**데이터 목록 조회**

2019-10-25

이승진

**학습목표**

데이터베이스 연결 정보를 설정 방법

jdbc 프로그래밍 기술

데이터 목록을 조회하여 화면에 출력하는 기능을 구현한다.

**목차**

[1. 배경지식 2](#_Toc24488688)

[1) JDBC 객체 2](#_Toc24488689)

[2) DTO, DAO 2](#_Toc24488690)

[2. 데이터베이스 연결 설정 3](#_Toc24488691)

[1) jdbc 드라이버 파일 3](#_Toc24488692)

[2) context.xml 파일 생성 3](#_Toc24488693)

[3. 데이터 조회 Java 코드 6](#_Toc24488694)

[1) DB.java 6](#_Toc24488695)

[2) Student.java 7](#_Toc24488696)

[3) StudentDAO.java 8](#_Toc24488697)

[4. JSP 파일 구현 10](#_Toc24488698)

[1) jdbc1/studentList1.jsp 10](#_Toc24488699)

[2) 에러 12](#_Toc24488700)

[5. 연습문제 15](#_Toc24488701)

[1) User.java 구현 15](#_Toc24488702)

[2) UserDAO.java 구현 15](#_Toc24488703)

[3) jdbc1/userList.jsp 16](#_Toc24488704)

[6. 부록 - 고급 Java 문법 활용하기 17](#_Toc24488705)

[1) 자원 반납과 finally 17](#_Toc24488706)

[2) try with resource 문법과 자원 반납 18](#_Toc24488707)

[3) AutoCloseable 인터페이스 21](#_Toc24488708)

# 배경지식

## JDBC 객체

|  |  |
| --- | --- |
| Connection | 데이터베이스에 연결하기 위한 객체이다 |
| PreparedStatement | SQL 명령을 실행하기 위한 객체이다. |
| ResultSet | SQL 명령의 조회 결과 데이터를 전달해 주는 객체이다. |

작업이 끝난 후, Connection 객체의 close 메소드는 반드시 호출해 주어야 한다.

PreparedStatement, ResultSet 객체의 close 메소드도 가급적 호출해 주는 것이 좋다.

이 클래스들은 AutoCloseable 인터페이스를 implements 하였다.

## DTO, DAO

### Data Transfer Object

데이터를 채워서 전달하는 용도로 구현된 클래스

### Data Access Object

DB 테이블에 대한 SELECT,INSERT, UPDATE, DELETE 작업이 메소드로 구현된 클래스

### 클래스에 역할 한 개만 구현

바람직한 객체지향 프로그래밍 측면에서,

하나의 클래스에는, 하나의 역할로 요약될 수 있는 기능들만 구현되어 있어야 한다.

여러가지 역할들을 클래스 하나에 모아서 구현하는 것은 바람직하지 않다.

데이터를 채워서 전달하는 용도로 구현되는 DTO 클래스에는

오직 데이터를 넣고 꺼내기 위한 메소드만 구현되어야 한다.

DAO 클래스에는,

어떤 DB 테이블에 대한 SELECT,INSERT, UPDATE, DELETE 작업만 구현되어야 한다.

가급적 DB 테이블 하나 당 DAO 클래스 한 개를 구현하는 것이 바람직하다.

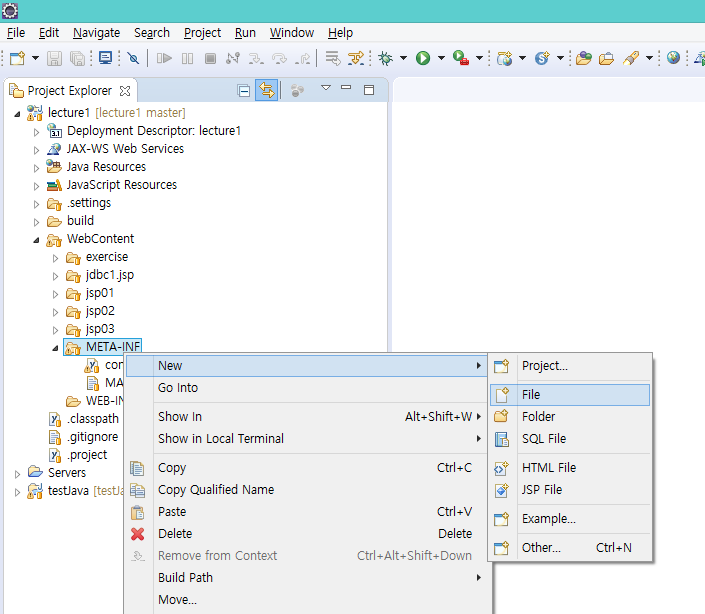
# 데이터베이스 연결 설정

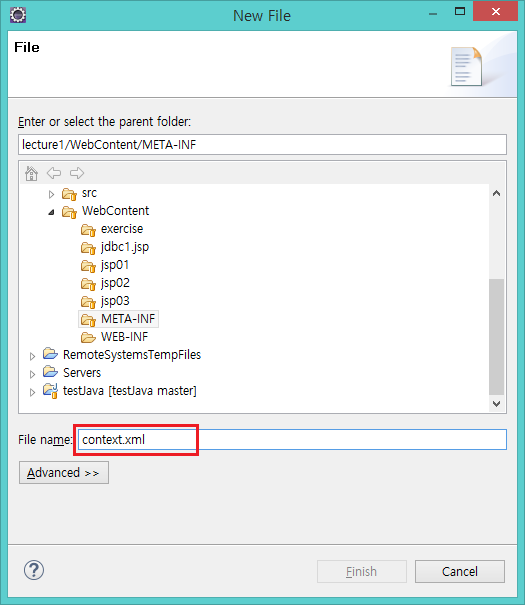
## jdbc 드라이버 파일

"tomcat폴더/lib" 폴더에 mysql-connector-java-버전-bin.jar 파일이 있어야 한다.

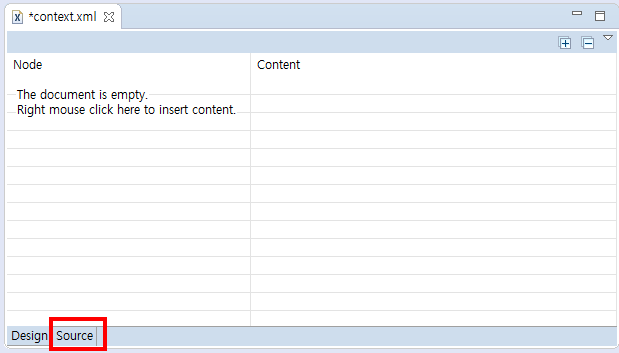
"01 MySQL 설치.docx" 문서에서 JDBC 드라이버 설치 부분을 참고하라.

## context.xml 파일 생성





File name: context.xml



### WebContent/META-INF/context.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <Context>  <Resource name="jdbc/student1" auth="Container" type="javax.sql.DataSource"  driverClassName="com.mysql.jdbc.Driver"  maxTotal="50" maxIdle="10" maxWaitMillis="10000"  username="user1" password="test123"  url="jdbc:mysql://localhost:3306/student1?useUnicode=true&amp;characterEncoding=utf8&amp;serverTimezone=UTC" />  </Context> |

위와 같이 입력하자.

위 파일의 내용 설명

|  |  |
| --- | --- |
| DB 서버 IP 주소 | localhost |
| DB 이름 | student1 |
| DB 사용자 계정 | user1 |
| 위 사용자 계정의 비밀번호 | test123 |

에 user1 계정이 DB에 등록되어 있어야 한다.

"01 MySQL 설치.docx" 문서를 참고하라.

MySQL 에는 student1 데이터베이스가 생성되어 있어야 한다.

"01 MySQL 설치.docx" 문서에서 Data Import 부분을 참고하라.

# 데이터 조회 Java 코드

## DB.java

### src/lecture1/DB.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | package lecture1;  import java.sql.Connection;  import java.sql.SQLException;  import javax.naming.InitialContext;  import javax.naming.NamingException;  import javax.sql.DataSource;  public class DB {  static DataSource dataSource = null;  public static Connection getConnection(String databaseName) throws SQLException, NamingException {  if (dataSource == null) {  InitialContext context = new InitialContext();  dataSource = (DataSource)context.lookup("java:comp/env/jdbc/" + databaseName);  }  return dataSource.getConnection();  }  } |

|  |  |
| --- | --- |
| package | lecture1 |
| class name | DB |

DB.getConnection 메소드는 Connection 객체를 생성하여 리턴한다.

Connection 객체는 DB에 연결하기 위한 JDBC 객체이다.

context.xml 파일에 등록된 정보로 DB에 연결하고, 그 DB 연결 객체를 리턴한다.

DB.getConnection 메소드의 파라미터는 DB 이름이다.

그 이름의 DB 연결 정보가 context.xml 파일에 등록되어 있어야 한다. (예: student1)

줄14의 if 문에 의해서, dataSource 객체는 한 개만 생성된다.

## Student.java

### src/lecture1/jdbc1/Student.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59 | package lecture1.jdbc1;  public class Student {  int id;  String studentNumber;  String name;  int departmentId;  int year;  String departmentName;  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  public String getStudentNumber() {  return studentNumber;  }  public void setStudentNumber(String studentNumber) {  this.studentNumber = studentNumber;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public int getDepartmentId() {  return departmentId;  }  public void setDepartmentId(int departmentId) {  this.departmentId = departmentId;  }  public int getYear() {  return year;  }  public void setYear(int year) {  this.year = year;  }  public String getDepartmentName() {  return departmentName;  }  public void setDepartmentName(String departmentName) {  this.departmentName = departmentName;  }  } |

Student 테이블에서 조회한 데이터를 채워서 전달하기 위한 객체이다.

이런 객체를 DTO (Data Transfer Object) 라고 부른다.

|  |  |
| --- | --- |
| package | lecture1.jdbc1 |
| class name | Student |

## StudentDAO.java

### src/lecture1/jdbc1/StudentDAO.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | package lecture1.jdbc1;  import java.sql.Connection;  import java.sql.PreparedStatement;  import java.sql.ResultSet;  import java.util.ArrayList;  import java.util.List;  import lecture1.DB;  public class StudentDAO {  public static List<Student> findAll() throws Exception {  String sql = "SELECT s.\*, d.departmentName " +  "FROM student s LEFT JOIN department d ON s.departmentId = d.id";  try (Connection connection = DB.getConnection("student1");  PreparedStatement statement = connection.prepareStatement(sql);  ResultSet resultSet = statement.executeQuery()) {  ArrayList<Student> list = new ArrayList<Student>();  while (resultSet.next()) {  Student student = new Student();  student.setId(resultSet.getInt("id"));  student.setStudentNumber(resultSet.getString("studentNumber"));  student.setName(resultSet.getString("name"));  student.setDepartmentId(resultSet.getInt("departmentId"));  student.setYear(resultSet.getInt("year"));  student.setDepartmentName(resultSet.getString("departmentName"));  list.add(student);  }  return list;  }  }  } |

|  |  |
| --- | --- |
| package | lecture1.jdbc1 |
| class name | StudentDAO |

findAll 메소드는,

DB의 student 테이블에서 조회한 데이터를 Student 객체에 채우고,

Student 객체들을 ArrayList에 채워서,

ArrayList 객체를 리턴한다.

(줄15) Connection 객체는 DB에 연결하기 위한 객체이다.

(줄16) PreparedStatement 객체는 SQL 명령을 실행하기 위한 객체이다.

(줄17)

PreparedStatement 객체의 executeQuery 메소드를 실행하면,

DB에서 SQL 명령이 실행되고,

SQL 명령의 조회 결과가 ResultSet 객체에 채워져 전달된다.

(줄19) 조회 결과 레코드 수 만큼 while 반복문이 실행된다.

(줄20~26) 조회 결과 데이터를 ResultSet 객체로부터 하나씩 꺼내서, Student 객체에 채운다.

위 코드는 Java7의 try with resource 문법을 사용하여 구현되었다.

이 문법은 connection, statement, resultSet 객체의 close 메소드를 자동으로 호출해 준다.

혹시 실수로 위 객체들의 close 메소드 호출을 빼먹으면, DB 성능에 큰 악영향이 초래되는데,

위 코드처럼 try with resource 문법을 사용하여 구현하면,

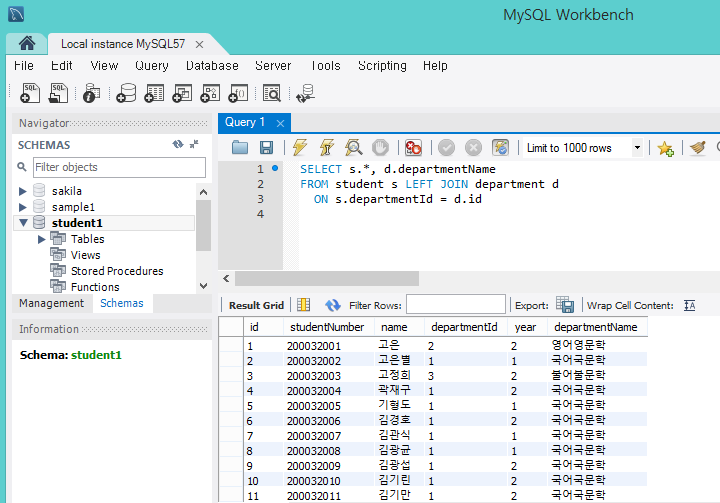
어떤 경우에도 위 객체들의 close 메소드가 자동으로 호출된다.

|  |
| --- |
| String sql = "SELECT s.\*, d.departmentName " +  "FROM student s LEFT JOIN department d ON s.departmentId = d.id"; |

위 코드에서 문자열 두 개를 더할 때, d.departmentName 부분과 FROM 사이에 공백이 있어야 함에 주의하자.

|  |
| --- |
| SELECT s.\*, d.departmentName  FROM student s LEFT JOIN department d ON s.departmentId = d.id |

위 SQL 명령을 MySQL Workbecnch에서 실행해 보자.



위 SQL 문법을 이해하지 못한다면, SQL 문법을 따로 공부해야 한다.

# JSP 파일 구현

## jdbc1/studentList1.jsp

### WebContent/jdbc1/studentList1.jsp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <%@ page import="java.util.List, lecture1.jdbc1.\*" %>  <%  List<Student> list = StudentDAO.findAll();  %>  <!DOCTYPE html>  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>  <style>  body { font-family: 굴림체; }  thead th { background-color: #eee; }  table.table { width: 700px; }  </style>  </head>  <body>  <div class="container">  <h1>학생목록</h1>  <table class="table table-bordered table-condensed">  <thead>  <tr>  <th>학번</th>  <th>이름</th>  <th>학과</th>  <th>학년</th>  </tr>  </thead>  <tbody>  <% for (Student student : list) { %>  <tr>  <td><%= student.getStudentNumber() %></td>  <td><%= student.getName() %></td>  <td><%= student.getDepartmentName() %></td>  <td><%= student.getYear() %></td>  </tr>  <% } %>  </tbody>  </table>  </div>  </body>  </html> |



<%@ page import="java.util.List, lecture1.jdbc1.\*" %>

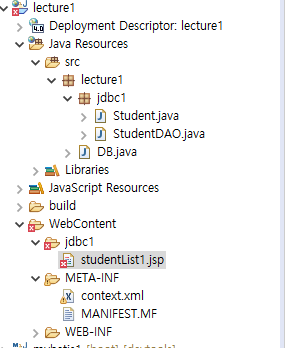
java.util.List 인터페이스에 대한 import 선언이다.

lecture1.jdbc1 패키지의 모든 클래스에 대한 import 선언이다.

JSP 파일에서 사용되는 클래스에 대한 import 선언이 이와 같은 형태로 구현되어야 한다.

import 선언이 없을 경우에, 어떤 에러가 발생하는지 확인해보라.

## 에러



studentList1.jsp 파일에 빨간색 표시가 보인다.

에러가 있다는 뜻이다.

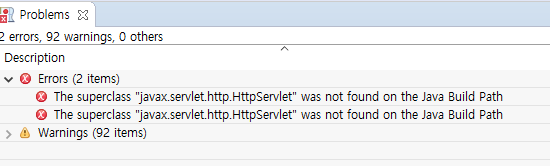
에러 메시지를 확인하자.

메뉴:

Window - Show View - Problems

Window - Show View - Other - General - Problems

위 두 메뉴 중에 하나를 실행하자.

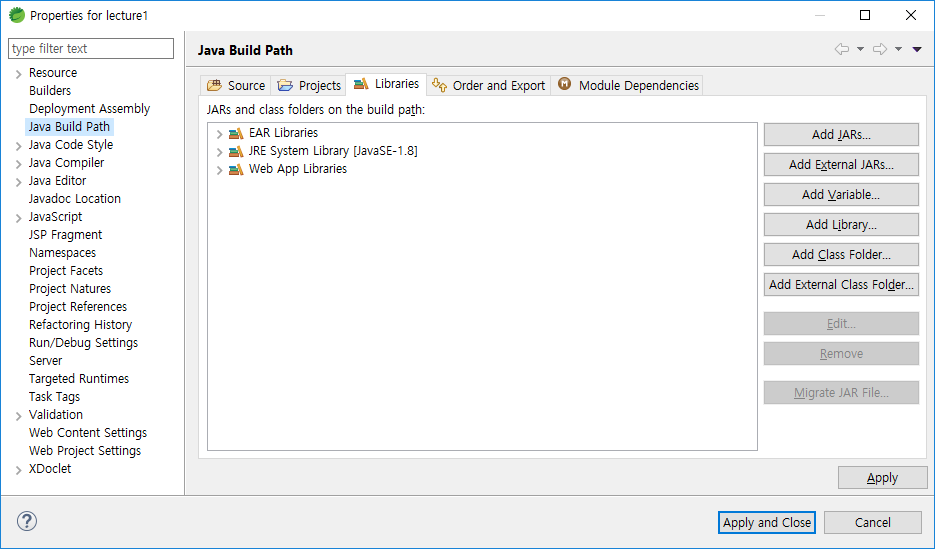


javax.servlet.http.HttpServlet 클래스가

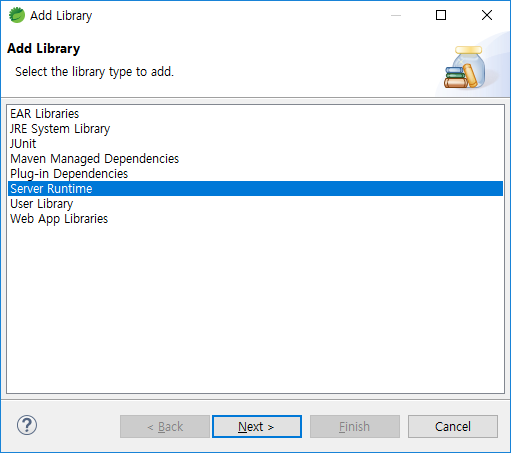
Java Build Path에 없다는 에러이다.

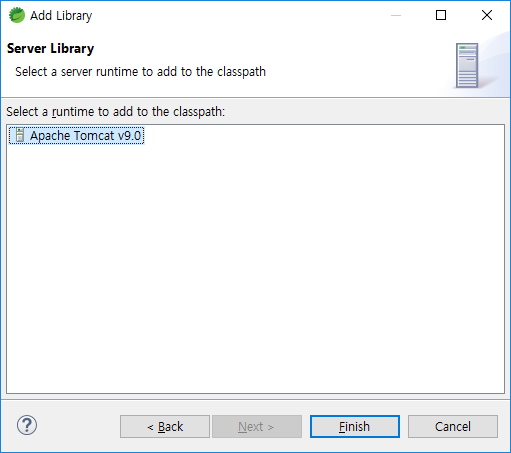
이 클래스가 포함된 라이브러리를 Java Build Path에 추가하자.

메뉴: Project - Properties - Java Build Path



Add Library 버튼 클릭





Finish 클릭

# 연습문제

student1 데이터베이스의 User 테이블의 레코드 목록을 출력하는 페이지를 구현하라.

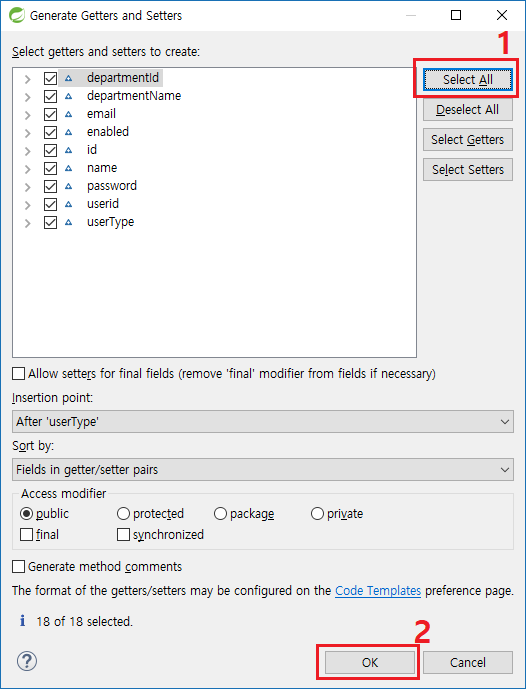
## User.java 구현

src/lecture1/jdbc1/User.java 파일을 아래와 같이 구현하라.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | package lecture1.jdbc1;  public class User {  int id;  String userid;  String password;  String name;  String email;  int departmentId;  String departmentName;  boolean enabled;  String userType;  } |

위 파일의 (줄13)에 커서를 위치한 상태에서

Source - Generate Getters and Setters 메뉴를 클릭하라.



위와 같이 선택하고 Ok 버튼을 클릭하면,

getter setter 메소드들이 자동으로 생성된다.

## UserDAO.java 구현

src/lecture1/jdbc1/UserDAO.java 구현

User 테이블의 모들 레코드를 조회하여, User 객체에 채우고, ArrayList<User> 객체를 리턴하는,

findAll 메소드를 구현하라.

User 테이블의 enabled 필드는 무시하고 구현하라.

## jdbc1/userList.jsp

WebContent/jdbc1/userList.jsp



# 부록 - 고급 Java 문법 활용하기

부록의 내용은 이번 학기 이 수업의 학습 목표에 포함되지 않는다.

## 자원 반납과 finally

### 자원반납 문제

|  |
| --- |
| 변수1 = new 자원객체1(); // 자원 할당  변수1.작업();  변수1.close(); // 자원 반납 |

위 코드에서 작업() 메소드가 에러 없이 실행된 경우에

마지막 문장에서 close() 메소드가 호출되어 자원이 반납된다.

그런데 만약 작업() 메소드 내부에서 exception이 발생하면,

실행흐름이 catch 블럭으로 점프하게 된다.

이 경우에, 마지막 문장도 건너뛰게 되어 close() 메소드가 호출되지 않는다.

자원을 반납하지 못하는 문제가 발생한다.

### finally 블럭에서 자원 반납 구현

변수1 = null;

변수2 = null;

try {

변수1 = new 자원객체1();

변수2 = new 자원객체2();

변수1.작업();

변수2.작업();

...

} finally {

if (변수1 != null) 변수1.close();

if (변수2 != null) 변수2.close();

}

try finally 문법으로 자원 반납(resource close)을 구현한 코드이다.

작업 메소드에서 exception이 발생한 경우에도,

반드시 close 메소드를 호출하여 자원 반납을 해야 하는 경우에

위 코드처럼 구현해야 한다.

exception이 발생해서 현재 메소드의 실행을 중단하고 나갈 때에도, finally 블럭은 반드시 실행된다.

즉 어떠한 경우에도 finally 블럭의 코드는 반드시 실행된다.

따라서 자원을 반납하지 못하는 문제가 발생하지 않는다.

## try with resource 문법과 자원 반납

### try with resource 문법

Java 1.7에 추가된 try with resource 문법을 사용하면

간결하게 구현할 수 있다.

try with resource 문법은 다음과 같다.

try (변수1 = new 자원객체1()) {

...

}

위 형태로 구현하면 자원객체의 close 메소드 호출은 자동으로 실행되기 때문에 finally 블럭을 생략해도 된다.

자원객체가 여러개면 다음과 같이 구현한다.

try (변수1 = new 자원객체1();

변수2 = new 자원객체2()) {

...

}

try 문장을 중첩하여 구현할 수도 있다.

try (변수1 = new 자원객체1();

변수2 = new 자원객체2()) {

...

try (변수3 = new 자원객체3()) {

...

}

}

실행흐름이 try 블럭을 빠져 나갈 때, 자원객체의 close 메소드가 자동으로 호출된다.

exception이 발생해서 실행흐름이 점프할 때에서, 자원객체의 close 메소드는 반드시 자동으로 호출된다.

### 예제#1

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | package net.skhu.tryWithResource;  import java.io.BufferedReader;  import java.io.FileReader;  public class Example1 {  static void printFile(String 파일명) throws Exception {  BufferedReader reader = new BufferedReader(new FileReader(파일명));  String s;  while ((s = reader.readLine()) != null) {  System.out.println(s);  }  reader.close();  }  public static void main(String[] args) throws Exception {  String 파일명 = "e:/temp/테스트.txt";  printFile(파일명);  }  } |

BufferedReader 객체를 다 사용한 후 close 메소드를 호출해서 파일을 닫아야 한다. (줄14)

그런데 위 코드에서

printFile 메소드를 실행하는 도중에 에러(exception)가 발생하면,

printFile 메소드 실행을 즉시 중단하고, 메소드 밖으로 리턴하게 된다.

이 경우에 (줄14)의 close 메소드는 호출되지 않기 때문에,

BufferedReader 객체가 close 되지 않는 문제가 있다.

### 예제#2

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | package net.skhu.tryWithResource;  import java.io.BufferedReader;  import java.io.FileReader;  public class Example2 {  static void printFile(String 파일명) throws Exception {  BufferedReader reader = null;  try {  reader = new BufferedReader(new FileReader(파일명));  String s;  while ((s = reader.readLine()) != null) {  System.out.println(s);  }  } finally {  if (reader != null) reader.close();  }  }  public static void main(String[] args) throws Exception {  String 파일명 = "e:/temp/테스트.txt";  printFile(파일명);  }  } |

printFile 메소드가 정상적으로 리턴하게 될 때 뿐만 아니라,

실행 도중 에러(exception)가 발생해서 리턴하게 될 때에도,

finally 블럭(줄16~18)은 반드시 실행된다.

즉 BufferedReader 객체의 close 메소드가 반드시 호출된다.

### 예제#3

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | package net.skhu.tryWithResource;  import java.io.BufferedReader;  import java.io.FileReader;  public class Example3 {  static void printFile(String 파일명) throws Exception {  try (BufferedReader reader = new BufferedReader(new FileReader(파일명))) {  String s;  while ((s = reader.readLine()) != null) {  System.out.println(s);  }  }  }  public static void main(String[] args) throws Exception {  String 파일명 = "e:/temp/테스트.txt";  printFile(파일명);  }  } |

Java7 에서 추가된 try with resource 문법으로 자원 반납을 구현하였다.

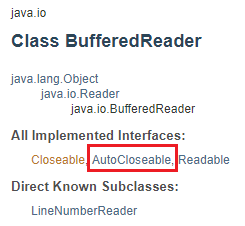
위 코드는 예제#2 와 동일하게 작동한다.

즉 BufferedReader 객체의 close 메소드가 반드시 호출된다.

예제#3의 코드가 예제#2 보다 훨씬 간결해서 좋다.

BufferedReader 클래스가 AutoCloseable 인터페이스를 implements 했음을 확인하자.

https://docs.oracle.com/javase/7/docs/api/java/io/BufferedReader.html



## AutoCloseable 인터페이스

AutoCloseable 인터페이스를 implements 한 클래스만 try with resource 문법을 사용할 수 있다.

이 인터페이스에는 close 메소드가 정의되어 있다.

따라서 이 인터페이스를 implements 한 클래스들은 반드시 close 메소드를 구현해야 한다.

https://docs.oracle.com/javase/7/docs/api/java/lang/AutoCloseable.html

### 예제

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | package net.skhu.tryWithResource;  class MyResource implements AutoCloseable {  public MyResource() {  System.out.println("자원이 할당됨");  }  public void 작업() throws Exception {  System.out.println("작업 중 에러 발생");  throw new Exception();  }  @Override  public void close() throws Exception {  System.out.println("자원이 반납됨");  }  }  public class Example4 {  public static void main(String[] args) throws Exception {  try (MyResource resource = new MyResource()) {  resource.작업();  }  }  } |

이 예제를 실행하고 분석해 보자.

작업 중간에 exception이 throw 되어도, close 메소드가 호출됨을 확인하자.

|  |
| --- |
| package lecture1.jdbc1;  import java.sql.Connection;  import java.sql.PreparedStatement;  import java.sql.ResultSet;  import java.util.ArrayList;  import java.util.List;  import lecture1.DB;  public class StudentDAOa {  public static List<Student> findAll() throws Exception {  String sql = "SELECT s.\*, d.departmentName " +  "FROM student s LEFT JOIN department d ON s.departmentId = d.id";  Connection connection = null;  PreparedStatement statement = null;  ResultSet resultSet = null;  ArrayList<Student> list = new ArrayList<Student>();  try {  connection = DB.getConnection("student1");  statement = connection.prepareStatement(sql);  resultSet = statement.executeQuery();  while (resultSet.next()) {  Student student = new Student();  student.setId(resultSet.getInt("id"));  student.setStudentNumber(resultSet.getString("studentNumber"));  student.setName(resultSet.getString("name"));  student.setDepartmentId(resultSet.getInt("departmentId"));  student.setYear(resultSet.getInt("year"));  student.setDepartmentName(resultSet.getString("departmentName"));  list.add(student);  }  } finally {  if (connection != null) connection.close();  if (statement != null) statement.close();  if (resultSet != null) resultSet.close();  }  return list;  }  } |

위 코드는 try finally 문법으로 StudentDAO 클래스를 구현한 것이다.

try with resource 문법보다 훨씬 장황하다.