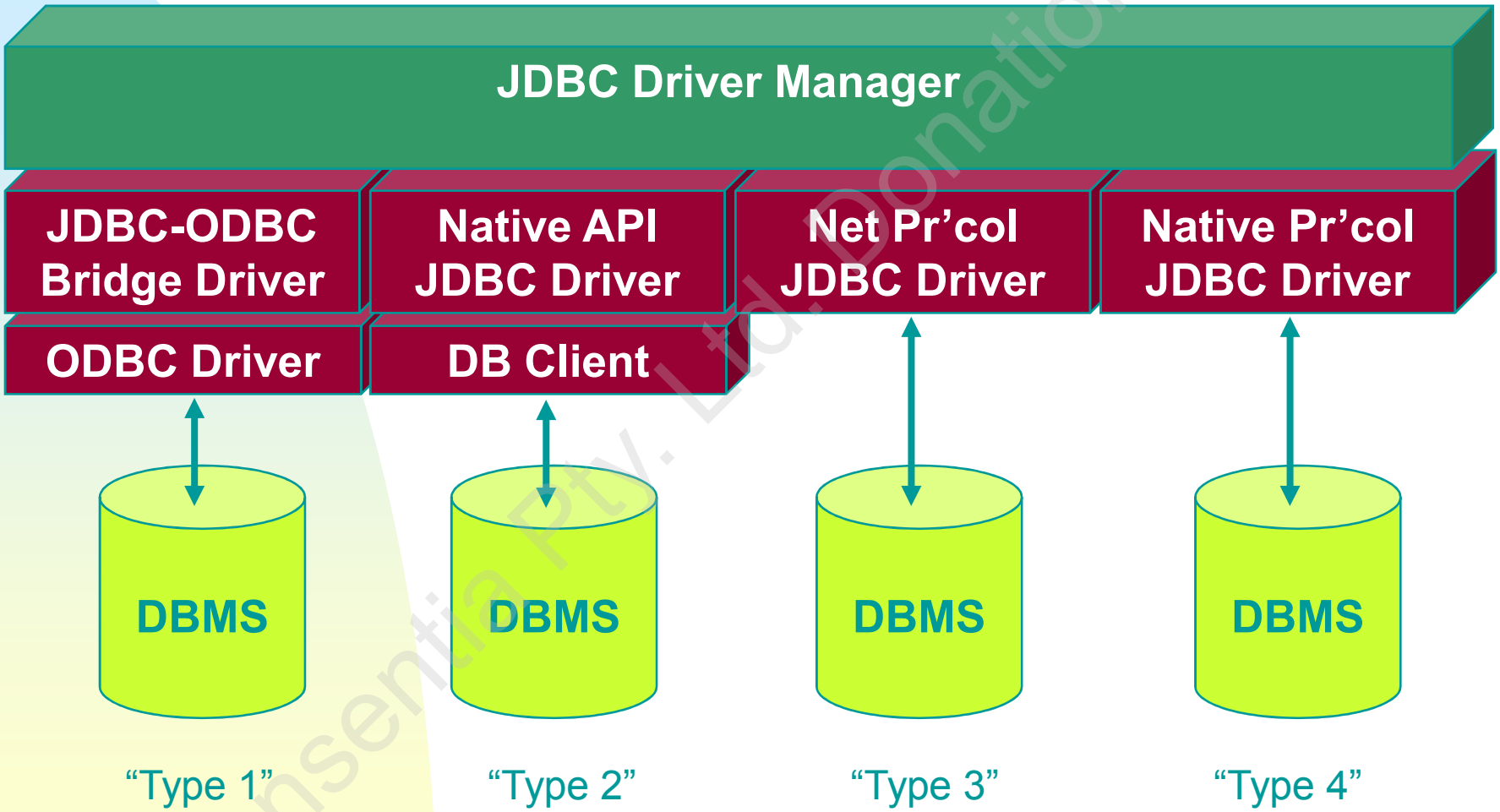
Java and Databases

- Several components of interest
  - DriverManager class
  - drivers
    - types
    - associated interfaces
  - miscellaneous classes
  - metadata facilities

Java and Databases

- DriverManager
  - support interaction between Java object and actual driver
  - can support many drivers of various types:
    - native API/partly Java
    - net protocol/all Java
    - native protocol/all Java
    - JDBC-ODBC bridge
  - also responsible for managing connection to database:
    - variety of getConnection () methods
    - setLoginTimeout (), setLogStream (), etc.

Java and Databases



Java and Databases

- A well-written Type 4 driver is likely to be the fastest driver, since it talks the native database protocol directly.
- A Type 4 driver may not be able to pass through a firewall if the native database protocol is not recognized by the firewall as valid. Type 3 drivers can avoid this is they use things like HTTP tunneling to "embed" their private protocol into a well-known protocol.
- Type 1 and 2 drivers cannot be used in “untrusted” applets.
- Type 1 and 2 drivers require that software be installed on client machines, whereas Type 3 and 4 drivers do not.
- Type 3 drivers can offer the benefits of middleware servers (connection pooling, data caching, and so on).
- Type 4 drivers are typically limited to a single database type, whereas Type 3 drivers can possibly communicate with databases from different vendors.

Java and Databases

- Drivers
  - conventionally pre-loaded (in static block) by:
    - Class.forName (*"fully.qualified.DriverName"*);
    - driver initialises itself when loaded
  - encapsulate 4 aspects:
    - Connection interface
    - Statement interface
      - PreparedStatement extension
      - CallableStatement extension
    - ResultSet interface
  - some methods:
    - acceptsURL (), jdbcCompliant ()
    - getPropertyInfo ()—what info should user supply

Java and Databases

- Connection interface
  - session between application and database
    - URL specifies DB and access protocol
      - also sub-protocol (where appropriate)
      - eg.: *jdbc:odbc://my.host.name:8888/ImportantStuff*
  - underlying object can only be obtained by call to
    - DriverManager.getConnection ()
  - some methods:
    - commit ()
      - most connections use ‘autocommit’ mode
    - rollback ()
    - close ()
    - setReadOnly ()

Transaction support

Java and Databases

```
import java.sql.*;
import java.util.Properties;

class Query
  {
  private String url = "jdbc:odbc:InventoryDatabase";
  private Connection dbConnection;

  // ensure the driver is loaded, initialised and registered
  static
    {
    Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
    }

  public Query (String userName, String passwd)
    {
    Properties info = new Properties ();

    info.put (“user”, userName);
    info.put (“password”, passwd);

    dbConnection = DriverManager.getConnection (url, info);
    }

  …
  }
```

Java and Databases

- Statement interface
  - envelopes an SQL query/command
  - statement is composed and forwarded to DBMS
  - methods
    - execute ()
      - *"Execute a SQL statement that may return multiple results. Under some (uncommon) situations a single SQL statement may return multiple result sets and/or update counts. … you're executing a stored procedure that you know may return multiple results, or … you're dynamically executing an unknown SQL string."*
    - executeQuery ()
      - execute a SQL SELECT known to return a single ResultSet
    - executeUpdate ()
      - UPDATE, DELETE, INSERT commands
      - return value indicates how many tuples were updated

Java and Databases

```
selectText = "SELECT firstname,lastname FROM addresses";
Statement selectStmt = dbConnection.createStatement ();
ResultSet result = selectStmt.executeQuery (selectText);
```

```
insertText = "INSERT INTO addresses VALUES (1,'John','Smith')";
Statement insertStmt = dbConnection.createStatement ();
int rowsInserted = insertStmt.executeUpdate (insertText);
```

Java and Databases

- PreparedStatment extends Statement
  - statement is pre-compiled and stored in a PreparedStatement object. Can use this to efficiently execute the statement multiple times
  - allows IN parameters within the query
    - referenced by number
    - setXXX () methods used

```
PreparedStatement ps = theConnection.prepareStatement
  (
  "UPDATE PresidentialCandidate " +
    "SET popularVote = ? " +
    "WHERE candidate = ?"
  );

ps.setInt (1, 50);
ps.setString (2, "Mickey Mouse");

ps.executeUpdate ();
```

Java and Databases

- can use a stream to send the contents of a file as an IN parameter:

```
java.io.File file = new java.io.File ("/tmp/data");
int fileLength = file.length ();
java.io.InputStream fin = new java.io.FileInputStream (file);
java.sql.PreparedStatement pstmt = connection.prepareStatement
  (
  "UPDATE Table5 SET stuff = ? WHERE index = 4"
  );
pstmt.setBinaryStream (1, fin, fileLength);
pstmt.executeUpdate ();
```

Java and Databases

- CallableStatement extends PreparedStatement
  - execute stored procedures via an SQL escape
  - allows for possible result parameters—register as OUT parameters
    - registerOutParameter ()
      - {? = call *FUNCTIONNAME* [,, ...]}
      - {call *PROCEDURENAME* [,, ...]}
      - {call *PROCEDURENAME* }
  - type of all OUT parameters must be registered prior to executing the stored procedure
    - java.sql.Type
    - getXXX ()
  - retrieve ResultSet data before OUT parameters

Java and Databases

```
CallableStatement cs = null;
try
  {
  cs = connection.prepareCall
    (
    "{ ? = call shipping_and_invoice (?, ?, ?, ?)}"
    );

  cs.registerOutParameter (1, java.sql.Types.Bit);
  cs.setString (2, selectedItemID);
  cs.setInt (3, quantityPurchased);
  cs.setString (4, userName);
  cs.setString (5, userAddress);

  cs.executeUpdate ();

  if (cs.getBoolean (1))
    System.out.println ("Request completed OK.");
  }

catch (final SQLException e)
  {
  }

finally
  {
  if (cs != null)
    cs.close ();
  }
```

Java and Databases

- java.sql.Type to JDBC/SQL type correspondence

Java Type	JDBC type
String	VARCHAR or LONGVARCHAR
java.math.BigDecimal	NUMERIC
boolean	BIT
byte	TINYINT
short	SMALLINT
int	INTEGER
long	BIGINT
float	REAL
double	DOUBLE
byte []	VARBINARY or LONGVARBINARY
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP

Java and Databases

		TINYINT	SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC	BIT	CHAR	VARCHAR	LONGVARCHAR	BINARY	VARBINARY	LONGVARBINARY	DATE	TIME	TIMESTAMP
byte	getByte()	■	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒						
short	getShort()	▒	■	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒						
int	getInt()	▒	▒	■	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒						
long	getLong()	▒	▒	▒	■	▒	▒	▒	▒	▒	▒	▒	▒	▒						
float	getFloat()	▒	▒	▒	■	▒	▒	▒	▒	▒	▒	▒	▒	▒						
double	getDouble()	▒	▒	▒	▒	■	■	▒	▒	▒	▒	▒	▒	▒						
java.math.BigDecimal	getBigDecimal()	▒	▒	▒	▒	▒	▒	■	■	▒	▒	▒	▒	▒						
boolean	getBoolean()	▒	▒	▒	▒	▒	▒	▒	▒	■	▒	▒	▒	▒						
String	getString()	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	■	■	▒	▒	▒	▒	▒	▒	▒
byte[]	getBytes()														■	■	▒			
java.sql.Date	getDate()											▒	▒	▒				■		▒
java.sql.Time	getTime()											▒	▒	▒					■	▒
java.sql.Timestamp	getTimestamp()											▒	▒	▒				▒		■
java.io.InputStream	getAsciiStream()											▒	▒	■	▒	▒	▒			
java.io.InputStream	getUnicodeStream()											▒	▒	■	▒	▒	▒			
java.io.InputStream	getBinaryStream()														▒	▒	■			
java.lang.Object	getObject()	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒

Java and Databases

- SQLException
  - overall process is a long chain of actions: any/multiple exceptions can be raised
  - JDBC allows chained exceptions
    - getNextException ()
  - three components, accessed via:
    - getMessage
    - getSQLState
    - getErrorCode

```
try
  {
  // something that might blow up...
  }
catch (SQLException ex)
  {
  System.out.println ("\n--- SQLException caught ---\n");
  while (ex != null)
    {
    System.out.println ("SQLState:  " + ex.getSQLState ());
    System.out.println ("ErrorCode: " + ex.getErrorCode ());
    System.out.println ("Message:   " + ex.getMessage ());
    System.out.println();
    ex = ex.getNextException ();
    }
  }
```

```
 --- SQLException caught ---
SQLState:  42501
ErrorCode: 271
Message:   There is already an object named 'Widgets' in the database. Severity 16, State 1, Line 1
```

Java and Databases

- SQLWarning
  - extends SQLException
    - don't stop the execution of a process
  - information on a database access warnings
  - actually uncommon; most common warning is DataTruncation
    - thrown when JDBC unexpectedly truncates a data value—reports a DataTruncation warning (on reads) or a DataTruncation exception (on writes)
  - Connection, ResultSet, Statement can generate warnings

Java and Databases

```
Statement stmt = con.createStatement ();
ResultSet rs = stmt.executeQuery ("SELECT widget_name FROM widgets");
SQLWarning warning = stmt.getWarnings ();
if (warning != null)
  {
  System.out.println ("\n---Execution Warning---\n");
  while  (warning != null)
    {
    System.out.println ("Message: " + warning.getMessage ());
    System.out.println ("SQLState: " + warning.getSQLState ());
    System.out.println ("Vendor error code: " + warning.getErrorCode ());
    System.out.println ();
    warning = warning.getNextWarning ();
    }
  }
System.out.println ("Widgets available at widget central:");
while (rs.next ())
  {
  System.out.println ("    " + rs.getString ("widget_name"));
  SQLWarning warn = rs.getWarnings ();
  if (warn != null)
    {
    System.out.println ("\n---ResultSet Warning---\n");
    while (warn != null)
      {
      System.out.println ("Message: " + warn.getMessage ());
      System.out.println ("SQLState: " + warn.getSQLState ());
      System.out.println ("Vendor error code: " + warn.getErrorCode ());
      System.out.println ();
      warn = warn.getNextWarning ();
      }
    }
  }
```

Java and Databases

- ResultSet interface
  - provides access to a table of data generated by executing a Statement
    - obtained through statement's getResultSet method
  - table rows are retrieved in sequence via a *cursor* pointing to the current row of data
  - some methods:
    - next ()
    - getXXX () methods
    - wasNull ()
    - close ()

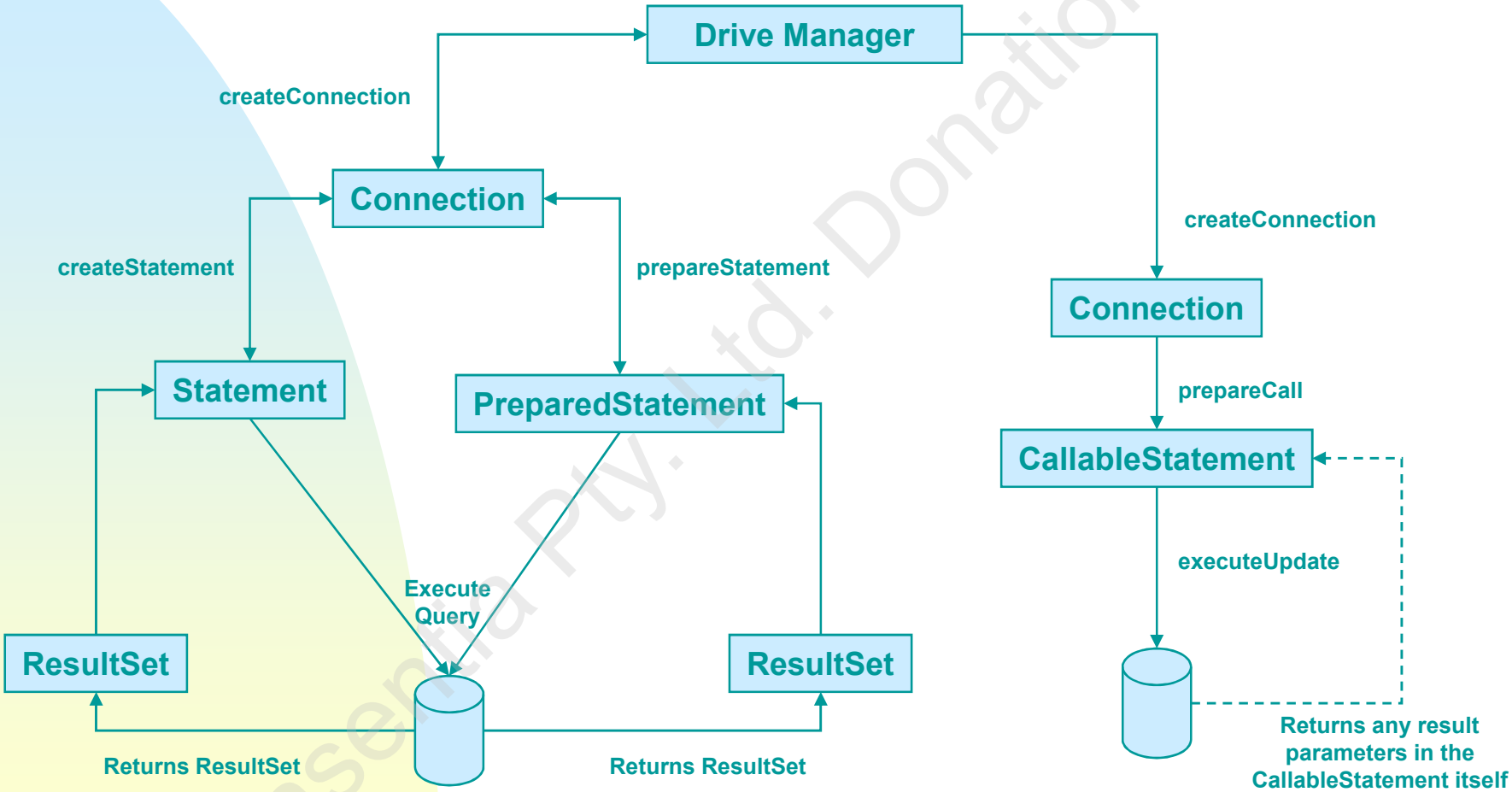
Java and Databases

```
void dispResultSet (ResultSet rs) throws SQLException
  {
  while (rs.next ())
    {
    System.out.print ("" + rs.getInt (1))
    String firstName = rs.getString ("FIRST_NAME");
    System.out.print (rs.wasNull () ? "[no first name]" : firstName);
    System.out.print (rs.getString ("LAST_NAME"));
    System.out.println (rs.getDate (4));
    }

  rs.close ();
  }
```

Java and Databases

- Interactions—a summary



Java and Databases

- Database Metadata
  - provides information about the database as a whole
  - often via return lists of information in ResultSets
  - *many* methods, including:
    - isReadOnly ()
    - getSQLKeywords ()
    - supportsStoredProcedures ()
    - supportsTransactions ()
    - supportsANSI92FullSQL ()
    - getProcedures ()
    - etc.

Java and Databases

- ResultSet Metadata
  - used to find out about the types and properties of the columns in a ResultSet
  - enhances reusability of processing code
  - many methods:
    - getColumnType ()
    - getColumnName ()
    - getColumnDisplaySize ()
    - isWriteable ()
    - etc.

Java and Databases

```
void dispResultSet (ResultSet rs) throws SQLException
  {
  ResultSetMetaData rsmd = rs.getMetaData ();

  int numCols = rsmd.getColumnCount ();

  for (int i = 1; i <= numCols; i ++)
    {
    if (i > 1)
      System.out.print (",");
    System.out.print (rsmd.getColumnLabel (i));
    }

  System.out.println ("");

  while (rs.next ())
    {
    for (int i = 1; i <= numCols; i ++)
      {
      if (i > 1)
        System.out.print (",");
      System.out.print (rs.getString (i));
      }
    System.out.println ("");
    }
}
```

Java and Databases

- JBDC 2.0
  - update introduced alongside Java 1.2
  - scrollable result sets
    - forward and backward movement, plus relative and absolute positioning. JDBC 2.0 also allows result sets to be directly updatable.

```
Connection con = DriverManager.getConnection ("jdbc:odbc:Salaries");
Statement stmt = con.createStatement (ResultSet.TYPE_SCROLL_SENSITIVE,
                                      ResultSet.CONCUR_UPDATABLE);
stmt.setFetchSize (25);
ResultSet rs = stmt.executeQuery ("SELECT position, salary FROM employees");

rs.first();
rs.updateString (1, "General Dogsbody");
rs.updateFloat("salary", rs.getFloat (2) - 10000.0F);
rs.updateRow();
```

Java and Databases

- batch updates
  - allows for multiple update statements (insert/update/delete) in a single request

```
con.setAutoCommit (false);
Statement stmt = con.createStatement ();
stmt.addBatch ("INSERT INTO character VALUES (99, 'Homer Simpson')");
stmt.addBatch ("INSERT INTO shows VALUES (10101, 'The Simpsons')");
stmt.addBatch ("INSERT INTO starring_in VALUES (99, 10101)");
stmt.addBatch ("DELETE FROM character WHERE name = 'Popeye'");

int [] updateCounts = stmt.executeBatch();
```

- support for storing persistent Java objects & SQL3 data types
  - application can also customize mapping of SQL3 structured types into Java language classes

```
PreparedStatement pstmt = con.preparedStatement
  ("UPDATE PERSONNEL SET Employee = ? WHERE Employee.no = 1001");
pstmt.setObject (1, emp);
pstmt.executeUpdate();

ResultSet rs = stmt.executeQuery ("SELECT Employee FROM PERSONNEL");
rs.next();
Employee emp = (Employee) rs.getObject (1);
```

Java and Databases

- rowsets
  - *"...make it easy to send tabular data over a network. They can also be used to provide scrollable result sets or updatable result sets when the underlying JDBC™ driver does not support them."*
  - important for JavaBeans
  - may or may not maintain an open database connection for "through" changes

```
public RowSet getAllTV_stars() throws SQLException
{
 Connection con = null;
 try
   {
   con = ds.getConnection ("username", "password");
   Statement stmt = con.createStatement ();
   ResultSet rs = stmt.executeQuery ("select * from TV_stars");
   CachedRowSet rset = new CachedRowSet ();
   rset.populate (rs);
   rs.close (); stmt.close ();
   return rset;
   }
 finally
   { if (con != null) con.close(); }
}
```

Java and Databases

- JNDI for naming databases
  - making the application code independent of a particular JDBC driver and JDBC URL
- connection pooling for improved performance
- distributed transaction support
  - standard 2-phase commit protocol used by the Java Transaction Service (JTS)
- miscellaneous new features
  - Unicode
  - java.math.BigDecimal
  - support for time zones
- a new package javax.sql
  - the parts of JDBC 2.0 which are related to other standard extensions: JNDI and JTS

Java and Databases

- SQLJ
  - Embedded SQL in Java
  - Similar to ANSI/ISO “Embedded SQL” for C, COBOL, FORTRAN, etc.
    - Oracle, IBM, Tandem, Sybase & JavaSoft seeking SQLJ as Java implementation of this standard
  - Predominantly for *static* SQL programs
  - Requires additional precompilation/translation
  - Syntax: #sql { SQL clauses }

```
void print_address (String name)
  {
  String addr;
  #sql { SELECT ADDRESS INTO :addr FROM PEOPLE
           WHERE NAME = :name };
  System.out.println( name + " lives at " + addr );
  }
```

Java and Databases

- Tips
  - Can't use JDBC-ODBC bridge from untrusted applets (i.e Browser-executed), so use all-Java driver whenever possible to avoid security problems, or use RMI-JDBC solution
  - JDK 1.0.2 doesn't support java.sql.*, so rename as jdbc.sql.* and download from server if required in older browsers
  - Watch for potential variations in behaviour of xxxMetaData interfaces for different drivers and databases with differing capabilities