PEMROGRAMAN BERORIENTASI OBJEK LANJUT – BS405

PERT 4 – JAVA DATABASE API BAG. 01

BY : SENDY FERDIAN SUJADI, S.KOM., M.T., CEH, CEI, MTCNA, MTCRE, MTCINE, MTA

TODAY'S MENUS:

1. **JDBC — Java Database Connectivity**
   - ➢Pengenalan library JDBC
   - ➢Diagram pengaksesan database melalui JDBC
   - ➢Step-by-step Setting JDBC untuk MySQL
   - ➢Cara Melakukan Query (SELECT)
   - ➢Cara Memproses Hasil Query
   - ➢Cara Update/Insert/Delete
2. **Hibernate**
   - ➢What is Hibernate
   - ➢Features
   - ➢Configurations
   - ➢CRUD (Create, Read, Update, Delete)
     - ➢HQL
     - ➢Criteria Query
     - ➢Native SQL

JDBC

JAVA DATABASE CONNECTIVITY

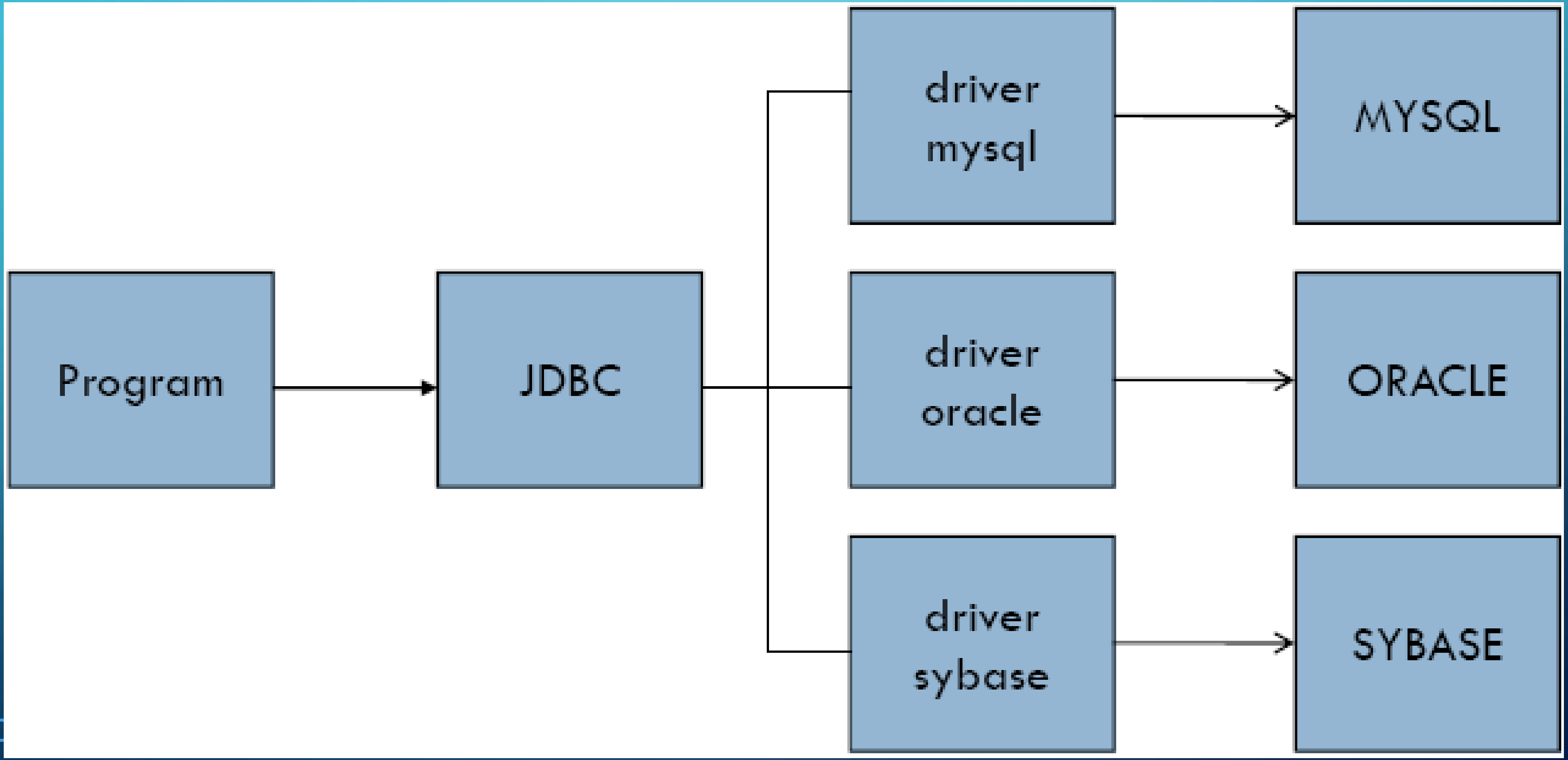
1ST THING 1ST : SIAPKAN MYSQL

- Pastikan di komputer kalian ada MySQL Server.
- Pastikan di komputer kalian ada SQLyog. (bisa diganti dengan phpMyAdmin, sqlWorkbrench, TOAT for MySQL, HeidiSQL, atau client application for MySQL lainnya)
- Jalankan MySQL server.
- Jalankan SQLyog.
- Jika belum ada database, create new database.
- Jika sudah ada database dengan ekstensi .sql maka dapat dilakukan import database.

PENGENALAN: JDBC LIBRARY

- JDBC — Java Database Connectivity — adalah suatu feature di Java yang memungkinkan kita untuk melakukan:
  - Koneksi ke hampir semua sistem RDBMS yang ada saat ini,
  - Eksekusi perintah SQL, dan
  - Memproses hasil perintah SQL melalui program Java
- Library JDBC terdiri atas class-class yang berguna untuk setiap tasks di dalam pemrosesan database, misalnya class untuk:
  - Membuat koneksi ke database
  - Membuat statement menggunakan SQL
  - Mengeksekusi query SQL (statement) di dalam database
  - Menampilkan records yang dihasilkan
- Semua class-class JDBC adalah bagian dari java.sql package.

BAGAIMANA PROGRAM JAVA MENGAKSES DATABASE MELALUI JDBC?



WHAT IS JDBC DRIVER?

- First Thing First: Setting Up a Driver
  - Sebelum Anda dapat menulis program Java yang mengakses database melalui JDBC, pertama-tama Anda harus meng-instal sebuah Driver yang menghubungkan class-class di dalam Java's database API dengan database yang akan digunakan.
- Untuk setiap database yang ada di pasaran saat ini, hampir semuanya memiliki driver JDBC-nya.
- Biasanya driver ini dapat didownload secara gratis melalui website perusahaan database masing-masing.
- Driver ini seringkali disebut juga sebagai "connector".
- Untuk NetBeans IDE, letakkan connector ini di folder Libraries

6 LANGKAH KONEKSI & AKSES DATABASE

1. Memanggil Driver JDBC.
2. Mendefinisikan URL untuk Koneksi Basis Data & Melakukan Koneksi tsb.
3. Membuat Objek Statement.
4. Melakukan Query/Update.
5. Memproses Hasil.
6. Menutup Koneksi.

1. MEMANGGIL DRIVER JDBC

- Sebelum kita dapat menggunakan JDBC untuk mengakses database SQL, kita harus melakukan koneksi terlebih dahulu.
- Langkah pertama dalam melakukan koneksi adalah: registrasi driver
- Caranya adalah dengan menggunakan method "forName" dari kelas "Class"
- Misalnya untuk meregistrasi connector MySQL, gunakan perintah berikut:

```
Class.forName("com.mysql.jdbc.Driver");
```

- Note that the forName method throws ClassNotFoundException, so you have to enclose this statement in a try/catch block that catches ClassNotFoundException.

```
try {
  Class.forName("com.mysql.jdbc.Driver");
}
catch (ClassNotFoundException){
  // error handling
}
```

2. MENDEFINISIKAN URL UNTUK KONEKSI BASIS DATA & MELAKUKAN KONEKSI TSB

- Buatlah sebuah method yang akan me-return sebuah objeck Connection.
- Method ini akan memanggil static method class DriverManager yaitu getConnection
- Static method getConnection (class DriverManager) memiliki 3 parameter yaitu url database, user name, dan password

```
String url = "jdbc:mysql://localhost/db_name";
String user = "root";
String pw = "";
con = DriverManager.getConnection(url, user, pw);
```

- Statement diatas akan melemparkan sebuah SQLException jika terjadi kesalahan. Dengan demikian, buat sebuah blok try-catch untuk exception ini.

PUTTING ALL TOGETHER (STEP 1 TO 2)

- Berikut ini adalah sebuah method yang me-return objek Connection yang berfungsi sebagai penghubung ke database MySQL:

```
private static Connection getConnection(){
    Connection con = null;
    try{
        Class.forName("com.mysql.jdbc.Driver");
        String url = "jdbc:mysql://localhost/db_name";
        String user = "root";
        String pw = "";
        con = DriverManager.getConnection(url, user, pw);
    }
    catch (ClassNotFoundException e){
        System.out.println(e.getMessage());
    }
    catch (SQLException e){
        System.out.println(e.getMessage());
    }
    return con;
}
```

3. MEMBUAT OBJEK STATEMENT

- Kita memerlukan objek Statement untuk melakukan query dan objek ini dapat dibuat dari objek Connection.

```
Statement st = con.createStatement();
```

- Statement: The Statement interface contains the methods necessary to send statements to the database for execution and return the results.
  - Use the **executeQuery** method to execute a **select** statement, or
  - Use the **executeUpdate** method to execute an **insert**, **update**, or **delete** statement.

4. MELAKUKAN QUERY ATAU UPDATE

- Setelah kita memiliki objek **Statement**, kita dapat menggunakannya untuk mengirimkan query dan mengeksekusinya dengan method **executeQuery** yang **menghasilkan** objek bertipe **ResultSet**.
- **ResultSet**: The ResultSet interface represents **rows returned** from a query.
- It provides methods you can use to **move from row to row** and to **get the data** for a column.

```
String query = "SELECT * FROM books";
ResultSet rs = st.executeQuery(query);
```

Note: Apabila ingin melakukan **insert / update / delete**, gunakan

```
st.executeUpdate(query);
```

5A. MEMPROSES HASIL QUERY

- Dalam memproses hasil, kita menggunakan objek **resultSet** karena hasil query disimpan dalam objek ini.
- Method utama yang sering digunakan: **next** dan **getString**

Contoh pemrosesan hasil query:

```
while (rs.next()){
   System.out.println(rs.getString(1) + " " +
   rs.getString(2));
}
```

- Kode di atas akan menampilkan **semua baris hasil** query yang masing-masing menampilkan data **kolom pertama dan kedua.**

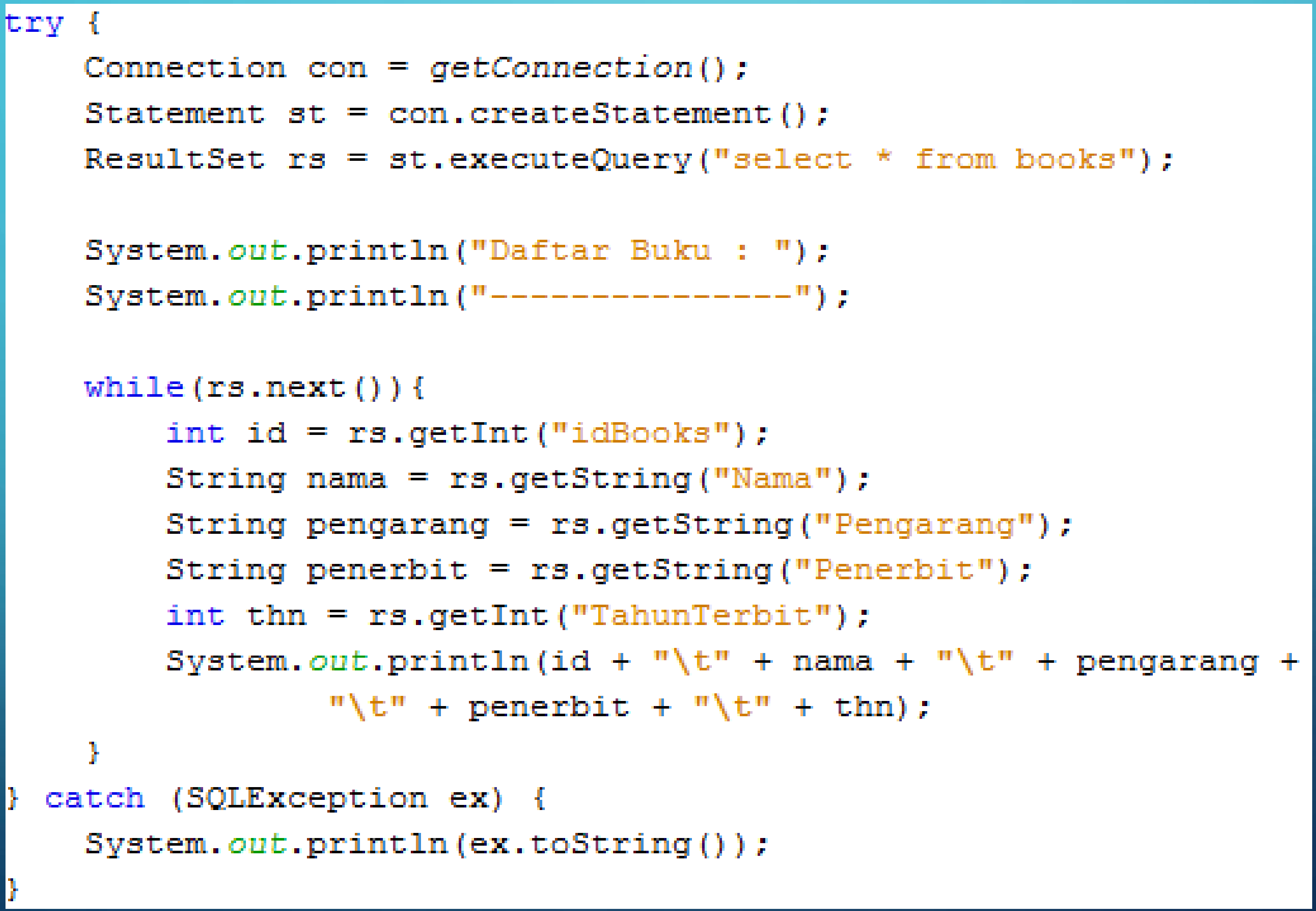
NAVIGATING THROUGH THE RESULT SET

- The **ResultSet object returned by the executeQuery statement contains** all the rows that are retrieved by the select statement.
- You can only access one of those rows at a time.
- You can use the methods shown in the table to move the cursor through a result set.

Method	Description
`void close()`	Closes the result set
`void last()`	Moves the cursor to the last row
`int getRow()`	Gets the current row number
`boolean next()`	Moves to the next row

```
while(rows.next()) {
  // process the current row
}
```

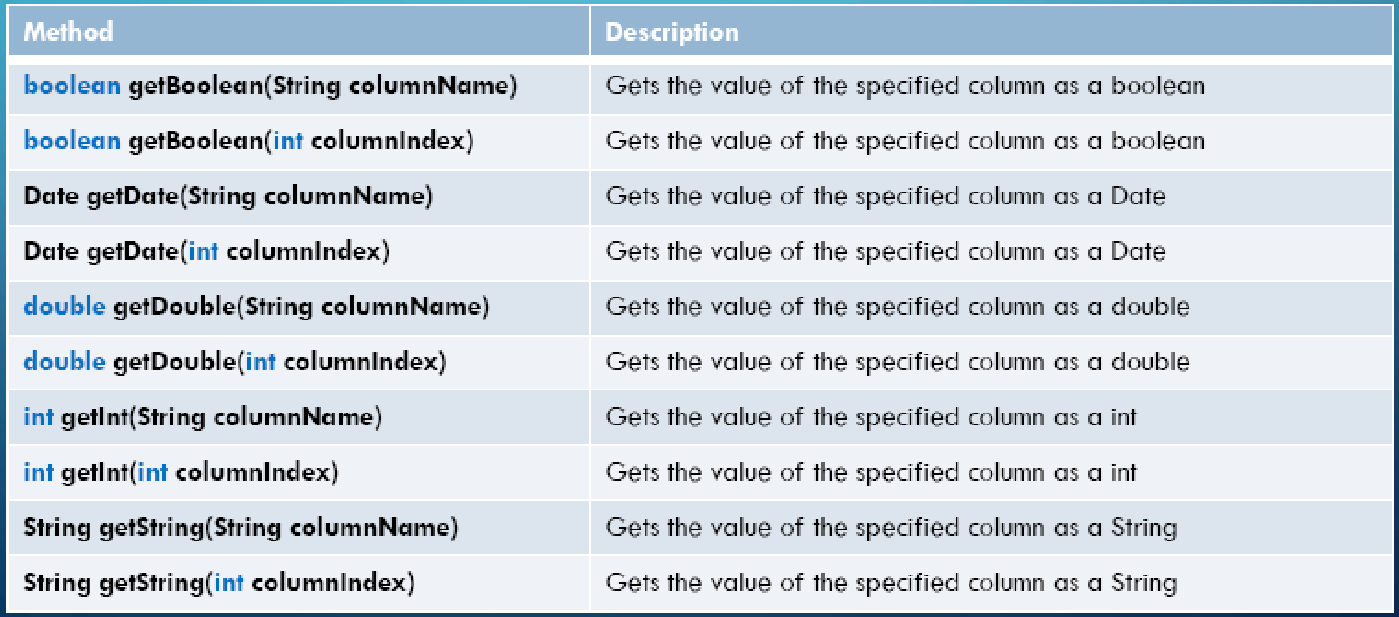
EXAMPLE : NAVIGATING THROUGH THE RESULT SET



```java
try {
    Connection con = getConnection();
    Statement st = con.createStatement();
    ResultSet rs = st.executeQuery("select * from books");

    System.out.println("Daftar Buku : ");
    System.out.println("---------------");

    while(rs.next()){
        int id = rs.getInt("idBooks");
        String nama = rs.getString("Nama");
        String pengarang = rs.getString("Pengarang");
        String penerbit = rs.getString("Penerbit");
        int thn = rs.getInt("TahunTerbit");
        System.out.println(id + "\t" + nama + "\t" + pengarang +
                "\t" + penerbit + "\t" + thn);
    }
} catch (SQLException ex) {
    System.out.println(ex.toString());
}
```

GETTING DATA FROM THE RESULT SET

- The following table lists the methods of the **ResultSet interface you can use to** retrieve data from the current row.
- As you can see, each of these methods comes in two versions:
  - One specifies the column **by name, the other by index number.**
  - If you know the index number, using it to access the column values is more efficient than using the column names.

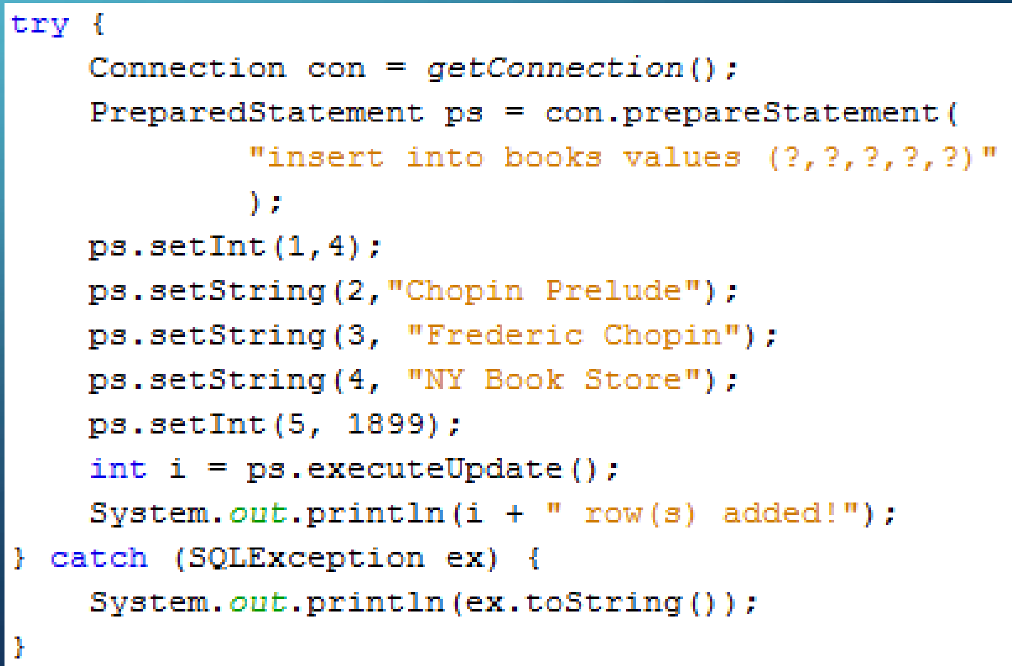


Method	Description
boolean getBoolean(String columnName)	Gets the value of the specified column as a boolean
boolean getBoolean(int columnIndex)	Gets the value of the specified column as a boolean
Date getDate(String columnName)	Gets the value of the specified column as a Date
Date getDate(int columnIndex)	Gets the value of the specified column as a Date
double getDouble(String columnName)	Gets the value of the specified column as a double
double getDouble(int columnIndex)	Gets the value of the specified column as a double
int getInt(String columnName)	Gets the value of the specified column as a int
int getInt(int columnIndex)	Gets the value of the specified column as a int
String getString(String columnName)	Gets the value of the specified column as a String
String getString(int columnIndex)	Gets the value of the specified column as a String

5B. PROSES INSERT/UPDATE/DELETE

- Setelah mengerti bagaimana menampilkan data, maka kita perlu mengerti bagaimana menambah / menghapus / mengupdate data ke tabel.
- Untuk melakukan hal tersebut, kita menggunakan method :
  `executeUpdate("perintah sql untuk insert / update / delete");`
- Method tersebut akan menghasilkan nilai integer yang merupakan jumlah baris yang dipengaruhi oleh proses update tersebut.
  int i = st.executeUpdate("delete from movie where id = ‘2' ");

- Untuk proses updating tabel, kita bisa juga menggunakan objek PreparedStatement.
- Untuk mengeksekusi PreparedStatement, kita gunakan method executeUpdate().



```
try {
    Connection con = getConnection();
    PreparedStatement ps = con.prepareStatement(
            "insert into books values (?,?,?,?,?)"
            );
    ps.setInt(1,4);
    ps.setString(2,"Chopin Prelude");
    ps.setString(3, "Frederic Chopin");
    ps.setString(4, "NY Book Store");
    ps.setInt(5, 1899);
    int i = ps.executeUpdate();
    System.out.println(i + " row(s) added!");
} catch (SQLException ex) {
    System.out.println(ex.toString());
}
```

6. MENUTUP KONEKSI

- Sebelum menutup koneksi basis data, kita juga perlu melepaskan objek ResultSet yang ada dengan kode berikut:

```
st.close();
```

- Untuk menutup koneksi ke basis data, kita tuliskan sbb:

```
conn.close();
```

HIBERNATE

Referensi

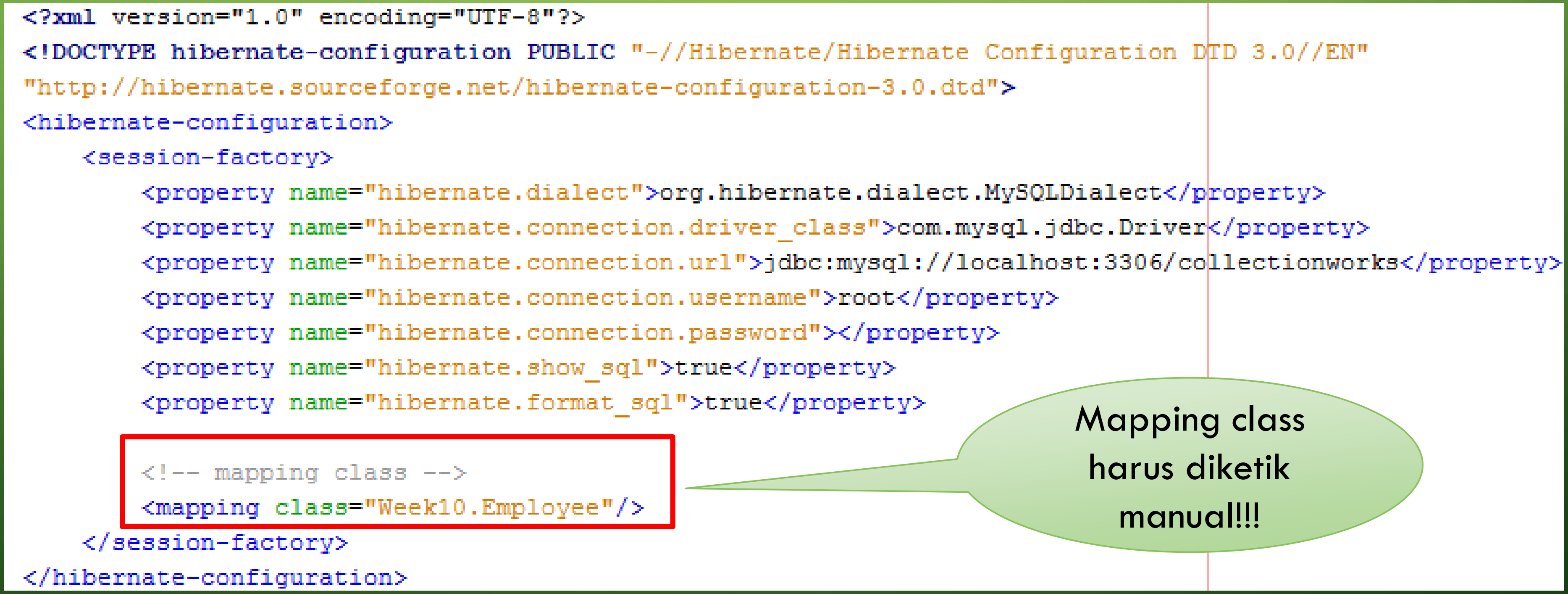
- http://www.hibernate.org
- hibernate_reference.pdf
- hibernate_annotations.pdf
- Google

WHAT IS HIBERNATE

- Hibernate adalah sebuah library ”Object Relational Mapping”(ORM) untuk bahasa pemrograman Java yang memberikan kerangka kerja untuk memetakan model domain berorientasi objek ke relasional database tradisional.
- Fitur-Fitur:
    - Pemilihan tipe data otomatis
    - Mendukung banyak database popular
    - Mapping Java Object menjadi Struktur Tabel Database
    - Sedikit menggunakan query SQL

HIBERNATE IN ACTION

- Hibernate.cfg.xml (chapter 3, hibernate_reference)
  - File konfigurasi hibernate
  - Berisi url, user, dan password database, driver, database yang digunakan, dll
  - Biasanya **diletakkan** di package **src** (default folder) project

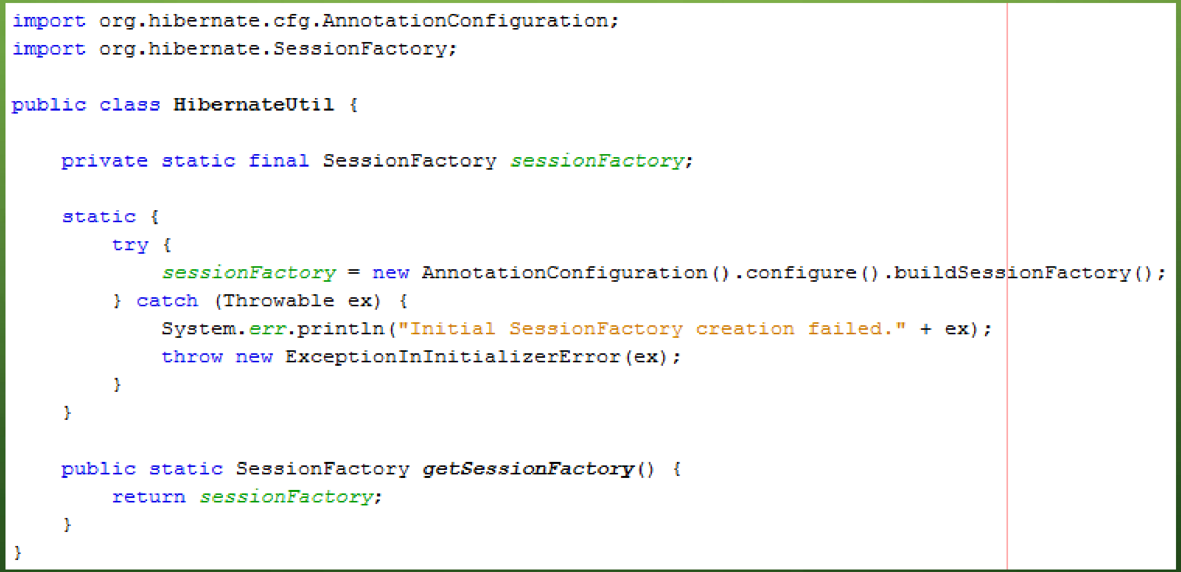


```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/collectionworks</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password"></property>
        <property name="hibernate.show_sql">true</property>
        <property name="hibernate.format_sql">true</property>

        <!-- mapping class -->
        <mapping class="Week10.Employee"/>
    </session-factory>
</hibernate-configuration>
```

Mapping class harus diketik manual!!!

HIBERNATE IN ACTION *CONTD…*

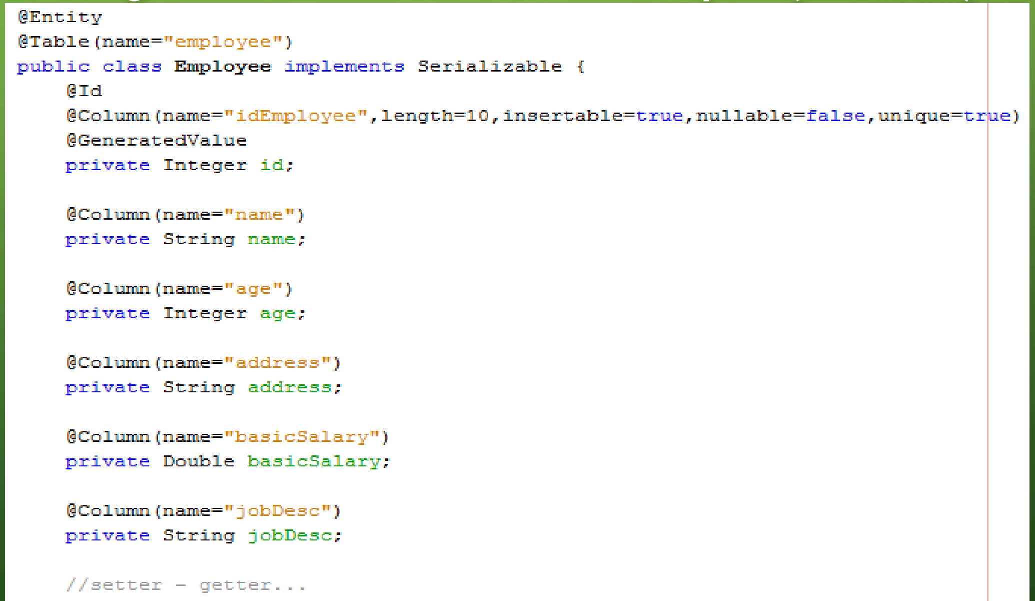
- HibernateUtil.java
  - kelas Java yang menangani startup dan membuat akses SessionFactory dengan "*convenient*".
  - **Hibernate SessionFactory** - factory yang menyediakan Session untuk bekerja dengan database dalam satu transaksi.
  - Factory ini membuat atau membuka session untuk berinteraksi dengan database.
  - **Session** dapat **diambil** dari **SessionFactory** dan setelah digunakan dapat di "***flush***" dan di **tutup**. Tetapi SessionFactory harus dibuat hanya sekali, dan dapat dikonfigurasi untuk menerima "*DataSource*" dan "*Transaction Attributes*".



```
import org.hibernate.cfg.AnnotationConfiguration;
import org.hibernate.SessionFactory;

public class HibernateUtil {

    private static final SessionFactory sessionFactory;

    static {
        try {
            sessionFactory = new AnnotationConfiguration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

HIBERNATE IN ACTION *CONTD…*

- Persistence Classes
  - Kelas Persistent adalah kelas pada aplikasi yang menerapkan entitas dari "business problem"(misalnya Customer and Order in an E-commerce application).
  - Terdiri dari atribut-atribut dan method setter-getter yang merepresentasikan data dalam database
  - Juga dikenal sebagai Plain Old Java Object(POJO)



```
@Entity
@Table(name="employee")
public class Employee implements Serializable {
    @Id
    @Column(name="idEmployee",length=10,insertable=true,nullable=false,unique=true)
    @GeneratedValue
    private Integer id;

    @Column(name="name")
    private String name;

    @Column(name="age")
    private Integer age;

    @Column(name="address")
    private String address;

    @Column(name="basicSalary")
    private Double basicSalary;

    @Column(name="jobDesc")
    private String jobDesc;

    //setter - getter...
```

HIBERNATE'S ANNOTATION

- We used annotation (@) to do the object relational mapping (@Entity, @Id, @Column, etc)
- Hibernate support EJB3 persistence annotations (JDK 5.0)
- These annotations are contained in javax.persistence package

ANNOTATION (@)

- The **@Entity** annotation is used to mark this class as an Entity (Entitas) bean. So the class should atleast have a package scope no-argument constructor.

- The **@Table** annotation is used to specify the table to persist the data. The name attribute refers to the table name.
- If @Table annotation is not specified then Hibernate will by default use the class name as the table name.

- The **@Id** annotation is used to specify the identifier property of the entity bean.
- The placement of the @Id annotation determines the default access strategy that Hibernate will use for the mapping.
- If the @Id annotation is placed over the field, then filed access will be used.

- The **@Column** annotation is used to specify the details of the column to which a field or property will be mapped.
- Here the id property is mapped to idEmployee column in the Employee table.
- If the @Column annotation is not specified by default the property name will be used as the column name.

- The **@GeneratedValue** annotation is used to specify the primary key generation strategy to use.
- If the strategy is not specified by default AUTO will be used.

QUERYING IN HIBERNATE

- Querying in Hibernate usually uses DAO class
  - HQL (Hibernate Query Languange) – Chapter 14
    - OO Query (Based on persistence/entity classes)
  - Criteria Queries (Preferable) – Chapter 15
    - No query needed, just use method provided by Hibernate
  - Native SQL Queries

HQL (HIBERNATE QUERY LANGUANGE)

Clauses	Description
from	The simplest form of an HQL query. Specifies the object whose instances are to be returned as the query result. Commonly used with the select clause.
select	Specifies objects and properties to be returned in the query result set. Used in conjunction with the from clause.
where	Specifies the conditions that should be satisfied by the instances returned as the query result. Used with select and/or from clause.
order by	Specifies the order (<u>asc</u>ending/<u>desc</u>ending) in which the properties of objects returned as query results should be listed. Used with the select and from clauses.
group by	Specifies the grouping criteria using objects properties, by which the list of objects returned as a query result should be grouped together. Used with the select and/or from clause.

GETTING DATA – HQL QUERY

- Using the `createQuery()` method of a Session object that **returns a Query object**.
- First, **instantiate the Session object** using the `openSession()` method of SessionFactory. Then, invoke the `createQuery()` method on the **resulting object**.

```
SessionFactory session = HibernateUtil.getSessionFactory();
session.openSession();
Query q = s.createQuery("from Employee order by id");
session.close(); //Agar session tidak memberatkan memori
List result = q.list();
```

- Here, the Query object q uses the **list()** method to return a List object containing a list of query results.

SAVING & UPDATING DATA

- Hibetnate has set of methods for saving and updating the values in the database.
  - save()
    - save method stores an object into the database. That means it insert an entry if the identifier doesn't exist, else it will throw error. If the primary key already present in the table, it cannot be inserted.
  - update() : the application loads an object in the first session
    - update method in the hibernate is used for updating the object using identifier. If the identifier is missing or doesn't exist, it will throw exception.
  - merge()
    - merge method in the hibernate is used for copy the state onto the persistence instance. If there is no persistent instance currently associated with the session, try to load it from the database, or create a new persistent instance
  - saveOrUpdate() : the application loads an object in the first session
    - This method calls save() or update() based on the operation. If the identifier exists, it will call update method else the save method will be called.

SAVING & UPDATING EXAMPLE

```
Employee emp = new Employee();
emp.setId(1);
emp.setFirstName("Foo");
emp.setLastName("Bar");
emp.setAddress("Arkansas");

Session s = beginSession();
Transaction t = s.beginTransaction();
s.save(emp);
t.commit();
```

```
Employee emp1 = new Employee();
emp1.setId(1);
emp1.setFirstName("Rudy");

Session s = beginSession();
Transaction t = s.beginTransaction();
s.saveOrUpdate(emp1);
t.commit();
```

```
Employee emp2 = new Employee();
emp2.setId(1);
emp2.setAddress("Ciroyom");


Session s = beginSession();
Transaction t = s.beginTransaction();
s.update(emp2);
t.commit();
```

DELETING DATA

- Deleting a single row using Hibernate is simple; get an instance of the persistent object you want to delete (ie: a Book, a Employee, etc) and an instance of the Session object and then called the delete() method:

```
int bookid = 71;
Session s = beginSession();
Book book = (Book)sess.load(Book.class, bookid);
s.delete(book);
s.getTransaction.commit();
```

- if you wanted to delete all the computer books for some reason, you'd have to first query the database using Hibernate to get a Collection of books, then iterate over the books, calling delete() on each one.

```
Collection books = BookFactory.getComputerBooks();
Session s = beginSession();
Iterator it = books.iterator();
while(it.hasNext()){
        Book book = (Book)it.next();
        s.delete(book);
}
s.getTransaction.commit();
```

DELETING DATA

- Instead of calling delete() on each one, you use a SQL (or a HQL) statement that fetches the objects and then pass the results directly to the delete method:

```
Session s = beginSession();
String query = “select book FROM Book as book where type = 'technical‘”;
s.delete(query);
s.getTransaction().commit();
```

THAT'S ALL FOR TODAY!

BY : SENDY FERDIAN SUJADI, S.KOM., M.T., CEH, CEI, MTCNA, MTCRE, MTCINE, MTA