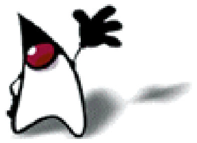
Accessing Databases with JDBC



OBJECTIVES

In this chapter you will learn:

- Relational database concepts
- To use Structured Query Language (SQL) to retrieve data from and manipulate data in a database
- To use the JDBC™ API of package `java.sql` to access databases

Introduction

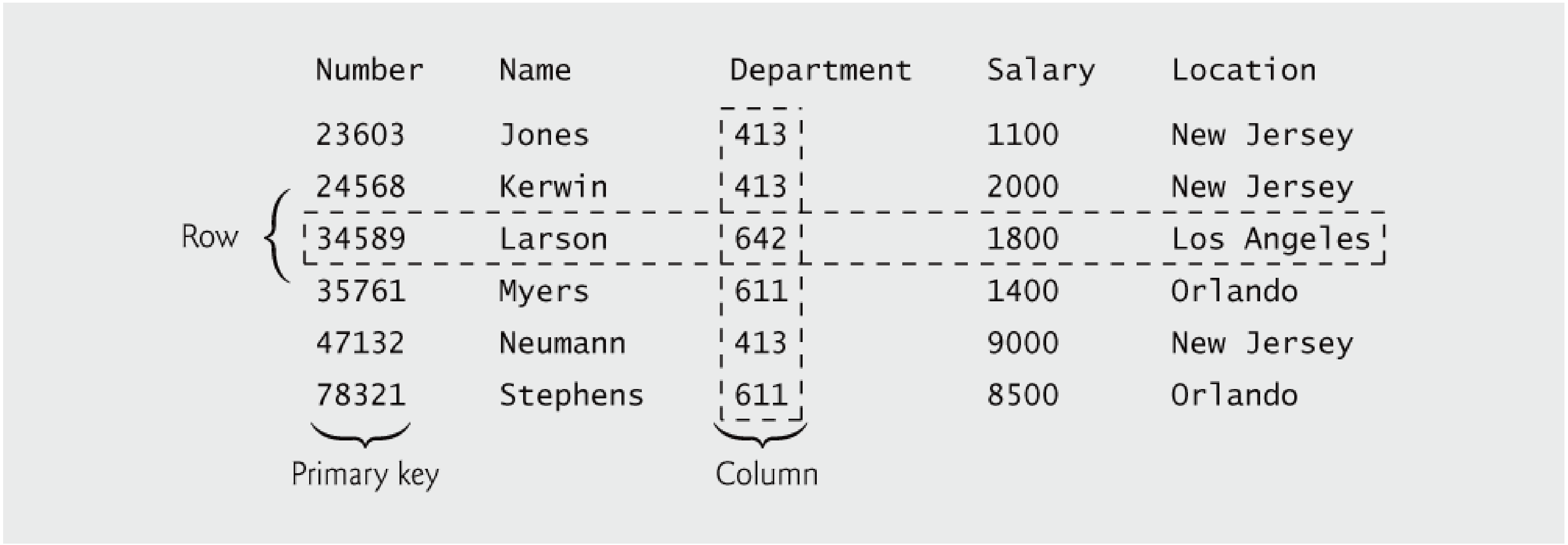
- Database
  - Collection of data
- DBMS
  - Database management system
  - Storing and organizing data
- SQL
  - Relational database
  - Structured Query Language

Introduction (Cont.)

- RDBMS
  - Relational database management system
  - MySQL
    - Open source
    - Available for both Windows and Linux
    - `dev.mysql.com/downloads/mysql/5.0.hml`
- JDBC
  - Java Database Connectivity
  - JDBC driver
    - Enable Java applications to connect to database
    - Enable programmers to manipulate databases using JDBC

Relational Databases

- Relational database
  - Table
    - Rows, columns
  - Primary key
    - Unique data
- SQL queries
  - Specify which data to select from a table


Row
Primary key
Column

Number	Name	Department	Salary	Location
23603	Jones	413	1100	New Jersey
24568	Kerwin	413	2000	New Jersey
34589	Larson	642	1800	Los Angeles
35761	Myers	611	1400	Orlando
47132	Neumann	413	9000	New Jersey
78321	Stephens	611	8500	Orlando

`Employee` table sample data.

Relational Database Overview: The `books` Database

- Sample `books` database
  - Tables
    - `authors`
      - `authorID`, `firstName`, `lastName`
    - `titles`
      - `isbn`, `title`, `editionNumber`, `copyright`, `publisherID`, `imageFile`, `price`

Column	Description
`authorID`	Author’s ID number in the database. In the `books` database, this integer column is defined as **autoincremented**—for each row inserted in this table, the `authorID` value is increased by 1 automatically to ensure that each row has a unique `authorID`. This column represents the table’s primary key.
`firstName`	Author’s first name (a string).
`lastName`	Author’s last name (a string).

`authors` table from the books database

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Andrew	Goldberg
4	David	Choffnes

Sample data from the authors table

Column	Description
`isbn`	ISBN of the book (a string). The table's primary key. ISBN is an abbreviation for "International Standard Book Number"—a numbering scheme that publishers use to give every book a unique identification number.
`title`	Title of the book (a string).
`editionNumber`	Edition number of the book (an integer).
`copyright`	Copyright year of the book (a string).

`titles` table from the `books` database

isbn	title	editionNumber	copyright
0131869000	Visual Basic How to Program	3	2006
0131525239	Visual C# How to Program	2	2006
0132222205	Java How to Program	7	2007
0131857576	C++ How to Program	5	2005
0132404168	C How to Program	5	2007
0131450913	Internet & World Wide Web How to Program	3	2004

Sample data from the titles table of the books database

SQL

- SQL keywords
  - SQL queries and statements

SQL keyword	Description
`SELECT`	Retrieves data from one or more tables.
`FROM`	Tables involved in the query. Required in every `SELECT`.
`WHERE`	Criteria for selection that determine the rows to be retrieved, deleted or updated. Optional in a SQL query or a SQL statement.
`GROUP BY`	Criteria for grouping rows. Optional in a `SELECT` query.
`ORDER BY`	Criteria for ordering rows. Optional in a `SELECT` query.
`INNER JOIN`	Merge rows from multiple tables.
`INSERT`	Insert rows into a specified table.
`UPDATE`	Update rows in a specified table.
`DELETE`	Delete rows from a specified table.

SQL query keywords

Basic SELECT Query

- Simplest format of a SELECT query
  - `SELECT * FROM` *tableName*
    - `SELECT * FROM authors`
- Select specific fields from a table
  - `SELECT authorID, lastName FROM authors`

authorID	lastName
1	Deitel
2	Deitel
3	Goldberg
4	Choffnes

Sample authorID and lastName data from the authors table

WHERE Clause

- specify the selection criteria
  - SELECT *columnName1*, *columnName2*, … FROM *tableName* WHERE *criteria*
    - SELECT title, editionNumber, copyright

      FROM titles

      WHERE copyright > 2002

title	editionNumber	copyright
Visual C# How to Program	2	2006
Visual Basic 2005 How to Program	3	2006
Java How to Program	7	2007
C How to Program	5	2007

Sampling of titles with copyrights after 2005 from table titles

WHERE Clause (Cont.)

- WHERE clause condition operators
  - <, >, <=, >=, =, <>
  - LIKE
    - wildcard characters % and _
    - SELECT authorID, firstName, lastName

      FROM authors

      WHERE lastName LIKE ‘D%’

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel

Authors whose last name starts with D from the authors table

ORDER BY Clause

- Optional ORDER BY clause
  - SELECT *columnName1*, *columnName2*, … FROM *tableName* ORDER BY *column* ASC

    -
      ```
      SELECT authorID, firstName, lastName
        FROM authors
        ORDER  BY lastName ASC
      ```
  - SELECT *columnName1*, *columnName2*, … FROM *tableName* ORDER BY *column* DESC

    -
      ```
      SELECT authorID, firstName, lastName
        FROM authors
        ORDER  BY lastName DESC
      ```

authorID	firstName	lastName
4	David	Choffnes
1	Harvey	Deitel
2	Paul	Deitel
3	Andrew	Goldberg

`Sample data from table authors in ascending order by lastName`

authorID	firstName	lastName
3	Andrew	Goldberg
1	Harvey	Deitel
2	Paul	Deitel
4	David	Choffnes

`Sample data from table authors in descending order by lastName`

INSERT Statement

- Insert a row into a table
  - `INSERT INTO` *tableName* (*columnName1*, … , *columnNameN*)

    `VALUES` (*value1*, … , *valueN*)
    - `INSERT INTO` authors (firstName, lastName)

      `VALUES` (‘Sue’, ‘Smith’)

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Andrew	Goldberg
4	David	Choffnes
5	Sue	Smith

Sample data from table Authors after an INSERT operation

UPDATE Statement

- Modify data in a table
  - UPDATE *tableName*

    SET *columnName1* = *value1*, … , *columnNameN* = *valueN*

    WHERE *criteria*
    - UPDATE authors

      SET lastName = ‘Jones’

      WHERE lastName = ‘Smith’ AND firstName = ‘Sue’

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Andrew	Goldberg
4	David	Choffnes
5	Sue	Jones

Sample data from table authors after an UPDATE operation

DELETE Statement

- Remove data from a table
  - DELETE FROM *tableName* WHERE *criteria*
    - DELETE FROM authors
      WHERE lastName = ‘Jones’ AND firstName = ‘Sue’

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Andrew	Goldberg
4	David	Choffnes

Sample data from table authors after a DELETE operation

JDBC

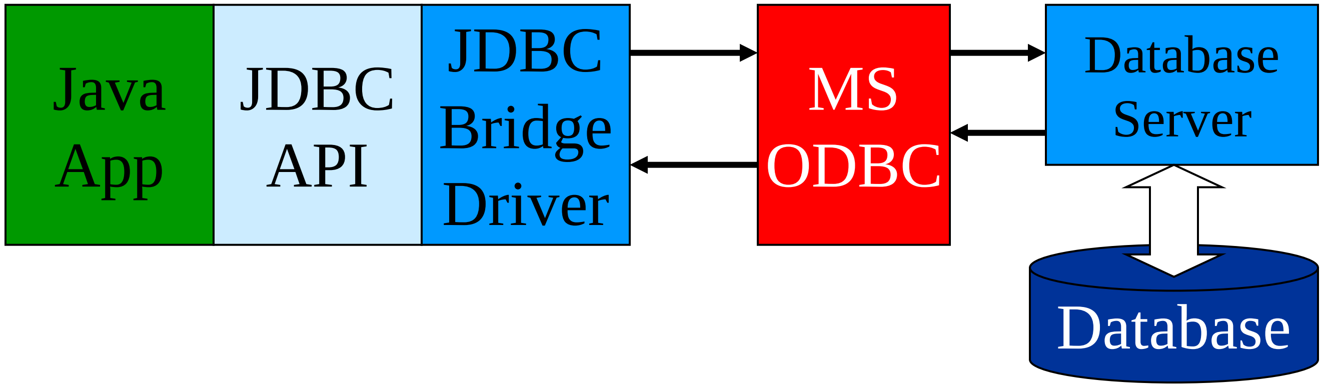
- JDBC is a Sun trademark
  - It is often taken to stand for Java Database Connectivity
- Java is very standardized, but there are many versions of SQL
- JDBC is a means of accessing SQL databases from Java
  - JDBC is a standardized API for use by Java programs
  - JDBC is also a specification for how third-party vendors should write database drivers to access specific SQL versions

Driver types

- There are four types of drivers:
  - **JDBC Type 1 Driver** -- JDBC/ODBC Bridge drivers
    - ODBC (Open DataBase Connectivity) is a standard software API designed to be independent of specific programming languages
    - Sun provides a JDBC/ODBC implementation
  - **JDBC Type 2 Driver** -- use platform-specific APIs for data access
  - **JDBC Type 3 Driver** -- 100% Java, use a net protocol to access a remote listener and map calls into vendor-specific calls
  - **JDBC Type 4 Driver** -- 100% Java
    - Most efficient of all driver types

JDBC Driver (1)

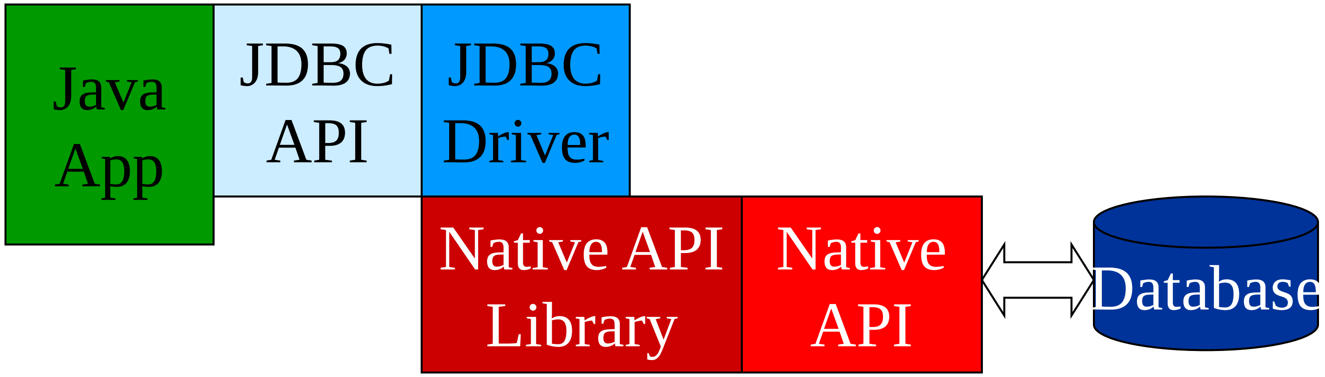
Type 1:
ODBC
Bridge



Type 1 JDBC drivers: ODBC Bridge: Drivers that implement the JDBC API as a mapping to another data access API, such as ODBC (Open Database Connectivity). Drivers of this type are generally dependent on a native library, which limits their portability. The JDBC-ODBC Bridge is an example of a Type 1 driver.

JDBC Driver (2)

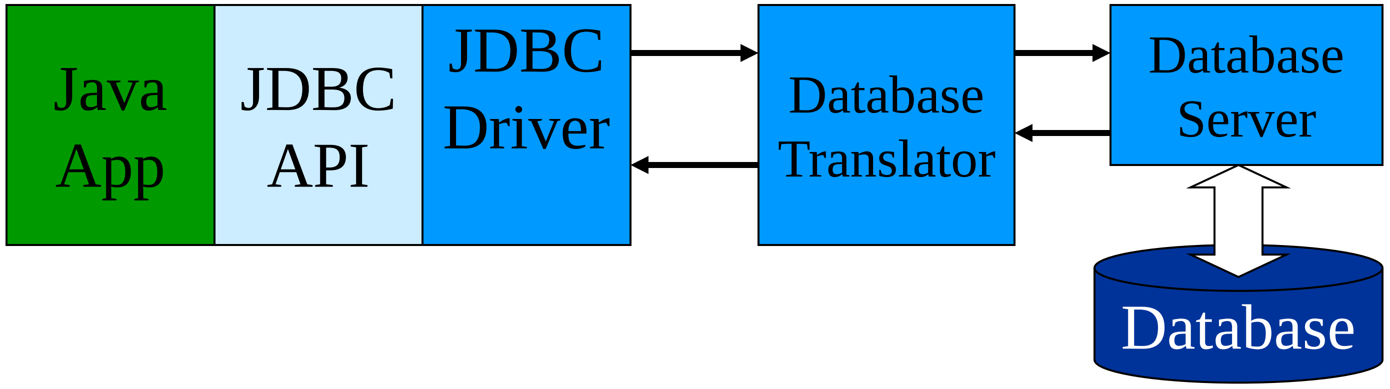
Type 2: Native DB API



Type 2 JDBC drivers: Native DB API: Drivers that are written partly in the Java programming language and partly in native code. These drivers use a native client library specific to the data source to which they connect. Again, because of the native code, their portability is limited. Oracle's OCI (Oracle Call Interface) client-side driver is an example of a Type 2 driver.

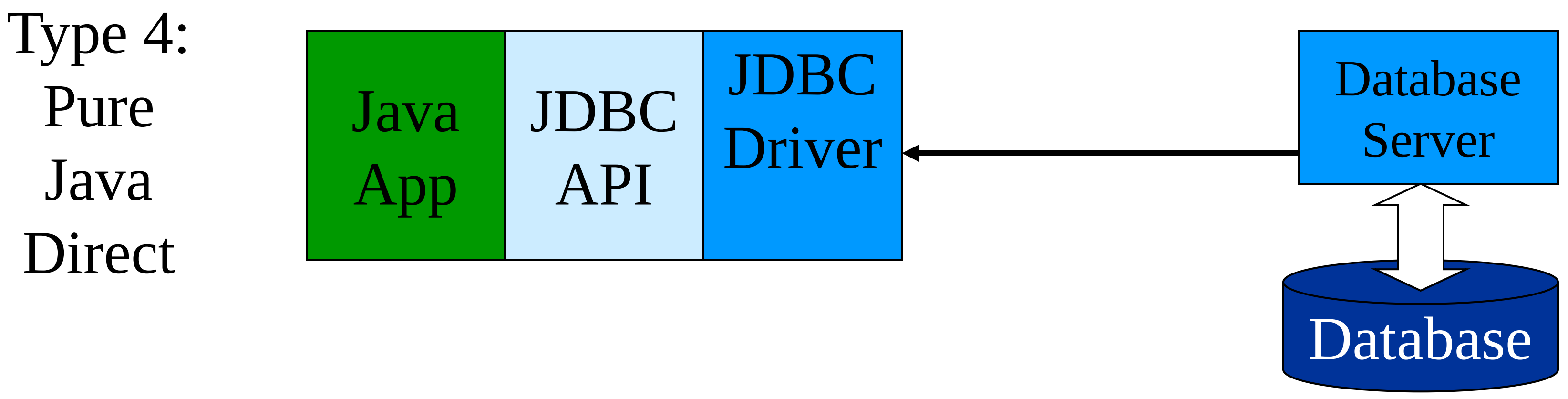
JDBC Driver (3)

Type 3:
Pure
Java



Type 3 JDBC drivers: Pure Java: Drivers that use a pure Java client and communicate with a middleware server using a database-independent protocol. The middleware server then communicates the client's requests to the data source.

JDBC Driver (4)



Type 4 JDBC drivers: Pure Java Direct: Drivers that are pure Java and implement the network protocol for a specific data source. The client connects directly to the data source.

JDBC Classes and Interfaces

Steps to using a database query:

- Load a JDBC “driver”
- Connect to the data source
- Send/execute SQL statements
- Process the results

Connector/J

- Connector/J is a JDBC Type 4 Driver for connecting Java to MySQL
- Installation is very simple:
  - Download the “Production Release” ZIP file from http://dev.mysql.com/downloads/connector/j/5.0.html
  - Unzip it
  - Put the JAR file where Java can find it
    - Add the JAR file to your CLASSPATH, or
    - In Eclipse: Project --> Properties --> Java Build Path --> Libraries --> Add External Jars…

Connecting to the server

- First, make sure the MySQL server is running
- In your program,

  -
    ```
    import java.sql.Connection;
    import java.sql.DriverManager;
    import java.sql.SQLException;
    ```
  - Register the JDBC driver,
    ```
    Class.forName("com.mysql.jdbc.Driver").newInstance();
    ```
  - Invoke the `getConnection()` method,
    ```
    Connection con =
        DriverManager.getConnection("jdbc:mysql:///myDB",
                                    myUserName,
                                    myPassword);
    ```
  - or `getConnection("jdbc:mysql:///myDB?user=dave&password=xxx")`

A complete program

-
  ```
  import java.sql.Connection;
  import java.sql.DriverManager;
  import java.sql.SQLException;

  public class JdbcExample1 {

      public static void main(String args[]) {
          Connection con = null;
          try {
              Class.forName("com.mysql.jdbc.Driver").newInstance();
              con = DriverManager.getConnection("jdbc:mysql:///test", "root", ”rootpswd");
              if (!con.isClosed())
                  System.out.println("Successfully connected to MySQL server...");
          } catch(Exception e) {
              System.err.println("Exception: " + e.getMessage());
          } finally {
              try {
                  if (con != null)
                      con.close();
              } catch(SQLException e) {}
          }
      }
  }
  ```

Using the Connection object

- public Statement createStatement()
  throws SQLException
  - Creates a Statement object for sending SQL statements to the database. SQL statements without parameters are normally executed using Statement objects.
  - The Statement object may be reused for many statements
- public PreparedStatement prepareStatement(String sql)
  throws SQLException
  - Creates a PreparedStatement object for sending parameterized SQL statements to the database.
  - A SQL statement with or without IN parameters can be pre-compiled and stored in a PreparedStatement object. This object can then be used to efficiently execute this statement multiple times.

Issuing queries

- The following are methods on the Statement object:
  - int executeUpdate() -- for issuing queries that modify the database and return no result set
    - Use for DROP TABLE, CREATE TABLE, and INSERT
    - Returns the number of rows in the resultant table
  - ResultSet executeQuery() -- for queries that do return a result set.
    - Returns results as a ResultSet object

Creating a table

- This example is from http://www.kitebird.com/articles/jdbc.html

-
  ```
  CREATE TABLE animal (
      id          INT UNSIGNED NOT NULL AUTO_INCREMENT,
      PRIMARY     KEY (id),
      name        CHAR(40),
      category    CHAR(40)
  )
  ```

-
  ```
  Statement s = conn.createStatement ();
  s.executeUpdate ("DROP TABLE IF EXISTS animal");
  s.executeUpdate (
          "CREATE TABLE animal ("
        + "id INT UNSIGNED NOT NULL AUTO_INCREMENT,"
        + "PRIMARY KEY (id),“
        + "name CHAR(40), category CHAR(40))");
  ```

Populating the table

```
int count;
count = s.executeUpdate (
          "INSERT INTO animal (name, category)"
          + " VALUES"
          + "('snake', 'reptile'),"
          + "('frog', 'amphibian'),"
          + "('tuna', 'fish'),"
          + "('racoon', 'mammal')");
  s.close ();
  System.out.println (count +
                    " rows were inserted");
```

ResultSet

- executeQuery() returns a ResultSet
  - ResultSet has a very large number of getXXX methods, such as
    - public String getString(String *columnName*)
    - public String getString(int *columnIndex*)
  - Results are returned from the current row
  - You can iterate over the rows:
    - public boolean next()
- ResultSet objects, like Statement objects, should be closed when you are done with them
  - public void close()

Example, continued

```
Statement s = conn.createStatement ();
s.executeQuery ("SELECT id, name, category " +
                        "FROM animal");
ResultSet rs = s.getResultSet ();
int count = 0;

    // Loop (next slide) goes here

rs.close ();
s.close ();
System.out.println (count + " rows were retrieved");
```

Example, continued

```
while (rs.next ()) {
      int idVal = rs.getInt ("id");
      String nameVal = rs.getString ("name");
      String catVal = rs.getString ("category");
      System.out.println (
              "id = " + idVal
              + ", name = " + nameVal
              + ", category = " + catVal);
      ++count;
  }
```

Prepared statements

- Prepared statements are precompiled, hence much more efficient to use

  -
    ```
    PreparedStatement s;
    s = conn.prepareStatement (
            "INSERT INTO animal (name, category VALUES(?,?)");
    s.setString (1, nameVal);
    s.setString (2, catVal);
    int count = s.executeUpdate ();
    s.close ();
    System.out.println (count + " rows were inserted");
    ```

Error handling

```
try {
    Statement s = conn.createStatement ();
    s.executeQuery ("XYZ"); // issue invalid query
    s.close ();
}
catch (SQLException e) {
    System.err.println ("Error message: "
                                    + e.getMessage ());
    System.err.println ("Error number: "
                                    + e.getErrorCode ());
}
```

The End