

Coding Task 4

DB Integration

Connect your application to the SQL database:

4. Write code to establish a connection to your SQL database.

Create a utility class DBConnection in a package util with a static variable connection of Type Connection and a static method getConnection() which returns connection. Connection properties supplied in the connection string should be read from a property file. Create a utility class PropertyUtil which contains a static method named getPropertyString() which reads a property file containing connection details like hostname, dbname, username, password, port number **and returns a connection string**.

In a Finance Management System, establishing a reliable and flexible connection to the SQL database is crucial for storing, retrieving, and managing financial data. To ensure maintainability and scalability, the application uses a structured utility-based approach to handle database connections using external property files. This documentation outlines the implementation steps for connecting the application to the SQL database.

Task 4.1 – Database Connection

To interact with the MySQL database, a utility class named DBConnUtil is created inside the util package. It contains a static method getConnection() which initializes and returns a Connection object using JDBC. This method reads the required connection details like URL, username, password, and driver from a separate property file, making the system flexible and secure. If the database connection fails, an appropriate error message is displayed in the console.

```
import java.sql.SQLException;
import java.util.Properties;

public class DBConnUtil {
    public static Connection getConnection() {
        Connection connection = null;
        try {

            Properties properties = DBPropertyUtil.getDBProperties();

            String url = properties.getProperty("db.url");
            String user = properties.getProperty("db.username");
            String password = properties.getProperty("db.password");
            String driver = properties.getProperty("db.driver");

            Class.forName(driver);
```

```

        connection = DriverManager.getConnection(url, user, password);

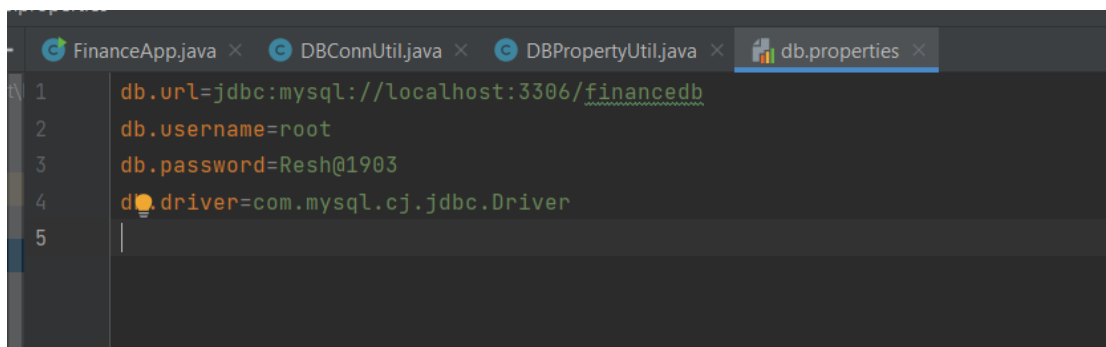
    } catch (ClassNotFoundException | SQLException e) {
        System.err.println("Database connection failed: " + e.getMessage());
    }

    return connection;
}

```

Task 4.2 – Property File

The connection-specific details such as db.url, db.username, db.password, and db.driver are externalized into a property file named db.properties, located in the resources directory. This approach separates configuration from code, allowing easy updates without modifying source files. Storing sensitive credentials in a properties file is a standard best practice for better code organization and security.



The screenshot shows an IDE with several tabs: FinanceApp.java, DBConnUtil.java, DBPropertyUtil.java, and db.properties. The db.properties tab is active, displaying the following content:

```

1 db.url=jdbc:mysql://localhost:3306/financedb
2 db.username=root
3 db.password=Resh@1903
4 db.driver=com.mysql.cj.jdbc.Driver
5

```

Task 4.3 – PropertyUtil Class

The DBPropertyUtil class is responsible for loading the connection details from the db.properties file. It provides a static method getDBProperties() which reads the file using FileInputStream and returns a Properties object containing all the necessary database credentials. This promotes modularity and avoids hard-coding sensitive information directly within the connection class.

```

package util;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

public class DBPropertyUtil {

    public static Properties getDBProperties() {
        Properties properties = new Properties();
        try (FileInputStream input = new FileInputStream("resources/db.properties")) {
            properties.load(input);
        } catch (IOException e) {
            System.err.println("Error loading database properties: " + e.getMessage());
        }
        return properties;
    }
}

```

Conclusion :

Task 4 focuses on building a clean, modular, and secure approach for managing database connections in the Finance Management System. By externalizing configuration details into a property file and encapsulating the logic within utility classes (DBConnUtil and DBPropertyUtil), the system becomes easier to maintain, scalable, and secure. This design allows for seamless changes to database credentials or drivers without altering the application logic, aligning with industry best practices for robust and flexible software development.
