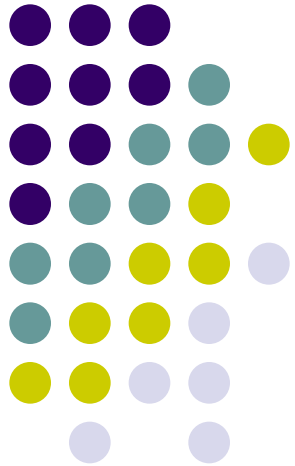
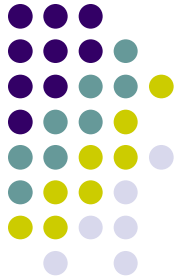
JDBC –
Java DataBase Connectivity

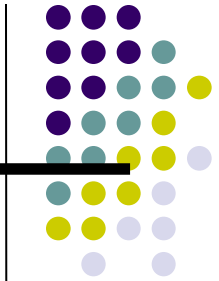


What is JDBC?



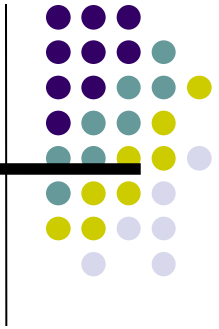
- "An API that lets you access virtually any tabular data source from the Java programming language"
  - JDBC Data Access API – JDBC Technology Homepage
  - What's a tabular data source?
- "… access virtually any data source, from relational databases to spreadsheets and flat files."
  - JDBC Documentation

What is JDBC?



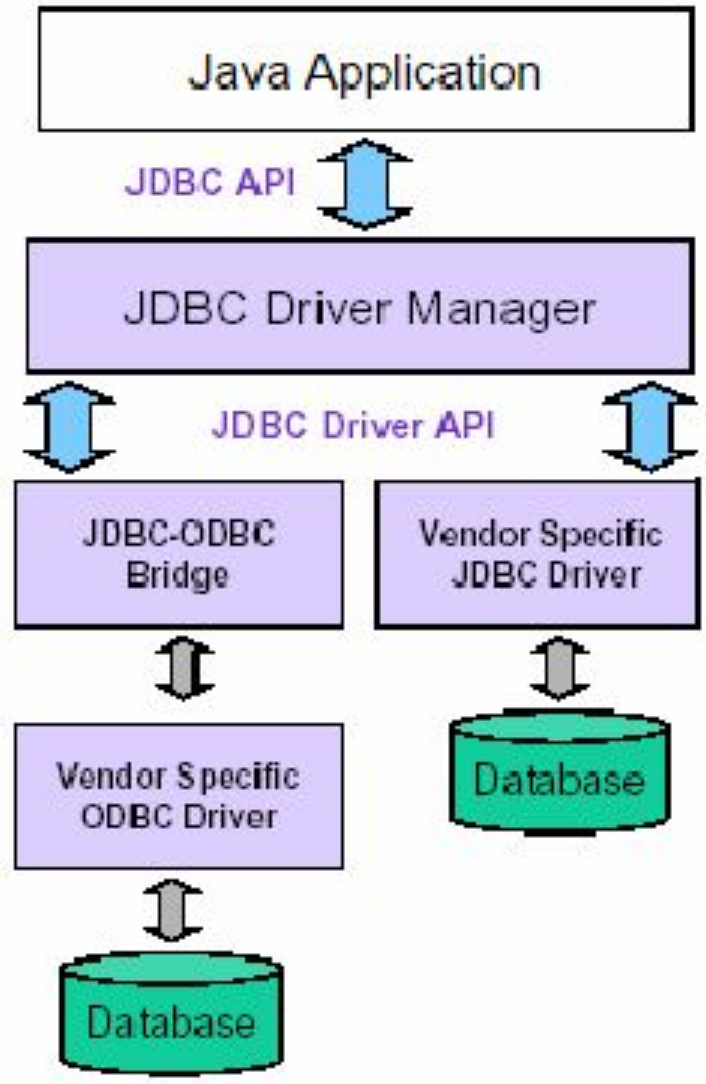
- JDBC provides Java applications with access to most database systems via SQL
- The architecture and API closely resemble Microsoft's ODBC
- JDBC 1.0 was originally introduced into Java 1.1
  - JDBC 2.0 was added to Java 1.2
- JDBC is based on SQL-92
- JDBC classes are contained within the java.sql package
  - There are few classes
  - There are several interfaces

Database Connectivity History



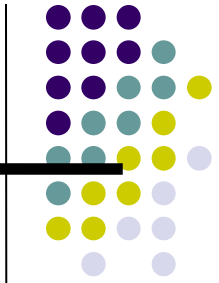
- Before APIs like JDBC and ODBC, database connectivity was tedious
  - Each database vendor provided a function library for accessing their database
  - The connectivity library was proprietary.
  - If the database vendor changed for the application, the data access portions had to be rewritten
  - If the application was poorly structured, rewriting its data access might involve rewriting the majority of the application
  - The costs incurred generally meant that application developers were stuck with a particular database product for a given application

General Architecture



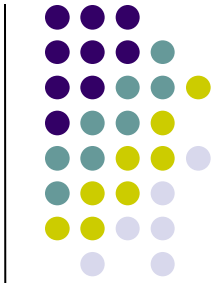
- Why is this architecture multi-tiered?

JDBC Drivers



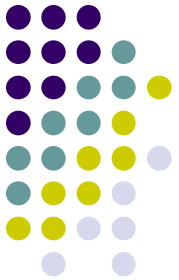
- There are 4 types of JDBC Drivers
  - Type 1 - JDBC-ODBC Bridge
  - Type 2 - JDBC-Native Bridge
  - Type 3 - JDBC-Net Bridge
  - Type 4 - Direct JDBC Driver
- Type 1 only runs on platforms where ODBC is available
  - ODBC must be configured separately
- Type 2 Drivers map between a proprietary Database API and the JDBC API
- Type 3 Drivers are used with middleware products
- Type 4 Drivers are written in Java
  - In most cases, type 4 drivers are preferred

Basic steps to use a database in Java



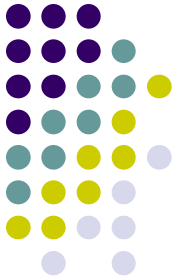
- 1.Establish a **connection**
- 2.Create JDBC **Statements**
- 3.Execute **SQL** Statements
- 4.GET **ResultSet**
- 5.**Close** connections

1. Establish a connection



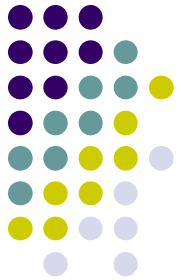
- **import java.sql.*;**
- **Load the vendor specific driver**
  - Class.forName("oracle.jdbc.driver.OracleDriver");
    - What do you think this statement does, and how?
    - Dynamically loads a driver class, for Oracle database
- **Make the connection**
  - Connection con = DriverManager.getConnection("jdbc:oracle:thin:@oracle-prod:1521:OPROD", username, passwd);
    - What do you think this statement does?
    - Establishes connection to database by obtaining a *Connection* object

2. Create JDBC statement(s)



- Statement stmt = con.createStatement() ;
- Creates a Statement object for sending SQL statements to the database

Executing SQL Statements



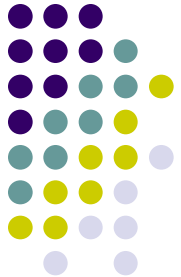
- String createLehigh = "Create table Lehigh " +
  "(SSN Integer not null, Name VARCHAR(32), " + "Marks Integer)";
  stmt.**executeUpdate**(createLehigh);
  //What does this statement do?

- String insertLehigh = "Insert into Lehigh values“ + "(123456789,abc,100)";
  stmt.**executeUpdate**(insertLehigh);

Get ResultSet

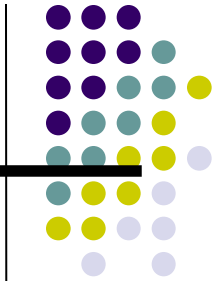
```
String queryLehigh = "select * from Lehigh";

ResultSet rs = Stmt.executeQuery(queryLehigh);
//What does this statement do?

while (rs.next()) {
    int ssn = rs.getInt("SSN");
    String name = rs.getString("NAME");
    int marks = rs.getInt("MARKS");
}
```

Close connection



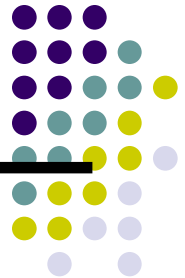
- stmt.close();
- con.close();

JDBC Interfaces



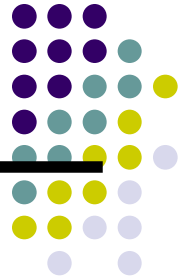
- Driver
  - All JDBC Drivers must implement the Driver interface. Used to obtain a connection to a specific database type
- Connection
  - Represents a connection to a specific database
  - Used for creating statements
  - Used for managing database transactions
  - Used for accessing stored procedures
  - Used for creating callable statements
- Statement
  - Used for executing SQL statements against the database

JDBC Interfaces



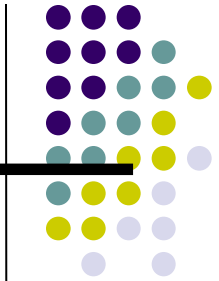
- ResultSet
  - Represents the result of an SQL statement
  - Provides methods for navigating through the resulting data
- PreparedStatement
  - Similar to a stored procedure
  - An SQL statement (which can contain parameters) is compiled and stored in the database
- CallableStatement
  - Used for executing stored procedures
- DatabaseMetaData
  - Provides access to a database's system catalogue
- ResultSetMetaData
  - Provides information about the data contained within a ResultSet

Using JDBC



- To execute a statement against a database, the following flow is observed
  - Load the driver (Only performed once)
  - Obtain a Connection to the database (Save for later use)
  - Obtain a Statement object from the Connection
  - Use the Statement object to execute SQL. Updates, inserts and deletes return Boolean. Selects return a ResultSet
  - Navigate ResultSet, using data as required
  - Close ResultSet
  - Close Statement
- Do NOT close the connection
  - The same connection object can be used to create further statements
  - A Connection may only have one active Statement at a time. Do not forget to close the statement when it is no longer needed.
  - Close the connection when you no longer need to access the database

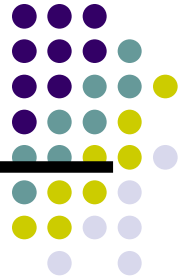
Loading Drivers



- Even a good API can have problems
  - Loading drivers fits into this category
- The DriverManager is a singleton
- Each JDBC Driver is also a singleton
- When a JDBC Driver class is loaded, it must create an instance of itself and register that instance with the JDBC DriverManager
- How does one load a "class" into the Virtual machine?
  - Use the static method Class.forName()

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Connecting to a Database

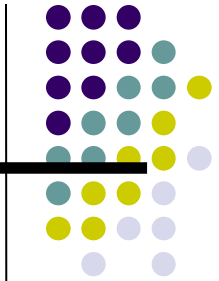


- Once a Driver is loaded, a connection can be made to the database
- The connection is defined by URL
  - The URL has the following form:
    jdbc:*driver:databasename*
  - Examples:
    jdbc:odbc*:*MyOdbcDatabase
    jdbc*:*postgres*:*WebsiteDatabase
    jdbc*:*oracle*:*CustomerInfo
- A connection is obtained in the following manner:

```
Connection aConnection =
   DriverManager.getConnection("jdbc:odbc:myDatabase");
```

- Overloaded versions of the getConnection method allow the specification of a username and password for authentication with the database.

Using a Connection

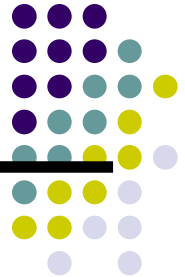


- The Connection interface defines many methods for managing and using a connection to the database

```
public Statement createStatement()
public PreparedStatement prepareStatement(String sql)
public void setAutoCommit(boolean)
public void commit()
public void rollback()
public void close()
```

- The most commonly used method is createStatement()
  - When an SQL statement is to be issued against the database, a Statement object must be created through the Connection

Using a Statement

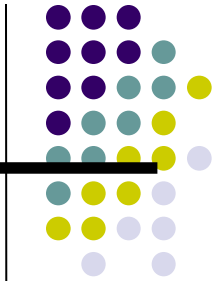


- The Statement interface defines two methods for executing SQL against the database

```
public ResultSet executeQuery(String sql)
public int executeUpdate(String sql)
```

- executeQuery returns a ResultSet
  - All rows and columns which match the query are contained within the ResultSet
  - The developer navigates through the ResultSet and uses the data as required.
- executeUpdate returns the number of rows changed by the update statement
  - This is used for insert statements, update statements and delete statements

Using a ResultSet



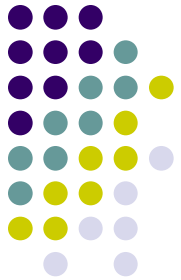
- The ResultSet interface defines many navigation methods

```
public boolean first()
public boolean last()
public boolean next()
public boolean previous()
```

- The ResultSet interface also defines data access methods

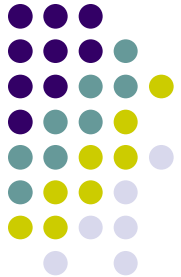
```
public int getInt(int columnNumber)      -- Note: Columns are
   numbered
public int getInt(String columnName)    -- from 1 (not 0)
public long getLong(int columnNumber)
public long getLong(String columnName)
public String getString(int columnNumber)
public String getString(String columnName)
```

Transactions and JDBC



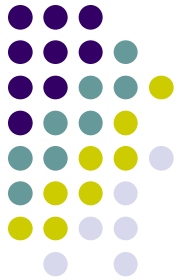
- JDBC allows SQL statements to be grouped together into a single transaction
- Transaction control is performed by the Connection object, default mode is auto-commit, I.e., each sql statement is treated as a transaction
- We can turn off the auto-commit mode with con.setAutoCommit(false);
- And turn it back on with con.setAutoCommit(true);
- Once auto-commit is off, no SQL statement will be committed until an explicit is invoked con.commit();
- At this point all changes done by the SQL statements will be made permanent in the database.

Handling Errors with Exceptions



- Programs should recover and leave the database in a consistent state.
- If a statement in the try block throws an exception or warning, it can be caught in one of the corresponding catch statements
- How might a finally {…} block be helpful here?
- E.g., you could rollback your transaction in a catch { …} block or close database connection and free database related resources in finally {…} block

JDBC references



- JDBC Data Access API – JDBC Technology Homepage
  - http://java.sun.com/products/jdbc/index.html
- JDBC Database Access – The Java Tutorial
  - http://java.sun.com/docs/books/tutorial/jdbc/index.html
- JDBC Documentation
  - http://java.sun.com/j2se/1.4.2/docs/guide/jdbc/index.html
- java.sql package
  - http://java.sun.com/j2se/1.4.2/docs/api/java/sql/package-summary.html
- JDBC Technology Guide: Getting Started
  - http://java.sun.com/j2se/1.4.2/docs/guide/jdbc/getstart/GettingStartedTOC.fm.html
- JDBC API Tutorial and Reference (book)
  - http://java.sun.com/docs/books/jdbc/