6 Побудова ПЗ для роботи з БД
6 Database application design

6.1 Технологія JDBC / JDBC technology

JDBC (Java DataBase Connectivity)

стандартний прикладний інтерфейс мови Java для організації взаємодії між застосуванням і СУБД

standard API (Application Programming Interface) used to organize interaction between the application and DBMS

Взаємодія здійснюється за допомогою драйверів JDBC, що забезпечують реалізацію загальних інтерфейсів для конкретних СУБД і конкретних протоколів

Interaction is implemented using JDBC drivers that provide common interfaces for certain DBMS and protocols

6.1 Технологія JDBC / JDBC technology

JDBC drivers

1 Використовує інший інтерфейс взаємодії з СУБД, зокрема ODBC (**JDBC-ODBC bridge**)

Driver uses another API to interact with ODBC (**JDBC-ODBC bridge**)

JDK: sun.jdbc.odbc.JdbcOdbcDriver

2 Працюючий через зовнішні (**native**) бібліотеки (клієнта СУБД)
Driver uses external (**native**) libraries (through the DBMS client)

3 Працюючий з **мережевим** і незалежним від СУБД протоколом з проміжним Java-сервером
Driver uses **network** and DBMS-independent protocol, interacts with intermediate Java-server

4 Мережевий драйвер, який працює **безпосередньо** з СУБД
Network driver that works **directly** with the DBMS

6.1 Технологія JDBC / JDBC technology

JDBC надає інтерфейс для розробників, які використовують різні СУБД

JDBC provides the interface for developers that use various DBMS

За допомогою JDBC відсилаються SQL-запити тільки до реляційних баз даних, для яких існують драйвери, які знають спосіб спілкування з сервером баз даних

JDBC is used to send SQL queries only to relational databases, for which drivers available to interact with the database server exist

JDBC не відноситься безпосередньо до J2EE, але так як взаємодія з СУБД є невід'ємною частиною корпоративних додатків, часто розглядається в даному контексті

JDBC does not belong to J2EE directly, but it is often considered as its part, since interaction with DBMS is integral part of enterprise applications

6.1 Технологія JDBC / JDBC technology

Послідовність дій / Sequence of actions

1 Завантаження класу драйвера бази даних

Load the class of the database driver

String driverName = "org.gjt.mm.mysql.Driver";

Class.forName(driverName);

2 Установка з'єднання з базою даних

Create connection with the database

Connection cn = DriverManager.getConnection(“jdbc:mysql://localhost/my_db”, “root”, “pass”);

3 Створення об'єкта для передачі запитів

Create object to send queries

Statement st = cn.createStatement();

6.1 Технологія JDBC / JDBC technology

Об'єкт класу **Statement** використовується для виконання SQL-запиту без його попередньої підготовки

The object of the **Statement** class is used to execute a SQL query without prior preparation

Можуть застосовуватися також об'єкти класів **PreparedStatement** і **CallableStatement** для виконання підготовлених запитів і збережених процедур

Objects of the **PreparedStatement** and **CallableStatement** classes can also be used to execute prepared queries and stored procedures.

4 Виконання запиту / Query execution

ResultSet rs = st.executeQuery(“SELECT * FROM my_table”);

6.1 Технологія JDBC / JDBC technology

Щоб додати, видалити або змінити інформацію в таблиці замість методу **executeQuery()** запит поміщається в метод **executeUpdate()**

To add, remove or modify information in the table instead of the **executeQuery()** method, the query is placed in the **executeUpdate()** method

5 Обробка результатів виконання запиту здійснюється методами інтерфейсу **ResultSet**

Processing the results of the query is performed by the methods of the interface **ResultSet**

При першому виклику методу **next()** покажчик переміщається на таблицю результатів вибірки в позицію першого рядка таблиці

The first time the **next()** method is called, the pointer is moved to the table of sample results at the position of the first line of the response table

6.1 Технологія JDBC / JDBC technology

6 Закриття з'єднання / Close the connection

cn.close();

Додатково потрібно підключити бібліотеку, яка містить драйвер MySQL
Additionally, you need to connect the library containing the MySQL driver

mysql-connector-java-3.1.12.jar

Призначена для користувача база даних має ім'я db2 і одну таблицю users
The user database is named *db2* and has one *users* table

Field Name	Data Type
name	String
phone	Numeric

6.1 Технологія JDBC / JDBC technology

```
try {
    try {
        Class.forName("org.gjt.mm.mysql.Driver");
        Connection cn = null;

        try {
            cn = DriverManager.getConnection("jdbc:mysql://localhost/db2", "root", "pass");
            Statement st = null;

            try {
                st = cn.createStatement();
                ResultSet rs = null;

                try {
                    rs = st.executeQuery("SELECT * FROM users");

                    while (rs.next()) {
                        System.out.println("Name:-> " + rs.getString(1) +
                            " Phone:-> " + rs.getInt(2));
                    }
                } finally {
                    if (rs != null) {
                        rs.close();
                    } else {
                        System.out.println("Error while reading from DB");
                    }
                }
```

6.1 Технологія JDBC / JDBC technology

```
28                } finally {
29                    if (st != null) {
30                        st.close();
31                    } else {
32                        System.out.println("Statement is not created");
33                    }
34                }
35            } finally {
36                if (cn != null) {
37                    cn.close();
38                } else {
39                    System.out.println("Connection is not created");
40                }
41            }
42        } catch (ClassNotFoundException e) {
43            System.out.println("Error while loading DB driver");
44        }
45    } catch (SQLException e) {
46    } catch (IOException e) {
47    }
48
```

6.1 Технологія JDBC / JDBC technology

Ще один спосіб з'єднання з базою даних можливий з використанням файлу ресурсів **database.properties**, в якому зберігаються, як правило, шлях до БД, логін і пароль доступу

Another way to connect to the database is possible using the resource file **database.properties**, which usually stores the database path, login and access password

url=jdbc:mysql://localhost/my_db?useUnicode=true&
characterEncoding=Cp1251

driver=org.gjt.mm.mysql.Driver

user=root

password=pass

6.1 Технологія JDBC / JDBC technology

```
1  public Connection getConnection() throws SQLException {
2      ResourceBundle resource = ResourceBundle.getBundle("database");
3
4      String url = resource.getString("url");
5      String driver = resource.getString("driver");
6      String user = resource.getString("user");
7      String pass = resource.getString("password");
8
9      try {
10         Class.forName(driver).newInstance();
11     } catch (ClassNotFoundException e) {
12         throw new SQLException("Driver is not loaded!");
13     } catch (InstantiationException e) {
14         e.printStackTrace();
15     } catch (IllegalAccessException e) {
16         e.printStackTrace();
17     }
18
19     return DriverManager.getConnection(url, user, pass);
20 }
```

6.1 Технологія JDBC / JDBC technology

ResultSetMetaData rsMetaData = rs.getMetaData(); (interface)

int getColumnCount()
String getColumnName(int column)
int getColumnType(int column)

DatabaseMetaData dbMetaData = cn.getMetaData(); (interface)

String getDatabaseProductName()
String getDatabaseProductVersion()
String getDriverName()
String getUserName()
String getURL()

6.1 Технологія JDBC / JDBC technology

Для подання запитів існують ще два типи об'єктів **PreparedStatement** і **CallableStatement**. Об'єкти першого типу використовуються при виконанні часто повторюваних запитів SQL. Такий оператор попередньо готується і зберігається в об'єкті, що прискорює обмін інформацією з базою даних. Другий інтерфейс використовується для виконання збережених процедур, створених засобами самої СУБД.

There are two other types of objects **PreparedStatement** and **CallableStatement** for representing queries. Objects of the first type are used when performing frequently repeated SQL queries. Such an operator is pre-prepared and stored in the object, which speeds up the exchange of information with the database. The second interface is used to execute stored procedures created by using the DBMS itself.

6.1 Технологія JDBC / JDBC technology

Для підготовки SQL-запиту, в якому відсутні конкретні параметри, використовується метод **prepareStatement (String sql)** інтерфейсу **Connection**, який повертає об'єкт **PreparedStatement**

To prepare an SQL query that does not contain specific parameters, use the **prepareStatement (String sql)** method of the **Connection** interface, which returns a **PreparedStatement** object

Установка вхідних значень конкретних параметрів цього об'єкта проводиться за допомогою методів **setString()**, **setInt()** і подібних до них, після чого і здійснюється безпосереднє виконання запиту методами **executeUpdate()**, **executeQuery()**

The input values of specific parameters of this object are set using the methods **setString()**, **setInt()** and similar ones, after which the query is executed directly by the methods **executeUpdate()**, **executeQuery()**

6.1 Технологія JDBC / JDBC technology

```
try {
    Class.forName("org.gjt.mm.mysql.Driver");
    Connection cn = null;

    try {
        cn = DriverManager.getConnection("jdbc:mysql://localhost/db3", "root", "");

        PreparedStatement ps = null;

        String sql = "INSERT INTO emp (id, name, surname, salary) VALUES (?,?,?,?)";
        ps = cn.prepareStatement(sql);

        Rec.insert(ps, 2505, "Mike", "Call", 620);
    } finally {
        if (cn != null) {
            cn.close();
        }
    }
} catch (Exception e) {
    e.printStackTrace();
}
```

6.1 Технологія JDBC / JDBC technology

```
public static void insert(PreparedStatement ps, int id, String name, String surname, int salary)
        throws SQLException {
    ps.setInt(1, id);
    ps.setString(2, name);
    ps.setString(3, surname);
    ps.setInt(4, salary);

    ps.executeUpdate();
}
```

Інтерфейс **CallableStatement** розширює можливості інтерфейсу **PreparedStatement** і забезпечує виконання збережених процедур

The **CallableStatement** interface extends the capabilities of the **PreparedStatement** interface and ensures that stored procedures are executed

Нехай в БД існує процедура, що зберігається **getempname**, яка за унікальним для кожного запису в таблиці **employee** числом SSN буде повертати відповідне йому ім'я

Suppose there is a **getempname** stored procedure in the database, which, by the unique number of SSN for each record in the **employee** table, will return the corresponding name

6.1 Технологія JDBC / JDBC technology

Отримання імені працівника через виклик збереженої процедури
Getting the name of employee through a stored procedure call

```
String SQL = "{call getempname (?,?)}";
CallableStatement cs = conn.prepareCall(SQL);
cs.setInt(1,822301);

// реєстрація вихідного параметра / output parameter registration
cs.registerOutParameter(2, java.sql.Types.VARCHAR);
cs.execute();
String empName = cs.getString(2);
System.out.println("Employee with SSN:" + ssn + " is " + empName);
```

6.1 Технологія JDBC / JDBC technology

Для фіксації результатів роботи SQL-операторів, логічно виконуваних в рамках деякої транзакції, використовується SQL-оператор **COMMIT**. В API JDBC ця операція виконується за замовчуванням після кожного виклику методів **executeQuery()** і **executeUpdate()**.

To fix the results of the work of SQL statements that are logically executed within a certain transaction, use the SQL **COMMIT** statement. In the JDBC API, this operation is performed by default after each call to the **executeQuery()** and **executeUpdate ()** methods.

Якщо ж необхідно згрупувати запити і тільки після цього виконати операцію **COMMIT**, спочатку викликається метод **setAutoCommit (boolean param)** інтерфейсу **Connection** з параметром **false**, в результаті виконання якого поточне з'єднання з БД переходить в режим неавтоматичного підтвердження операцій.

If it is necessary to group the queries and only after that perform the **COMMIT** operation, first the **Connection** interface's **setAutoCommit(boolean param)** method is called with the parameter **false**, as a result of which the current connection to the database goes into the non-automatic confirmation of operations.

6.1 Технологія JDBC / JDBC technology

Підтверджує виконання SQL-запитів метод **commit()** інтерфейсу **Connection**, в результаті дії якого всі зміни таблиці виконуються як одна логічна дія

The **commit()** method of the **Connection** interface confirms the execution of SQL queries, as a result of which all changes to the table are made as one logical action

Якщо ж транзакція не виконана, то методом **rollback()** скасовуються дії всіх запитів SQL, починаючи від останнього виклику **commit()**

If the transaction is not completed, then the **rollback()** method cancels the actions of all SQL queries, starting from the last **commit()** call

6.1 Технологія JDBC / JDBC technology

```
1  Connection cn = null;
2
3  try {
4      cn = getConnection();
5      cn.setAutoCommit(false);
6
7      Statement st = cn.createStatement();
8
9      try {
10         // execute updates
11         // ...
12
13         cn.commit();
14     } catch (SQLException e) {
15         cn.rollback();
16
17         // print errors
18         // ...
19     } finally {
20         if (cn != null) {
21             cn.close();
22         }
23     }
24 } catch (SQLException e) {
25     // print errors
26     // ...
27 }
```

6.1 Технологія JDBC / JDBC technology

Рівні ізоляції транзакцій визначені у вигляді констант інтерфейсу **Connection** (по зростанню рівня обмеження):

Transaction isolation levels are defined as the constants of the **Connection** interface (by the level of isolation):

TRANSACTION_NONE

TRANSACTION_READ_UNCOMMITTED

TRANSACTION_READ_COMMITTED

TRANSACTION_REPEATABLE_READ

TRANSACTION_SERIALIZABLE

Метод **boolean supportsTransactionIsolationLevel(int level)** інтерфейсу **DatabaseMetaData** визначає, чи підтримується заданий рівень ізоляції транзакцій

Method **boolean supportsTransactionIsolationLevel(int level)** of the interface **DatabaseMetaData** tells whether the transaction isolation level supported

6.1 Технологія JDBC / JDBC technology

Методи інтерфейсу **Connection** визначають доступ до рівня ізоляції

Methods of the **Connection** interface provide access to the transaction isolation level

int getTransactionIsolation()

повертає поточний рівень ізоляції

returns the current isolation level

void setTransactionIsolation(int level)

встановлює потрібний рівень

sets the required isolation level

6.1 Технологія JDBC / JDBC technology

При великій кількості клієнтів, що працюють з додатком, до його бази даних виконується велика кількість запитів. Встановлення з'єднання з БД є дорогою (по необхідним ресурсам) операцією. Ефективним способом вирішення даної проблеми є організація **пулу** (pool) використовуваних з'єднань, які не зачиняються фізично, а зберігаються в черзі і надаються повторно для інших запитів.

With a large number of clients working with the application, a large number of queries are made to its database. Establishing a database connection is an expensive (by the required resources) operation. An effective way to solve this problem is to organize a **pool** of used connections that are not physically closed, but are stored in a queue and re-provided for other requests.

6.1 Технологія JDBC / JDBC technology

Пул з’єднань – це одна з стратегій надання з'єднань з додатком

Connection pooling is one of the strategies for providing connections to an application

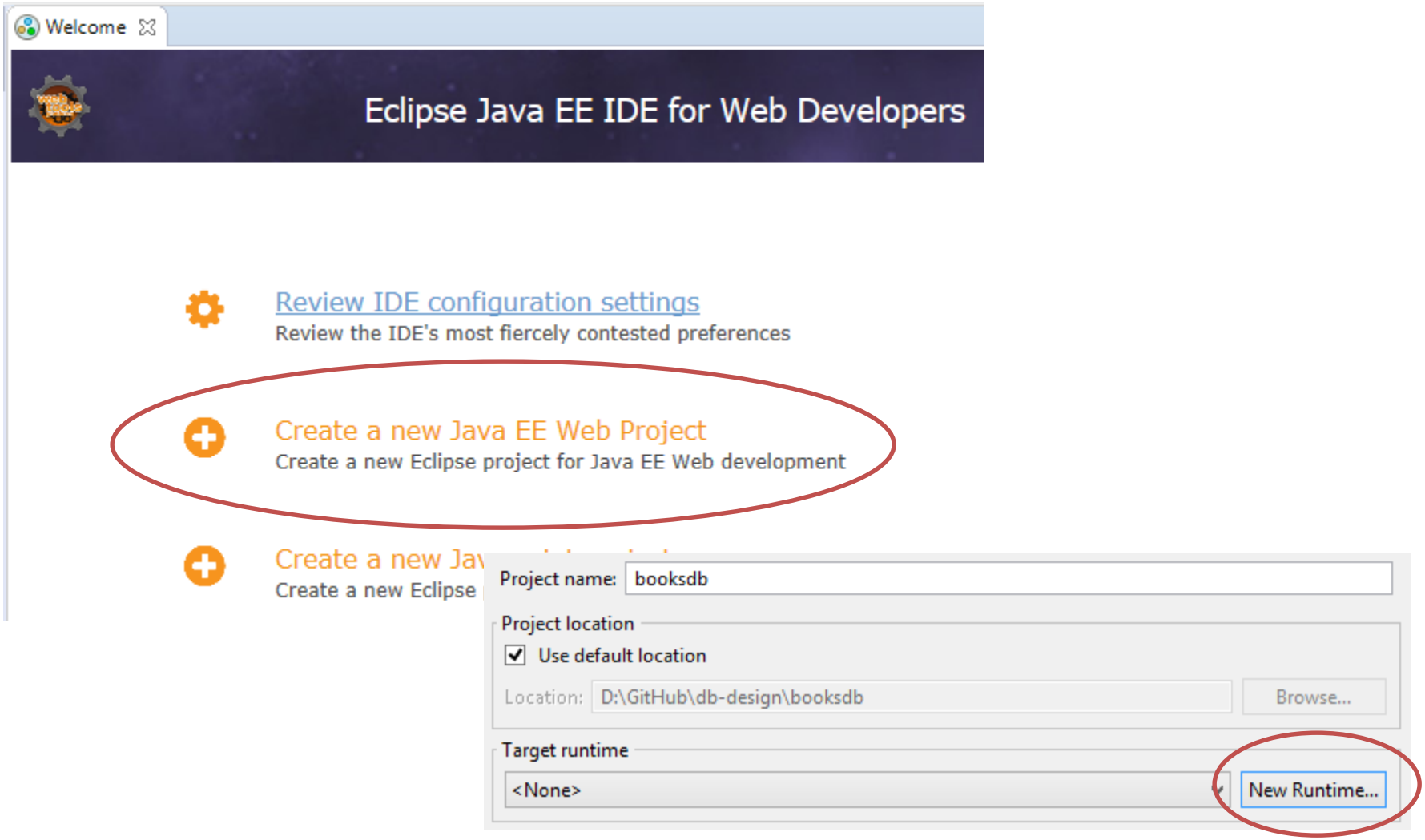
Пул з’єднань можна організувати за допомогою класу **PoolProperties** контейнера Apache Tomcat

A pool of connections can be organized using the **PoolProperties** class of the Apache Tomcat container

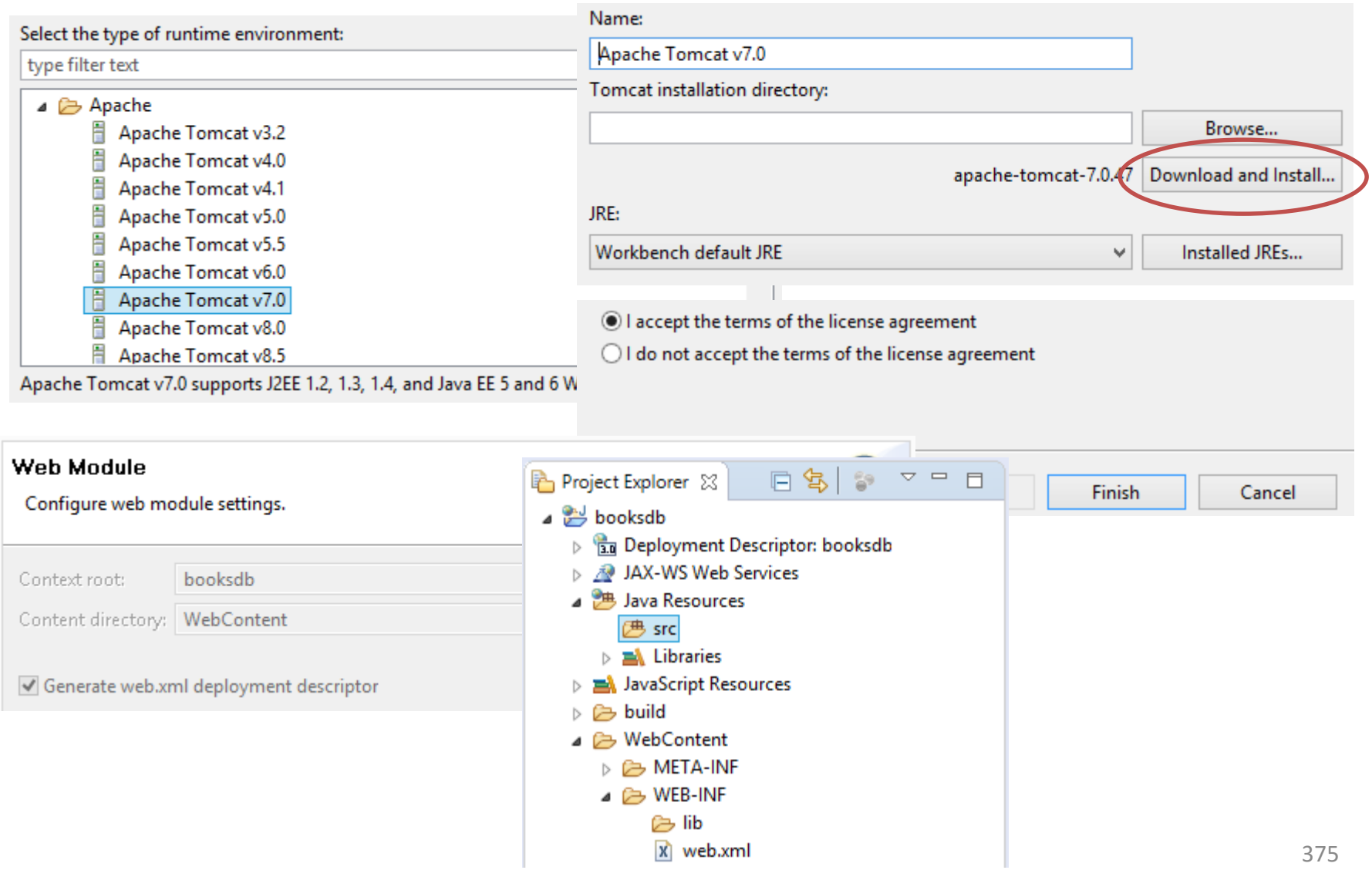
Для полегшення створення пулу з'єднань в ApacheTomcat визначено власний клас **DataSource** на основі інтерфейсу **javax.sql.DataSource**

To facilitate the creation of a connection pool, ApacheTomcat defines its own **DataSource** class based on the **javax.sql.DataSource** interface.

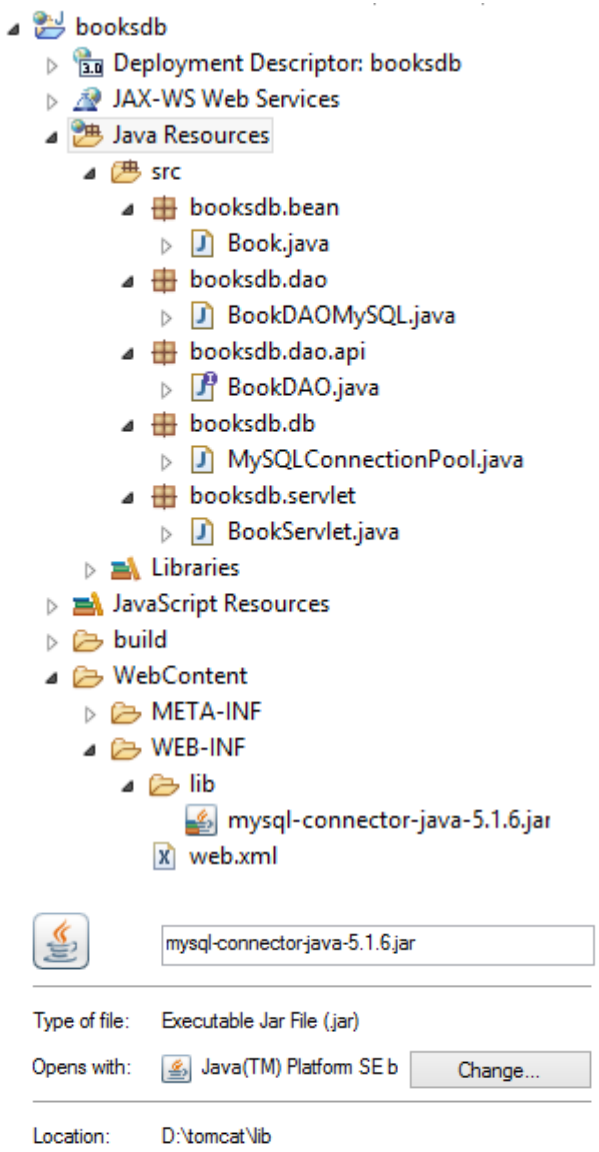
6.1 Технологія JDBC / JDBC technology



6.1 Технологія JDBC / JDBC technology



6.1 Технологія JDBC / JDBC technology



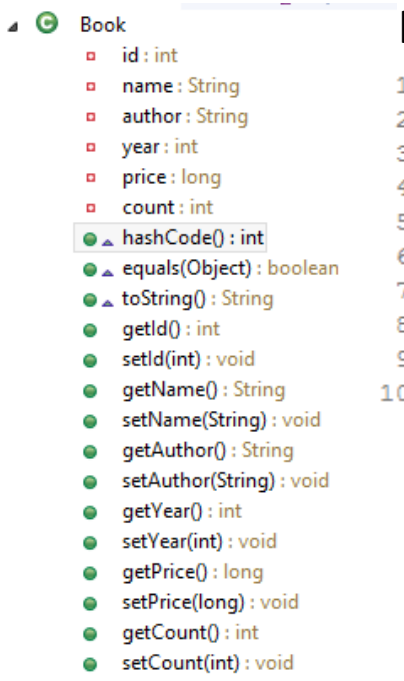
booksdb.db.MySQLConnectionPool

```
1  package booksdb.db;
2
3  import java.sql.Connection;
4  import java.sql.SQLException;
5
6  import org.apache.tomcat.jdbc.pool.DataSource;
7  import org.apache.tomcat.jdbc.pool.PoolProperties;
8
9  public class MySQLConnectionPool {
10
11     public static Connection getConnection() throws SQLException {
12         PoolProperties p = new PoolProperties();
13
14         p.setUrl("jdbc:mysql://localhost:3306/book");
15         p.setDriverClassName("com.mysql.jdbc.Driver");
16         p.setUsername("root");
17         p.setPassword("");
18
19         DataSource ds = new DataSource();
20         ds.setPoolProperties(p);
21
22         return ds.getConnection();
23     }
24 }
```

6.1 Технологія JDBC / JDBC technology

booksdb.bean.Book

```
1  package booksdb.bean;
2
3  public class Book {
4      private int id;
5      private String name;
6      private String author;
7      private int year;
8      private long price;
9      private int count;
10
```



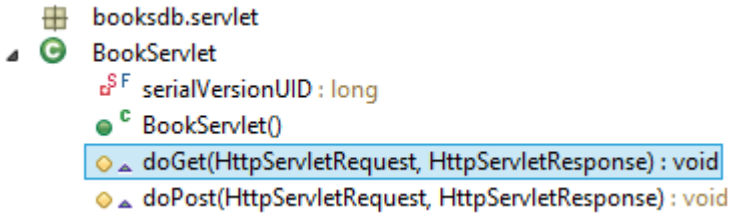
booksdb.dao.api.BookDAO

```
1  package booksdb.dao.api;
2
3  import java.util.List;
4
5  import booksdb.bean.Book;
6
7  public interface BookDAO {
8
9      List<Book> getAll();
10 }
```

booksdb.dao.BookDAOMySQL.getAll()

```
public List<Book> getAll() {
    List<Book> books = new ArrayList<Book>();
    Connection cn = null;
    try {
        cn = MySQLConnectionPool.getConnection();
        Statement st = cn.createStatement();
        try {
            ResultSet rs = st.executeQuery("SELECT * FROM books");
            while (rs.next()) {
                Book book = new Book();
                book.setId(rs.getInt("book_ID"));
                book.setName(rs.getString("b_name"));
                book.setAuthor(rs.getString("b_author"));
                book.setYear(rs.getInt("b_year"));
                book.setPrice(rs.getLong("b_price"));
                book.setCount(rs.getInt("b_count"));
                books.add(book);
            }
        } finally {
            if (cn != null) {
                cn.close();
            }
        }
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
    return books;
}
```

6.1 Технологія JDBC / JDBC technology



booksdb.servlet.BookServlet

```
16 @WebServlet("/BookServlet")
17 public class BookServlet extends HttpServlet {
18     private static final long serialVersionUID = 1L;
19
20     public BookServlet() {
21         super();
22     }
23
24     protected void doGet(HttpServletRequest request,
25             HttpServletResponse response) throws ServletException, IOException {
26         response.setContentType("text/html; charset=UTF-8");
27
28         BookDAO bookDAO = new BookDAOMySQL();
29
30         PrintWriter out = response.getWriter();
31         out.print("<h2>Books</h2>");
32
33         for (Book book : bookDAO.getAll()) {
34             out.print("<p>Book #" + book.getId() + "</br>Name: " + book.getName()
35                     "</br>Author: " + book.getAuthor() + "</br>Year: " +
36                     book.getYear() + "</br>Price: " + book.getPrice() +
37                     "</br>Count: " + book.getCount() + "</p>");
38         }
39
40         out.close();
41     }
```

6.1 Технологія JDBC / JDBC technology