Object Oriented Programming

LESSON 08

Java Database Connectivity

Outline

Overview

In this chapter, you are going to learn about

- Know Exception
- Know how to create Exception
- Know how to use JDBC
- Know how to use JDBC with MySQL Database
- Know how to implement transaction

Learning content

1. Java Exception
   - Keyword finally
   - Keyword throw
   - The try-with-resources Statement
2. Declaring Interface
   - JDBC Architecture
   - Anatomy of Data Access
   - Basic steps to use a database
3. JDBC with MySQL Database
   - Establish a connection
   - Create JDBC statement(s)
   - Executing SQL Statements
4. Data Manipulation
   - Selecting data
   - Updating data
   - Removing data
5. Transactions and JDBC
   - About transaction
   - Transaction example

Pre-Test

Question	Possible answers	Correct Answer	Question Feedback
What happen when accessing to an index out of range of an array?	a) Error and crash application b) Error message is shown to user c) Program is not usable	a) Error and crash application b) Error message is shown to user	Program is usable as long we access index in range of array.
Can we prevent error from accessing index outside array's range?	a) Yes b) No	a) Yes	Test passed index before accessing its value by index.

1. Java Exception

- When executing Java code, different errors can occur:
  - coding errors made by the programmer,
  - errors due to wrong input, or
  - other unforeseeable things.
- When an error occurs, Java will normally stop and generate an error message. The technical term for this is: Java will throw an **exception** (throw an error).

1. Java Exception

- The try statement allows you to define a block of code to be tested for errors while it is being executed.
- The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.
- The try and catch keywords come in pairs:

```
try {
    // Block of code to try
} catch(Exception e) {
    // Block of code to handle errors
}
```

1. Java Exception

- Example:

```java
public class MyClass {
    public static void main(String[ ] args) {
        int[] myNumbers = {1, 2, 3};
        System.out.println(myNumbers[10]); // error coz upper bound is 2!
    }
}
```

- Output:

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 10
        at MyClass.main(MyClass.java:4)
```

1. Java Exception

- Example with try and catch:

```
public class MyClass {
    public static void main(String[ ] args) {
        try {
            int[] myNumbers = {1, 2, 3};
            System.out.println(myNumbers[10]);
        } catch (Exception e) {
            System.out.println("Something went wrong.");
        }
    }
}
```

- Output:

```
Something went wrong.
```

1.1. Keyword finally

- The finally statement lets you execute code, after try...catch, regardless of the result:

```
public class MyClass {
  public static void main(String[ ] args) {
    try {
      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[10]);
    } catch (Exception e) {
      System.out.println("Something went wrong.");
    } finally {
      System.out.println("The 'try catch' is finished.");
    }
  }
}
```

```
public class MyClass {
  public static void main(String[ ] args) {
    try {
      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[2]);
    } catch (Exception e) {
      System.out.println("Something went wrong.");
    } finally {
      System.out.println("The 'try catch' is finished.");
    }
  }
}
```

```
Something went wrong.
The 'try catch' is finished.
```

```
3
The 'try catch' is finished.
```

1.2. Keyword throw

- The throw statement allows you to create a custom error.
- The throw statement is used together with an exception type. There are many exception types available in Java:
  - ArithmeticException,
  - FileNotFoundException, ArrayIndexOutOfBoundsException,
  - SecurityException,
  - …

1.2. Keyword throw

- Example:
  Throw an exception:
  - If **age** is below **18** (print "Access denied").
  - If age is **18 or older**, print "Access granted":

```java
public class MyClass {
    static void checkAge(int age) {
        if (age < 18) {
            throw new ArithmeticException("Access denied - You must be at least 18 years old.");
        }
        else {
            System.out.println("Access granted - You are old enough!");
        }
    }

    public static void main(String[] args) {
        checkAge(15); // Set age to 15 (which is below 18...)
    }
}
```

```
Exception in thread "main" java.lang.ArithmeticException: Access denied - You must be at least
18 years old.
            at MyClass.checkAge(MyClass.java:4)
            at MyClass.main(MyClass.java:12)
```

1.3. The try-with-resources Statement

- The try-with-resources statement is a try statement that declares one or more resources. A resource is an object that must be closed after the program is finished with it.
- The try-with-resources statement ensures that each resource is closed at the end of the statement.
- Any object that implements java.lang.AutoCloseable, which includes all objects which implement java.io.Closeable, can be used as a resource.

1.3. The try-with-resources Statement

- The following example reads the first line from a file. It uses an instance of BufferedReader to read data from the file. BufferedReader is a resource that must be closed after the program is finished with it:

```
static String readFirstLineFromFile(String path) throws IOException {
    try (BufferedReader br = new BufferedReader(new FileReader(path))) {
        return br.readLine();
    }
}
```

1.3. The try-with-resources Statement

- You may declare one or more resources in a try-with-resources statement.
- The following example retrieves the names of the files packaged in the zip file zipFileName and creates a text file that contains the names of these files:

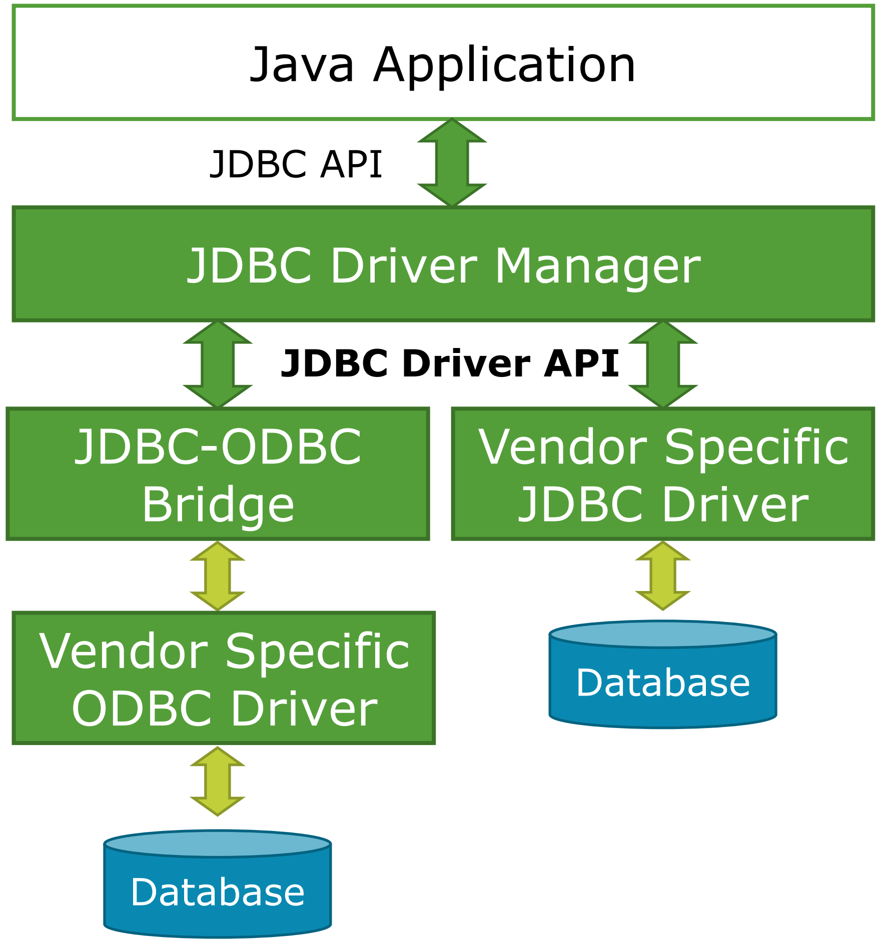
```java
public static void writeToFileZipFileContents(String zipFileName,
                                               String outputFileName)
        throws java.io.IOException {

    java.nio.charset.Charset charset =
            java.nio.charset.StandardCharsets.US_ASCII;
    java.nio.file.Path outputFilePath =
            java.nio.file.Paths.get(outputFileName);

    // Open zip file and create output file with
    // try-with-resources statement

    try (
            java.util.zip.ZipFile zf =
                    new java.util.zip.ZipFile(zipFileName);
            java.io.BufferedWriter writer =
                    java.nio.file.Files.newBufferedWriter(outputFilePath, charset)
    ) {
        // Enumerate each entry
        for (java.util.Enumeration entries =
             zf.entries(); entries.hasMoreElements();) {
            // Get the entry name and write it to the output file
            String newLine = System.getProperty("line.separator");
            String zipEntryName =
                    ((java.util.zip.ZipEntry)entries.nextElement()).getName() +
                            newLine;
            writer.write(zipEntryName, 0, zipEntryName.length());
        }
    }
}
```

2. JDBC

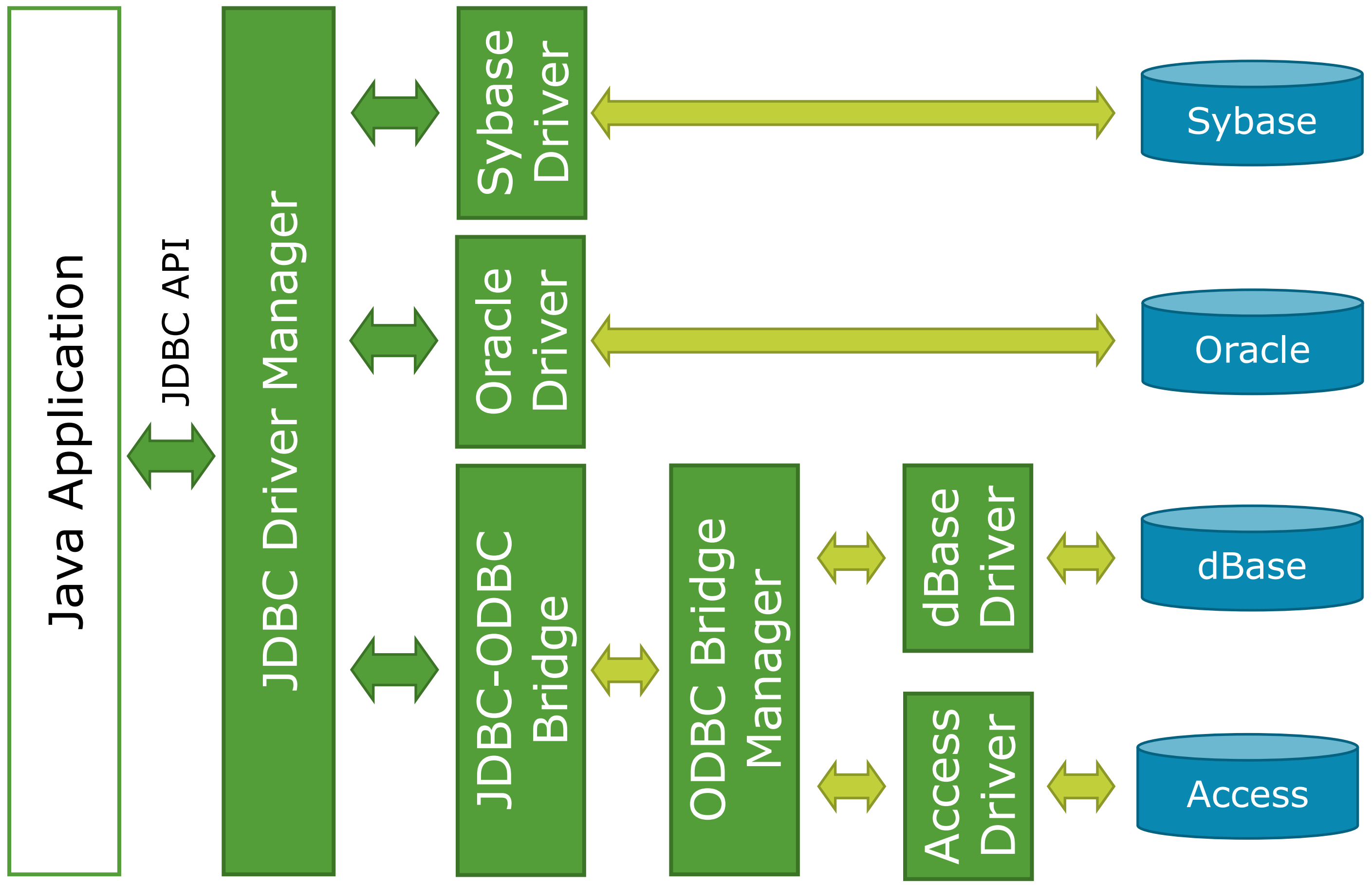
- "An API that lets you access virtually any tabular data source from the Java programming language"
    - JDBC Data Access API – JDBC Technology Homepage
  - What's an API?
    - See J2SE documentation
  - What's a tabular data source?
- "… access virtually any data source, from relational databases to spreadsheets and flat files."
  - JDBC Documentation
- We'll focus on accessing MySQL databases

2.1. JDBC Architecture

- What design pattern is implied in this architecture?
  - Bridge design pattern
- What does it buy for us?
  - One code supports multiple types of DB
  - Easily add supports for future DB types
- Why is this architecture also multi-tiered?
  - To isolate DB-related works and Business Logic
  - Make easier to code by focusing only Java code part



2.2. Anatomy of Data Access



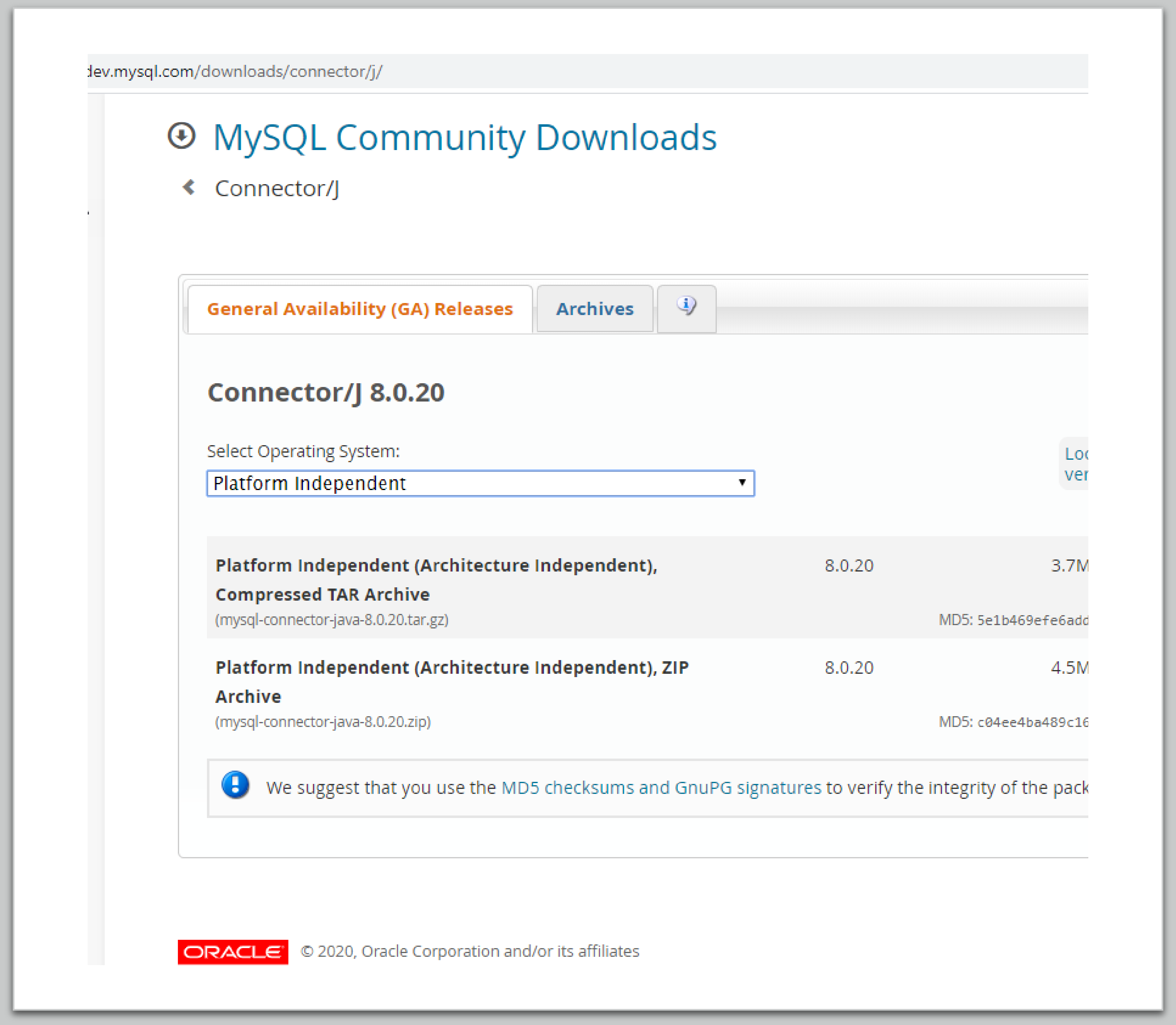
2.3. Basic steps to use a database

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

3. JDBC with MySQL Database

To connect to MySQL we need Connector/J that can be downloaded from:

https://dev.mysql.com/downloads/connector/j/


dev.mysql.com/downloads/connector/j/

MySQL Community Downloads

Connector/J

General Availability (GA) Releases | Archives

Connector/J 8.0.20

Select Operating System:

Platform Independent

Platform Independent (Architecture Independent), Compressed TAR Archive (mysql-connector-java-8.0.20.tar.gz)	8.0.20	3.7M MD5: 5e1b469efe6add
Platform Independent (Architecture Independent), ZIP Archive (mysql-connector-java-8.0.20.zip)	8.0.20	4.5M MD5: c04ee4ba489c16

We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the pack

ORACLE © 2020, Oracle Corporation and/or its affiliates

3.1. Establish a connection

- **import java.sql.*;**
- **Load the vendor specific driver**
  - Class.forName("com.mysql.cj.jdbc.Driver");
    - What do you think this statement does, and how?
      - ➢ Dynamically loads a driver class, for MySQL database
- **Make the connection**
  - Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/i4db?user=root&password=secret");
    - What do you think this statement does?
      - ➢ Establishes connection to database by obtaining a *Connection* object

3.2. Create JDBC statement(s)

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

3.3. Executing SQL Statements

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

- String insertStudent = "Insert into students values“ +
  "(e20226789,abc,100)";
  stmt.**executeUpdate**(insertStudent);

4. Data Manipulation

After connecting to DataBase we can:

- Select data
- Insert data
- Update data
- Delete data

4.1. Select data

```
String queryStudent = "select * from students";

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

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

4.2. Inserting data

```
String queryStudent = "insert into students(Name,Marks) values('Sophy',9)";

int affectedRowCount = Stmt.executeUpdate(queryStudent);
```

4.3. Deleting data

String queryStudent = "delete * from students where ID=1";

int affectedRowCount = Stmt.**executeUpdate**(queryStudent);

5. Transactions and JDBC

A database transaction symbolizes a unit of work performed within a database management system (or similar system) against a database, and treated in a coherent and reliable way independent of other transactions.

5.1. 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.

5.2. Transactions Example

```
conn.setAutoCommit(false);

Statement statement = conn.createStatement(); PreparedStatement psInsert =
        conn.prepareStatement("insert into students(Name,Marks) values('Sophy','9')");

psInsert.execute();

conn.commit();
```

Reference

- 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/