

Power Programming in Java

Session: 8

JDBC API



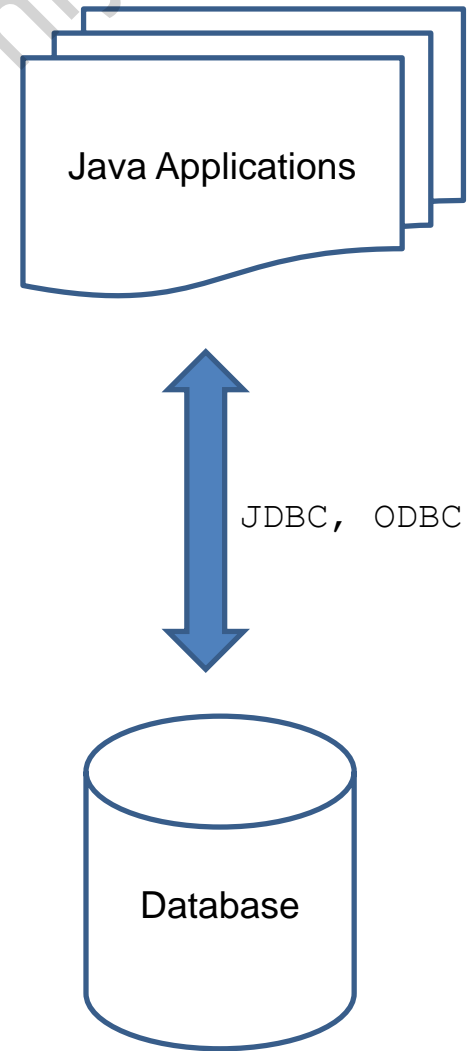


- ◆ Describe JDBC
- ◆ Describe JDBC Driver Types
- ◆ Explain JDBC Application Development Process
- ◆ Describe Database Meta Information

For Aptech Centre Use Only



- ◆ Activities in a database application involve opening a connection, communicating with a database, executing SQL statements, and retrieving query results.
- ◆ To connect Java applications with databases, API software for database connectivity, known as Java Database Connectivity (JDBC), is used.
- ◆ This software API is a collection of application libraries and database drivers, whose implementation is independent of programming languages, database systems, and operating systems.
- ◆ Open DataBase Connectivity (ODBC) and JDBC are two widely used APIs for such activities.





ODBC is an API provided by Microsoft for accessing the database.

It uses Structured Query Language (SQL) as its database language.

It provides functions to insert, modify, and delete data and obtain information from the database.

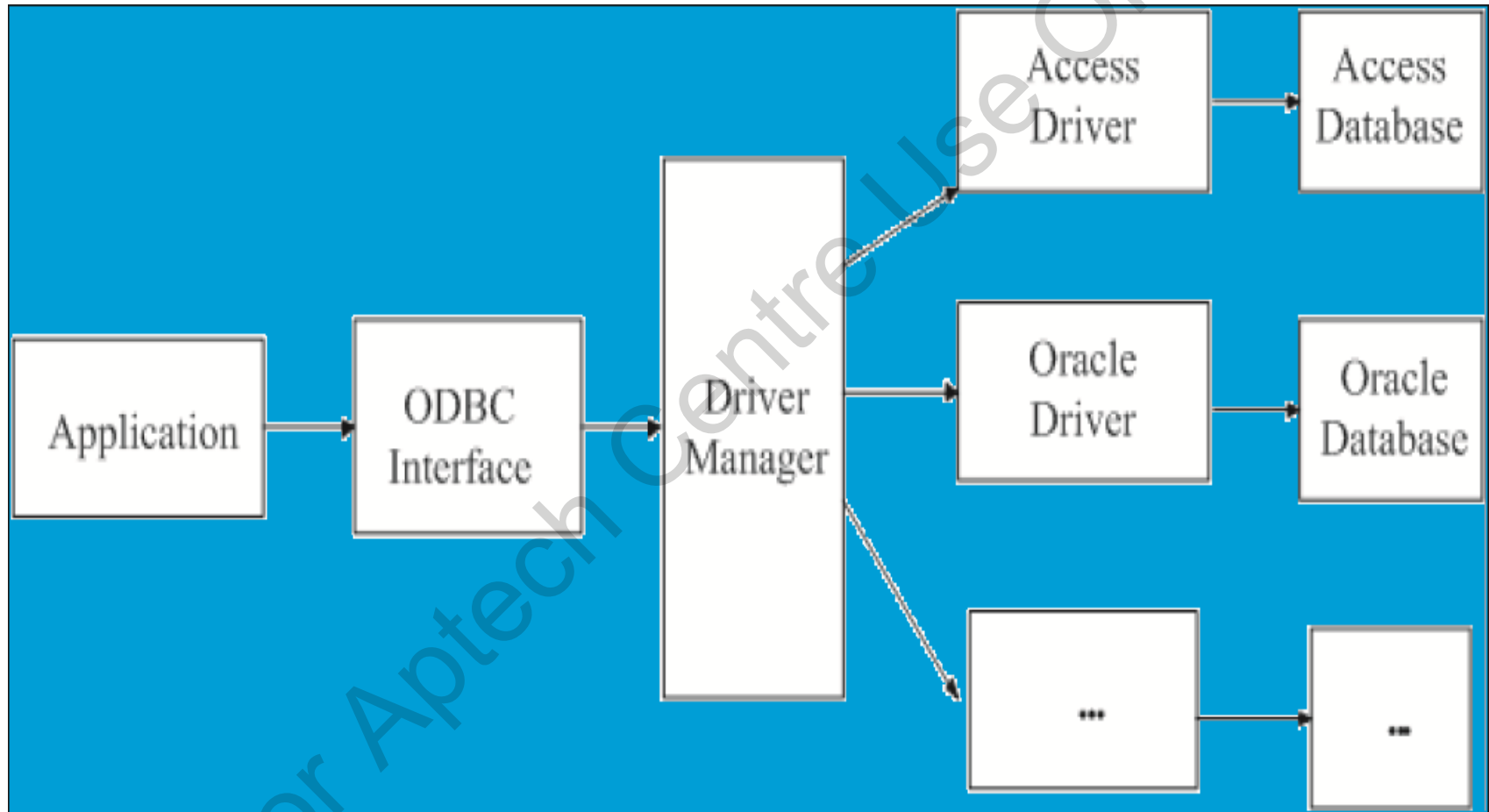
The application can be a GUI program written in Java, VC++, or any other software.

The application makes use of ODBC to connect with the databases.

The driver manager is part of Microsoft ODBC API and is used to manage various drivers in the system including loading.



- ◆ Figure displays an ODBC connection.





JDBC API has a set of classes and interfaces written in Java used for accessing tabular data.



Combination of JDBC API and Java platform offers advantage of accessing any type of data source and flexibility of running on any platform which supports JVM.



A single program with JDBC implementation can send Structured Query Language (SQL) or other statements to the suitable data source or database.



Three tasks that can be performed by using JDBC drivers are as follows:

- Establish a connection with the data source
- Send queries and update statements to the data source
- Process the results



JDBC is ODBC translated into an object-oriented interface that is natural for Java programmers.

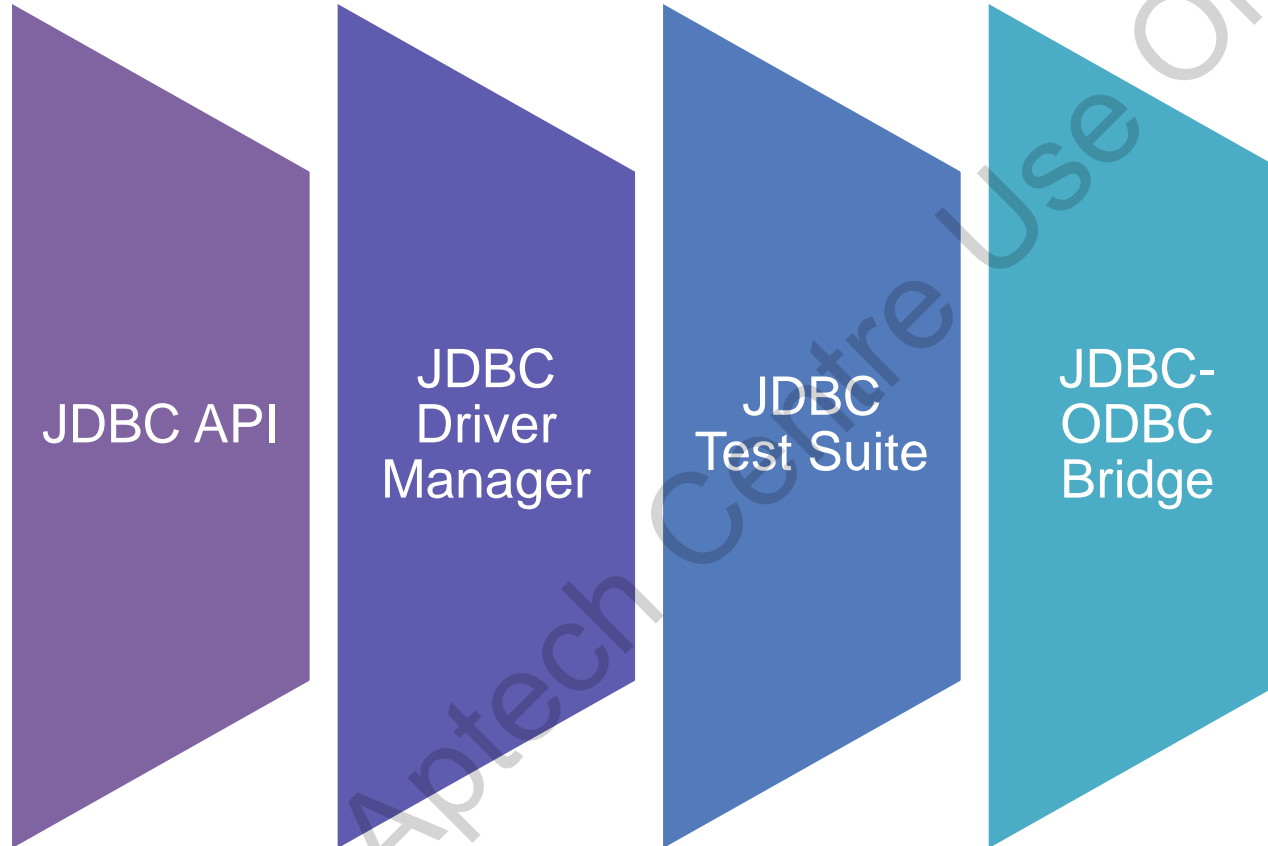
JDBC keeps things simple while allowing more advanced capabilities wherever required.

The JDBC code is installable, portable, and secure on all Java platforms from network computers to mainframes.

JDBC is a standard interface for Java programs to access relational databases.

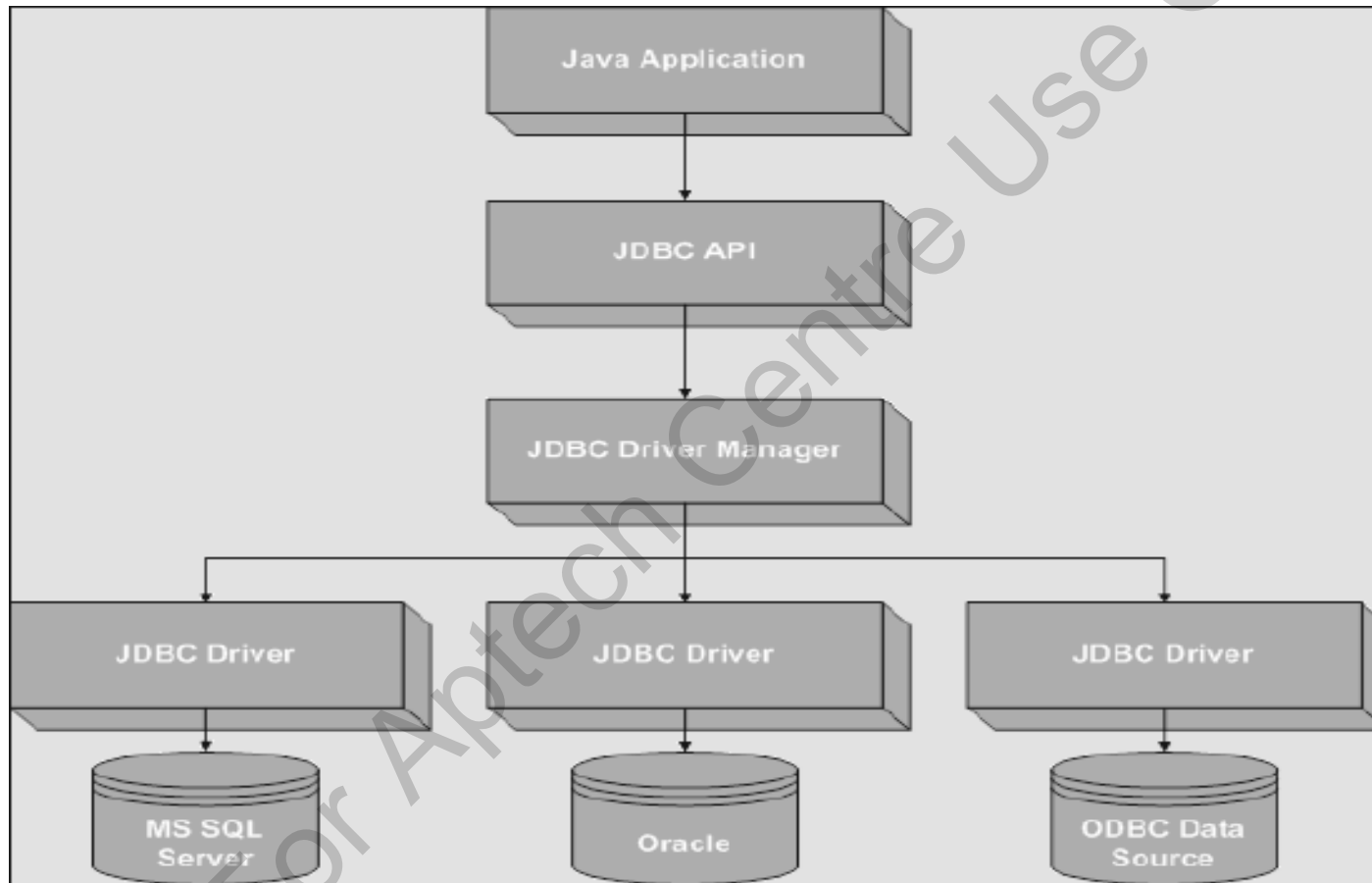
For Aptechn Centre Use Only

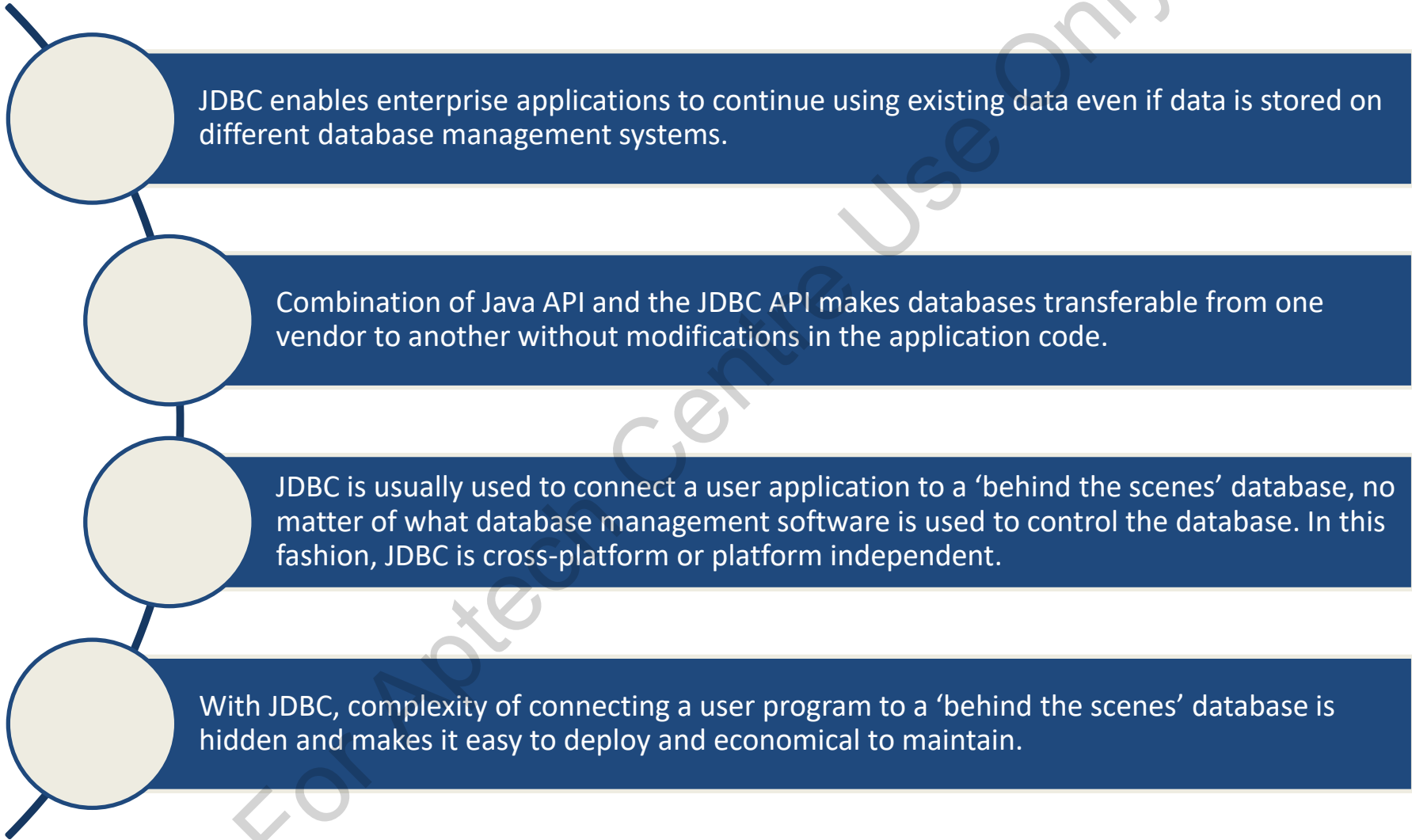
Product Components of JDBC





- ◆ Figure shows location of driver manager with respect to JDBC drivers and applications.





JDBC enables enterprise applications to continue using existing data even if data is stored on different database management systems.

Combination of Java API and the JDBC API makes databases transferable from one vendor to another without modifications in the application code.

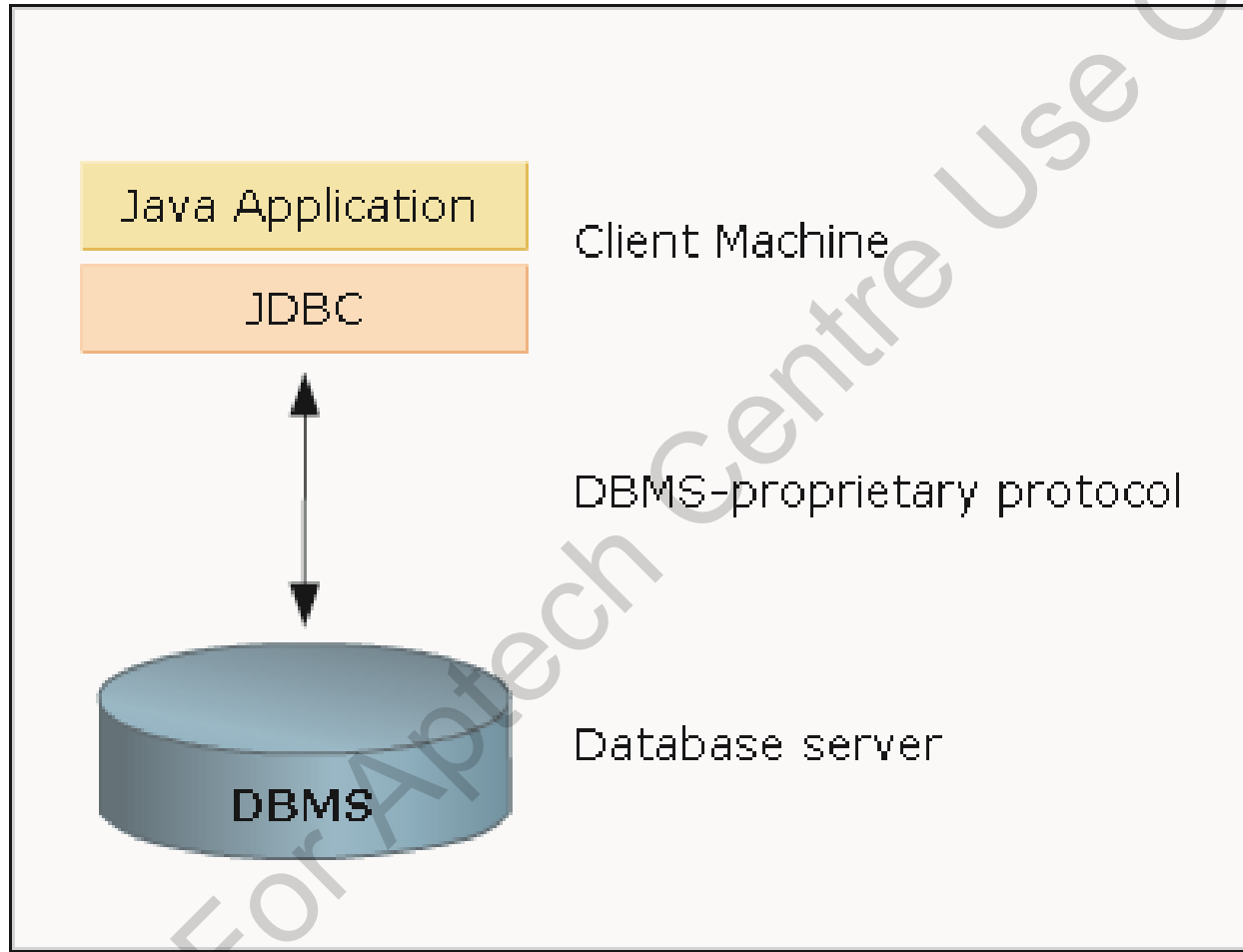
JDBC is usually used to connect a user application to a 'behind the scenes' database, no matter of what database management software is used to control the database. In this fashion, JDBC is cross-platform or platform independent.

With JDBC, complexity of connecting a user program to a 'behind the scenes' database is hidden and makes it easy to deploy and economical to maintain.

Two-Tier Data Processing Model



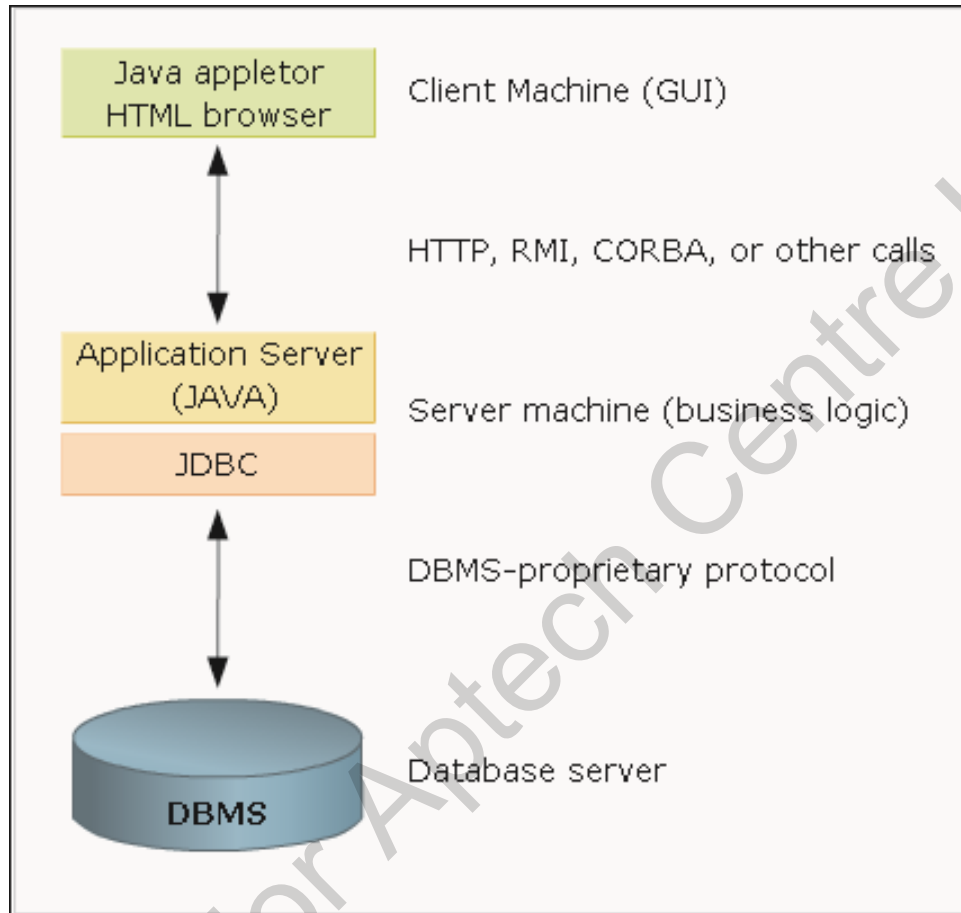
- ◆ Figure displays the two-tier data processing model.



Three-Tier Data Processing Model

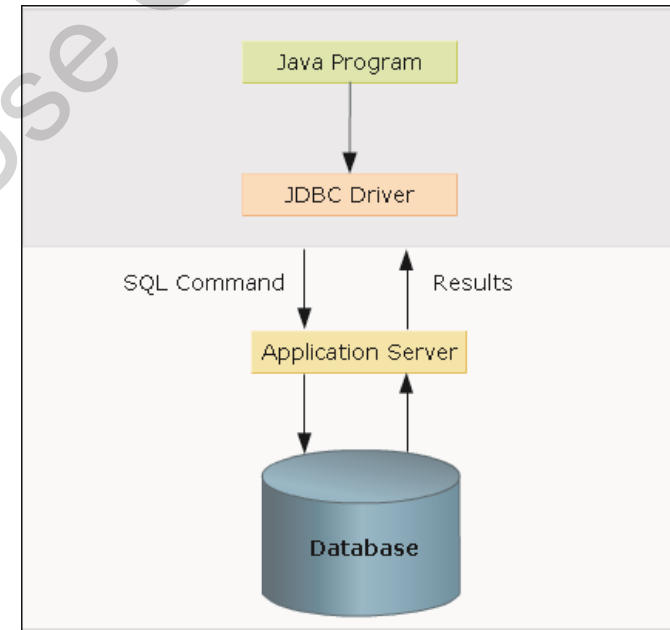


- ◆ Figure displays the three-tier data processing model.





- ◆ Is a collection of specifications that defines how databases and applications communicate with each other.
- ◆ Its core is based on Java, so, it is used as common platform for building middle-tier of three-tier architecture.

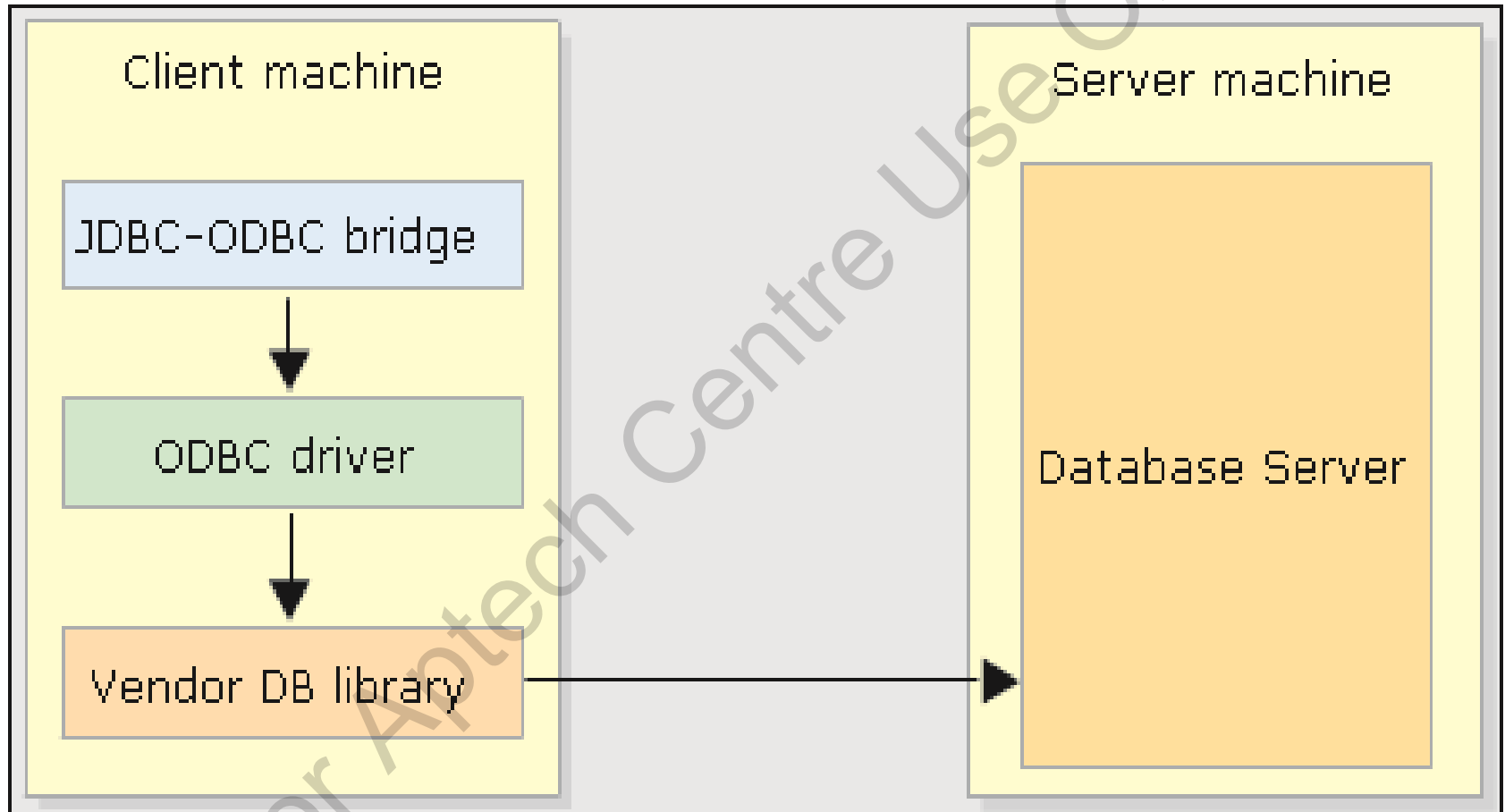




Driver Type	Driver Name	Description
Type I	ODBC-JDBC Bridge	Translates JDBC calls into ODBC calls
Type II	Native API-Java/ Partly Java	Translates JDBC calls into database-specific calls or native calls
Type III	JDBC Network-All Java	Maps JDBC calls to the underlying 'network' protocol, which in turn calls native methods on the server
Type IV	Native Protocol-All Java	Directly calls RDBMS from the client machine



- ◆ Figure displays the JDBC type 1 driver.





Features:

- ◆ The Type 1 drivers use a bridging technology that provides JDBC access via ODBC drivers.
- ◆ This establishes a link between JDBC API and ODBC API.
- ◆ The ODBC API is in turn implemented to get the actual access to the database via the standard ODBC drivers.

Advantages:

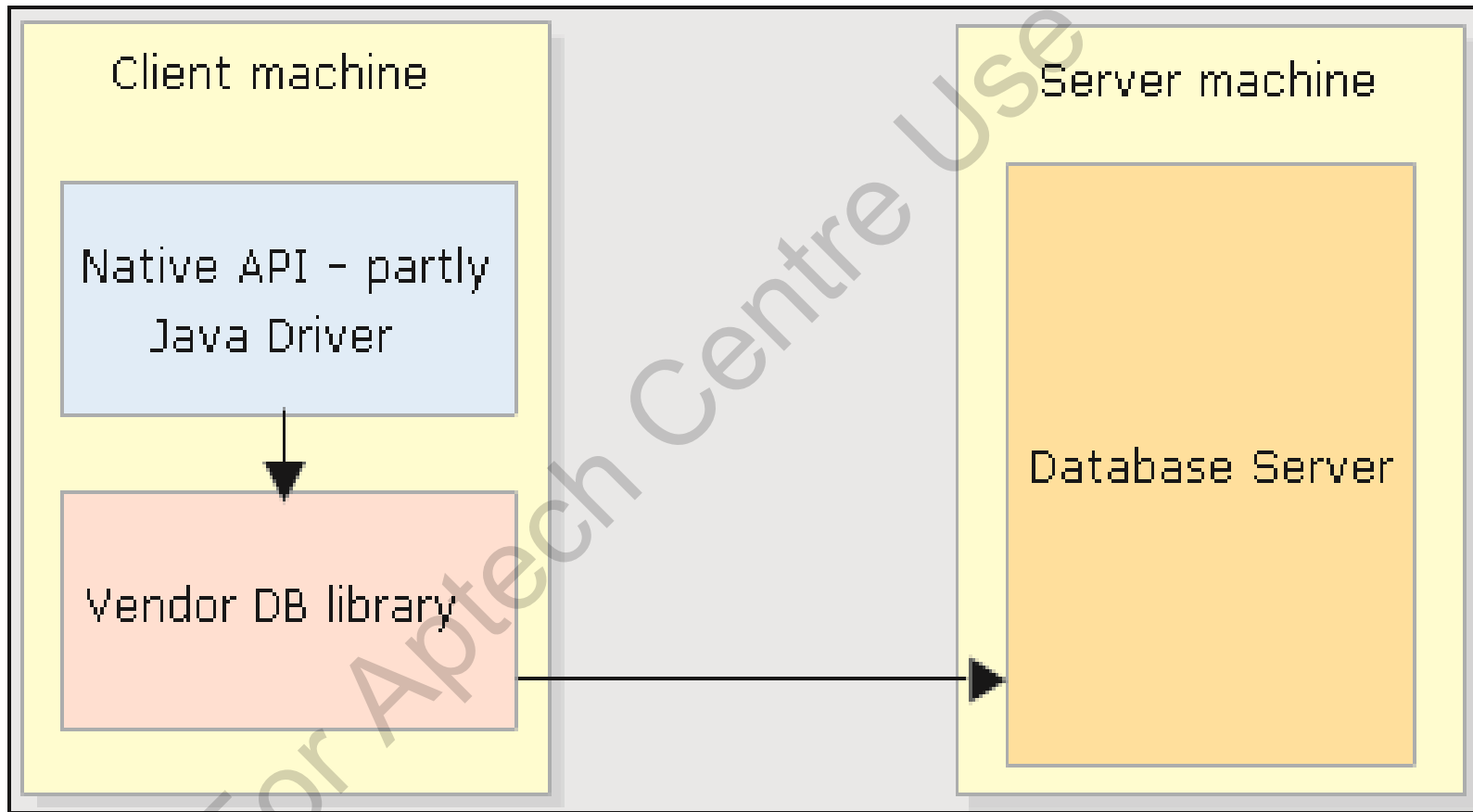
- ◆ The Type 1 drivers are written to allow access to various databases through pre-existing ODBC drivers.
- ◆ In some cases, they are the only option to databases like MS-Access or Microsoft SQL Server for having ODBC native call interface.

Disadvantages:

- ◆ The Type 1 driver does not hold good for applications that do not support software installations on client machines.
- ◆ The native ODBC libraries and the database client codes must reside on the server, which in turn reduces the performance.



- ◆ Figure displays the JDBC type 2 driver.





Features:

- ◆ Type 2 driver comprises Java code that converts JDBC calls into calls on a local database API for Oracle, Sybase, DB2, or any other type of DBMS.
- ◆ This implies that driver calls native methods of individual database vendors to get database access.
- ◆ This kind of driver basically comes with database vendors to interpret JDBC calls to the database-specific native call interface, for example, Oracle provides OCI driver.
- ◆ Type 2 driver also requires native database-specific client libraries to be installed and configured on the client machine like Type 1 drivers.



Advantages:

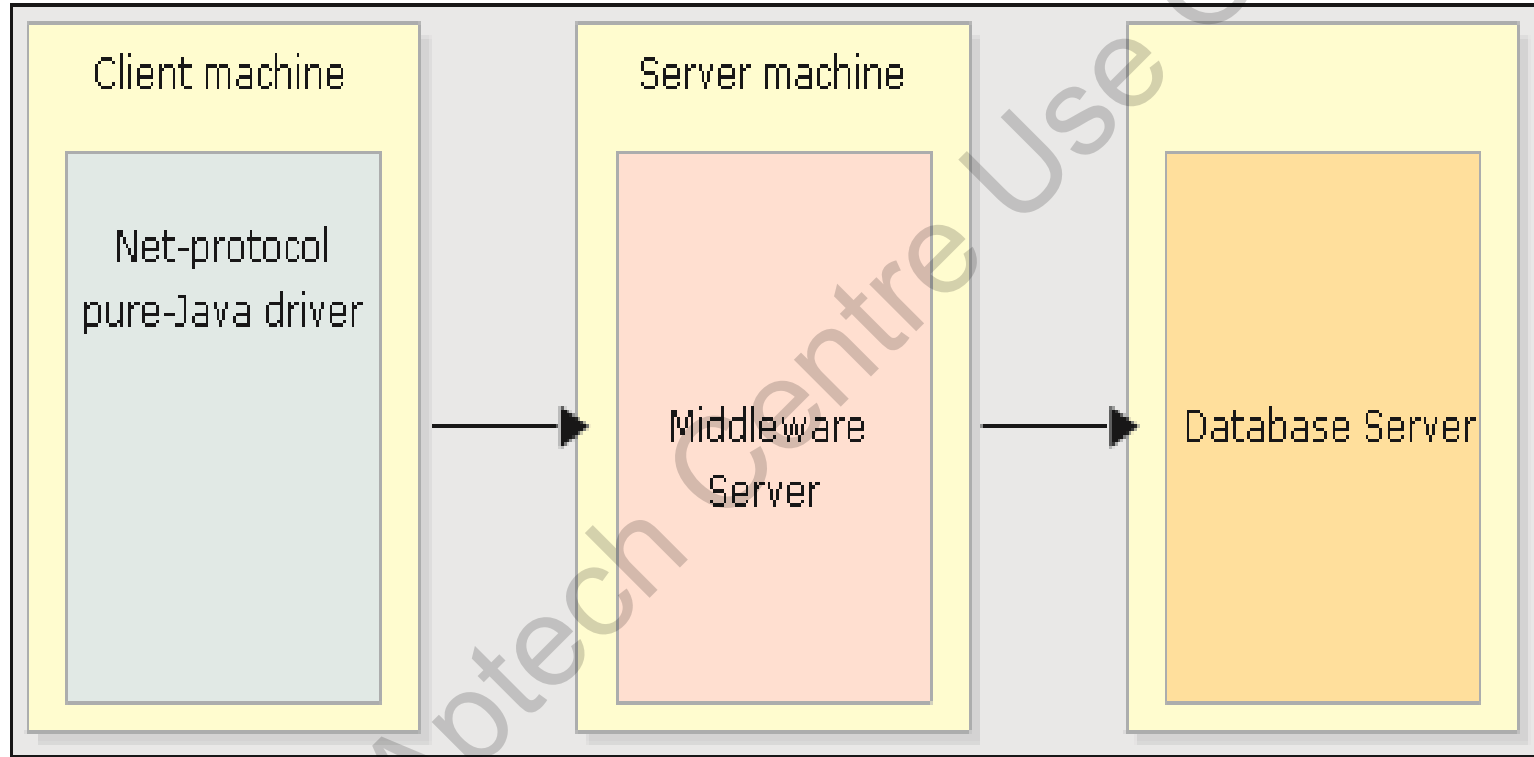
- ◆ The Type 2 driver yields better performance than Type 1 driver.
- ◆ Type 2 drivers are generally faster than Type 1 drivers as the calls get converted to database-specific calls.

Disadvantages:

- ◆ The Type 2 driver does not support applications that do not allow software installations on client machines as it requires native database codes to be configured on client machines.
- ◆ These database specific native code libraries must reside on the server, in turn reducing the performance.



- ◆ Figure displays the JDBC type 3 driver.





Features:

- ◆ The Type 3 driver is a pure Java driver that converts the JDBC calls into a DBMS-independent network protocol, which is again translated to database-specific calls by a middle-tier server.
- ◆ This driver does not require any database-specific native libraries to be installed on the client machines.
- ◆ The Web-based applications should preferably implement Type 3 drivers as this driver can be deployed over the Internet without installing a client.

Advantages:

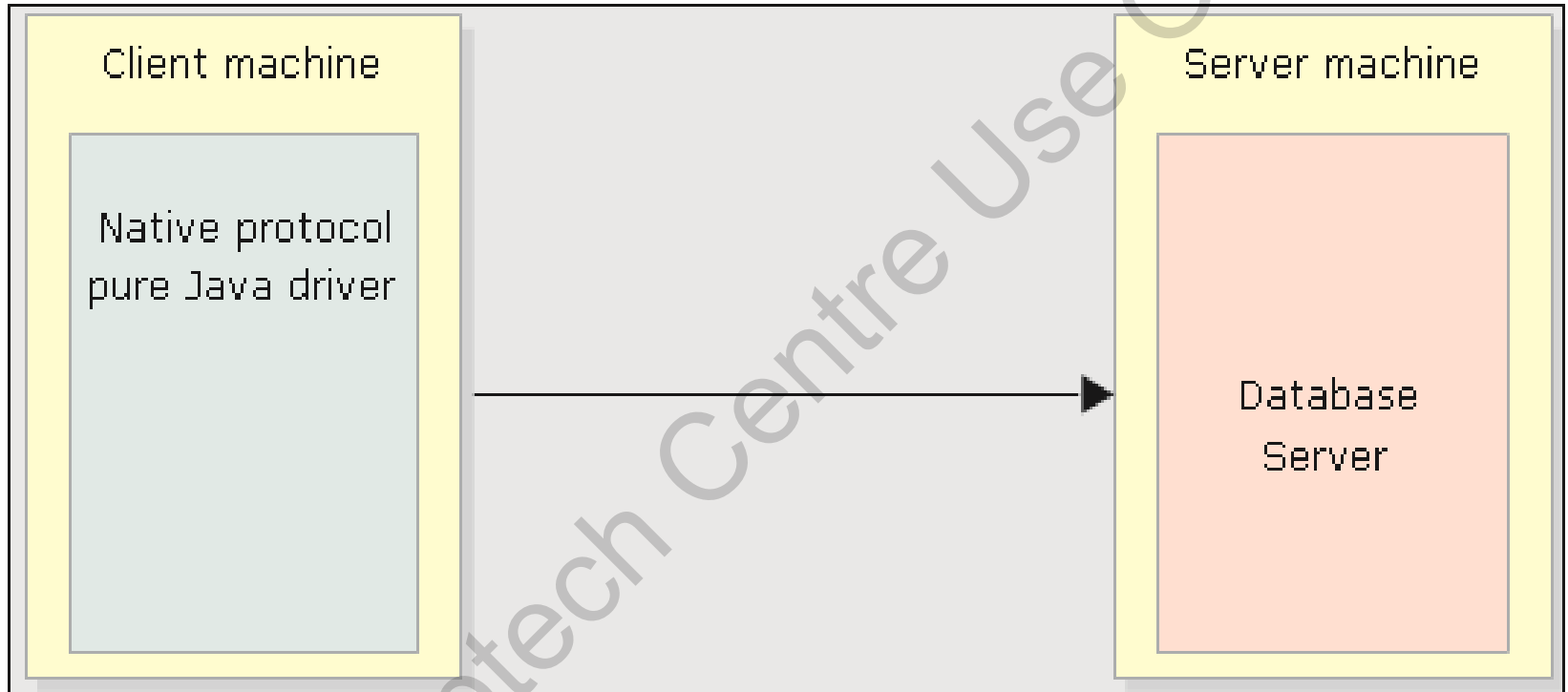
- ◆ The Type 3 driver is the most flexible type as it does not require any software or native services to be installed on client machine.
- ◆ It provides a high degree of adaptability to change and control underlying database without modifying the client side driver.

Disadvantages:

- ◆ Database-specific code must be executed in the middle-tier server.
- ◆ As it supports Web-based applications, it must implement additional security like access through firewalls.



- ◆ Figure displays JDBC type 4 driver.





Features:

- ◆ Type 4 drivers are pure Java drivers that convert JDBC calls into network protocol that communicates directly with database.
- ◆ This links client call directly with DBMS server and provides a practical solution for accessing Intranet.
- ◆ In most cases, drivers are provided by database vendors and do not require any database-specific native libraries to be configured on client machine and can be deployed on the Web without installing a client, as required for Type 3 drivers.

Advantages:

- ◆ Type 4 drivers communicate directly with the database engine using Java sockets, rather than through middleware or a native library.
- ◆ This is the reason that these drivers are the fastest JDBC drivers available.
- ◆ No additional software like native library is required for installation on clients.

Disadvantages:

- ◆ These are database-specific.
- ◆ Hence, if in case, the back-end database changes, the application developer may have to purchase and deploy a new Type 4 driver specific to the new database.



- ◆ JDBC API defines a set of interfaces and classes to communicate with the database.
- ◆ Following are some of the interfaces in this package:
 - ◆ **Connection**: This is used to maintain and monitor database sessions. Data access can also be controlled using the transaction locking mechanism.
 - ◆ **DatabaseMetaData**: This interface provides database information such as the version number, names of the tables, and functions supported. It also has methods that help in discovering database information such as the current connection, tables available, schemas, and catalogues.
 - ◆ **Driver**: This interface is used to create Connection objects.
 - ◆ **PreparedStatement**: This is used to execute pre-compiled SQL statements.
 - ◆ **ResultSet**: This interface provides methods for retrieving data returned by a SQL statement.
 - ◆ **ResultSetMetaData**: This interface is used to collect the metadata information associated with the last `ResultSet` object.
 - ◆ **Statement**: It is used to execute SQL statements and retrieve data into the `ResultSet`.



Following table describes some of the classes in this package:

Class Name	Description
Date	This class contains methods for performing conversion of SQL date formats to Java Date formats.
DriverManager	This class is used to handle loading and unloading of drivers and establish connection with the database.
DriverPropertyInfo	The methods in this class are used to retrieve or insert driver properties.
Time	This class provides formatting and parsing operations for time values.



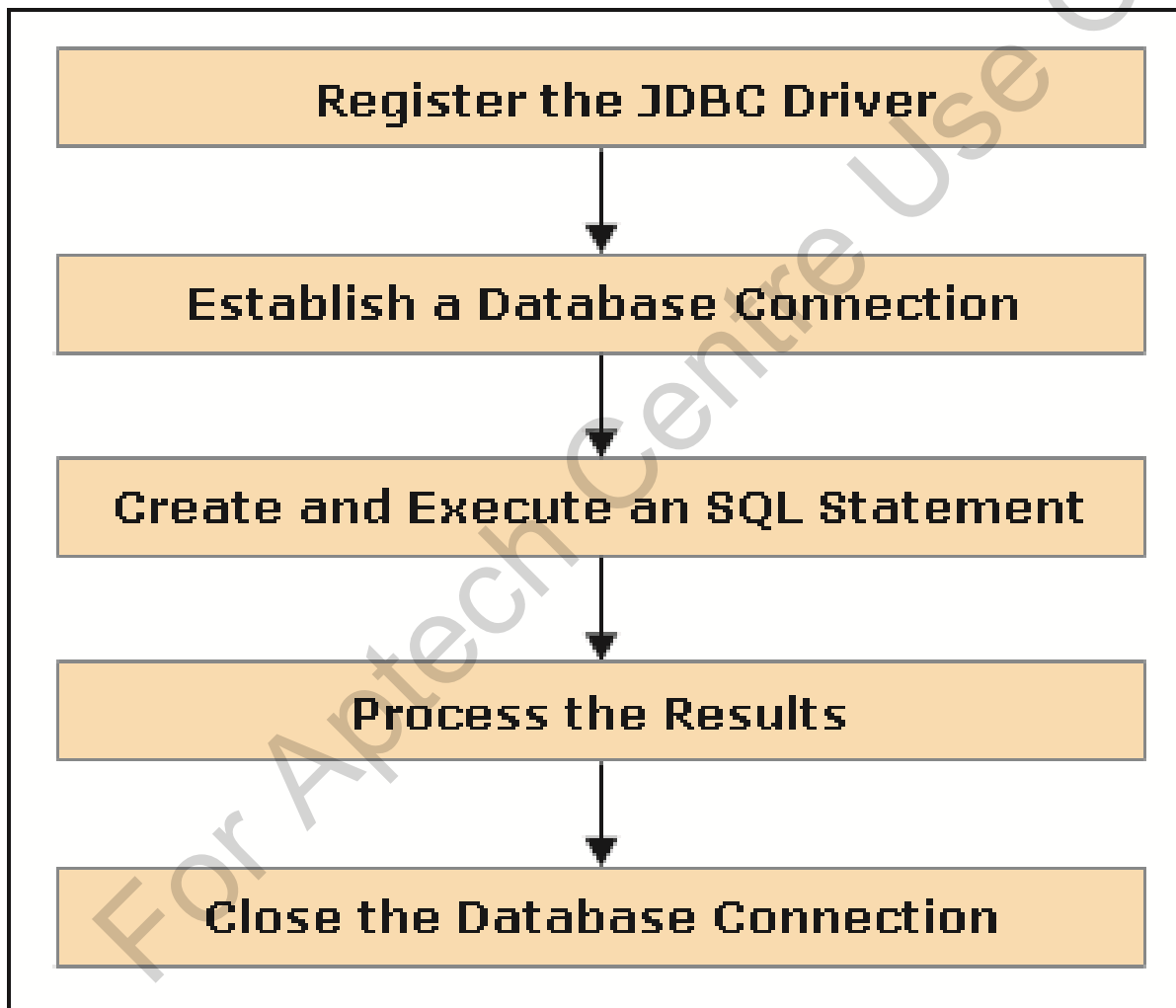
Exceptions defined by the package are listed in following table:

Exception	Description
<code>DataTruncation</code>	This exception is raised when a data value is truncated.
<code>SQLException</code>	This exception provides information on database access errors or other errors.
<code>SQLWarning</code>	This exception provides information on database access warnings.
<code>BatchUpdateException</code>	This exception is raised when an error occurs during batch update operation.

Steps to Develop a JDBC Application



- ◆ Figure displays the steps to develop a JDBC application.





- ◆ Load the driver class by using the `Class.forName()` method.
- ◆ Invoking the `Class.forName()` method creates an instance of the specified driver class and registers it with the `DriverManager` class.
- ◆ Once the driver is loaded successfully, the connection with a database can be established.

Syntax

```
Class.forName(<protocol>)
```



- ◆ The connection is established by using the `DriverManager.getConnection()` method.
- ◆ The method checks for the following:
 - ◆ All the available drivers for the eligibility to make a connection.
 - ◆ A driver that recognizes the URL, sent by the client.
- ◆ Following elements are used to establish a connection:
 - ◆ `Connection URL`
 - ◆ `DriverManager.getConnection()`

For Aptech Centre Use Only



- ◆ A Statement object must be created for query execution.
- ◆ The Statement object is created by using the `Connection.createStatement()` method.
- ◆ A Statement object can be classified into three categories based on the type of SQL statements sent to the database.
- ◆ Statement and PreparedStatement is inherited from Statement interface.
- ◆ The CallableStatement is inherited from the PreparedStatement interface and executes a call to stored procedure.
- ◆ A PreparedStatement object executes a precompiled SQL statement with or without IN parameters.

Syntax

```
public Statement createStatement() throws SQLException
```

Code Snippet

```
Connection cn = DriverManager.getConnection("jdbc:odbc:demo", "sa",  
    "playware");  
Statement st = cn.createStatement();
```



- ◆ The `executeQuery()` method retrieves information from the database.
- ◆ It accepts a simple SQL `SELECT` statement as a parameter and returns the database rows in form of a `ResultSet` object.
- ◆ The database query execution is based on the following:
 - ◆ **`executeQuery()`** : This method executes any SQL statement with a "SELECT" clause, that returns the result of the query as a result set.
 - ◆ **`ResultSet` object**: `ResultSet` objects receive and store the data in the same form as it is returned from the SQL queries. The `ResultSet` object is generally created by using the `executeQuery()` method.

Using executeQuery() and ResultSet Objects [2-2]



Syntax

```
public ResultSet executeUpdate(String sql) throws  
SQLException
```

Code Snippet

```
ResultSet rs = st.executeQuery("SELECT * FROM  
Department");
```

For Aptech Centre Use Only

Using executeUpdate() and execute() Methods



- ◆ The `executeUpdate()` method executes INSERT, DELETE, UPDATE, and other SQL DDL (Data Definition Language) such as CREATE TABLE, DROP TABLE, and so on.
- ◆ The method returns an integer value indicating the row count.

Syntax

```
public int executeUpdate(String sql) throws  
SQLException
```

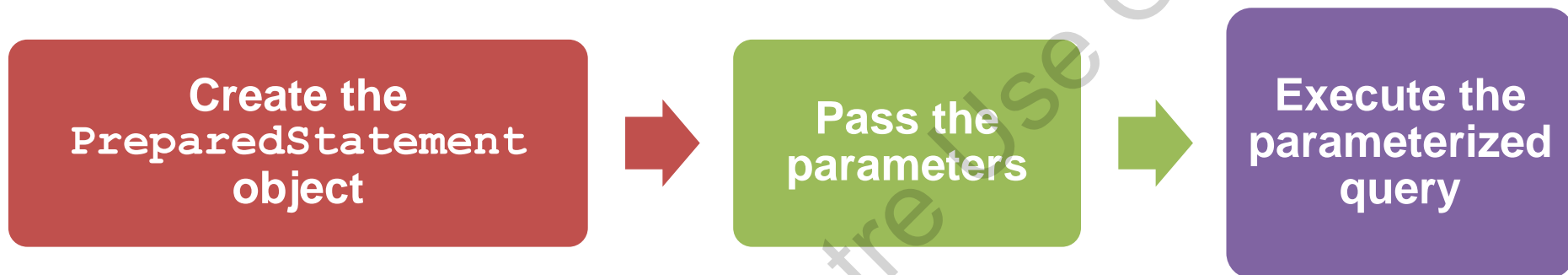
- ◆ The `execute()` method executes SQL statements that returns more than one result set.
- ◆ The method returns true if a result set object is generated by the SQL statements.

Syntax

```
public boolean execute (String sql) throws  
SQLException
```



- ◆ Figure shows the steps to create parameterized queries:



- ◆ Code Snippet demonstrates the steps to create parameterized query.

Code Snippet

```
. . .  
// Create PreparedStatement object  
String sqlStmt = "UPDATE Employees SET Dept_ID = ? WHERE Dept_Name LIKE ?";  
PreparedStatement pStmt = cn.prepareStatement(sqlStmt);  
  
// Passing parameters  
pStmt.setInt(1, 25);  
pStmt.setString(2, "Production");  
  
// Executes the executeUpdate() method  
pStmt.executeUpdate();
```



- ◆ Commonly occurring exceptions during database handling with JDBC include the following:
 - ◆ `ClassNotFoundException`
 - ◆ `SQLException`
 - `ClassNotFoundException`
 - ◆ While loading a driver using `Class.forName()`, if the class for the specified driver is not present in the package, then the method invocation throws a `ClassNotFoundException`.
 - ◆ The method should be wrapped inside a try block, followed by a catch block, which deals with the thrown exception (if any).
 - ◆ Code Snippet shows use of the `ClassNotFoundException`.

Code Snippet

```
try {  
    Class.forName("SpecifiedDriverClassName");  
} catch (java.lang.ClassNotFoundException e) {  
    System.err.print("ClassNotFoundException: ");  
    System.err.println(e.getMessage());  
}
```



- SQLException
 - ◆ Every method defined by the JDBC objects can throw `java.sql.SQLException`.
 - ◆ Hence, whenever, these methods are used, they should be wrapped inside a try/catch block to handle the exceptions.
 - ◆ Code Snippet shows the use of `SQLException`.

Code Snippet

```
try {  
    // Code that could generate an exception goes here.  
    /* If an exception is generated, the catch block below will print  
    out information about it. */  
}  
catch (SQLException ex)  
{  
    System.err.println("SQLException: " + ex.getMessage());  
    System.out.println("ErrorCode: " + ex.getErrorCode());  
}
```

Necessity for Processing Queries [1-2]



- ◆ Once the database query is executed and `ResultSet` object is created, the next step is to process and retrieve the result from the `ResultSet`.
- ◆ As the data in the `ResultSet` is in a tabular format and the cursor is positioned before the first row, it must traverse the rows using the `next()` method.
- ◆ The `next()` method allows to traverse forward by moving the cursor one row forward.
- ◆ It returns a boolean `true` if the current cursor position is on a valid row and returns a `false` when the cursor is placed at a position after the last row.

For Aptechnology Centre Use Only



- ◆ Code Snippet demonstrates use of the `next()` method.

Code Snippet

```
ResultSet rs1 = st1.executeQuery("SELECT Employee_Name  
FROM Employees");  
while (rs1.next()) {  
    String name=rs1.getString("Employee_Name");  
    System.out.println(name);  
}
```

- ◆ Here, the code retrieves the employee name from the Employee table and displays them in a sequence using the `next()` method.



getString()

Is used for retrieving a string value (SQL type VARCHAR) and assigning into Java `String` object.

getInt()

Is used for retrieving an integer value from the current row and assigning into a Java integer variable.

getFloat()

Is used for retrieving a float value from a current row and assigning into a Java float variable.

getObject()

Is used for retrieving the value from a current row as an object and assigning to `Object` in Java programming language.



- ◆ Use the `close()` method to close the database connection.
- ◆ To keep the database-releasing methods tidy and always released (even though an exception may be thrown), database connections should be closed within a `finally` block.

Syntax

```
public void close()
```

Code Snippet

```
st1.close();  
cn1.close();
```

- ◆ The `Statement` and the `Connection` object have been closed using the `close()` method.



- ◆ The dictionary meaning of metadata is data about data.
- ◆ In the context of databases, it can also be defined as information that defines the structure and properties of the data stored in a database.
- ◆ JDBC support the access of metadata by providing several methods.
- ◆ For example, a table in a database has its defined name, column names, datatypes for its columns and the owner for the table, this description is known as the metadata.

For Aptechnology Centre USA Only



- ◆ The information that describes about the database itself is known as the database metadata.
- ◆ This metadata information is stored by the objects of DatabaseMetaData interface in the java.sql package.
- ◆ To create a DatabaseMetaData object, a new instance of the DatabaseMetaData interface is created.
- ◆ This newly created object is assigned with the results fetched by the `getMetaData()` method, called by the `Connection` object.

Code Snippet

```
DatabaseMetaData dmd = cn.getMetaData();
```

When a Java application gets a valid connection, this code creates a metadata object.



Category	Method Name	Description
Escape Characters	<code>getSearchStringEscape()</code>	Retrieves and returns the string to escape wildcard characters.
Database Information	<code>getDatabaseProductName()</code>	Retrieves the name of the connected database product and returns the same as a string.
	<code>getDatabaseProductVersion()</code>	Retrieves the version number of the connected database product and returns the same in a string format.
	<code>isReadOnly()</code>	Returns a boolean value of true if the database is in read-only form, else returns a false.
Driver Information	<code>getDriverName()</code>	Retrieves the name of the currently used JDBC driver and returns the same as a string.
	<code>getDriverVersion()</code>	Retrieves the version number of the currently used JDBC driver and returns the same as a string.
	<code>getDriverMajorVersion()</code>	Retrieves the major version number of the currently used JDBC driver and returns the same as an integer.



- ◆ Following table describes some more methods:

Method Name	Description
<code>getURL()</code>	This method returns the URL for this DBMS as a String.
<code>getUserName()</code>	This method returns the user name as known to the database in the form of a String object.
<code>getConnection()</code>	This method returns the connection object that produced this metadata object.
<code>supportsPositionedDelete()</code>	This method returns a boolean value of true or false depending on whether the database supports positioned DELETE statements.
<code>supportsPositionedUpdate()</code>	This method returns a boolean value of true or false depending on whether the database supports positioned UPDATE statements.
<code>supportsTransactions()</code>	This method returns whether the database supports transactions and returns a boolean value of true if transaction is supported, otherwise returns false.



- ◆ The information that describes about the data contained in a `ResultSet` is known as the `ResultSet` metadata.
- ◆ This metadata information is stored by the objects of `ResultSetMetaData` interface in the `java.sql` package.
- ◆ This object can give the information about attributes like column names, number of columns, and data types for the columns in the result set.
- ◆ To retrieve all these information, a `ResultSetMetaData` object is created and assigned with the results fetched by the `getMetaData()` method, called by the `ResultSet` object.

Code Snippet

```
ResultSetMetaData rmd = rs.getMetaData();
```



- ◆ `getColumnName()`
 - ◆ Retrieves the name of the specified column and returns the same as a `String`.
 - ◆ **Example:** `String colName = rmd1.getColumnName(2);`
- ◆ `getColumnCount()`
 - ◆ Retrieves and returns the number of columns as an integer in the current `ResultSet` object.
 - ◆ **Example:** `int totalCols = rmd1.getColumnCount();`
- ◆ `getColumnType()`
 - ◆ Retrieves the SQL type of the specified column and returns the same as a `String`.
 - ◆ **Example:** `String colType = rmd1.getColumnType();`



- ◆ SQL Server 2019 is a widely used database from Microsoft.
- ◆ It allows the user to manipulate data with a large number of data types, stored procedures, and offers a secure platform for developing enterprise database applications.
- ◆ Java applications can be connected to SQL Server 2019 through Type 4 JDBC Driver.

Syntax

```
jdbc:sqlserver://serverName;instanceName:portNumber;property=value  
[;property =value]
```

Code Snippet

```
jdbc:sqlserver://localhost;user=Author;password=*****;
```



- ◆ JDBC is a software API to access database connectivity for Java applications.
- ◆ JDBC offers four different types of drivers to connect to the databases.
- ◆ The `java.sql` package offers classes that sets up a connection with databases, sends the SQL queries to the databases and retrieving the computed results.
- ◆ The `Statement` object is used to send and execute the SQL queries and the `ResultSet` object is used to retrieve the computed data rows with the help of methods.
- ◆ SQL Server 2019 is a popular database from Microsoft.
- ◆ The information that describes the data in database is known as the metadata.
- ◆ The `java.sql` package mainly defines the `DatabaseMetaData` interfaces to access metadata for the database itself and the `ResultSetMetaData` interface to access the different columns in a result set.