

**NAME:    MAXWEL OCHIENG**

**ADMISSION NO:    20/02816**

## ADVANCED JAVA PROGRAMMING

Using real world business applications write programs to facilitate the following

### 1. Java Database Connectivity (JDBC)

20 marks

**Java Database Connectivity** or **JDBC** is a Java API(Application Programming Interface) used to connect with any table-like database, also called a **relational database**. It can be used to access databases like Oracle Database, MySQL or PostgreSQL. It is defined as part of the Java Standard Library and the APIs are provided by the *java.sql.\** packages.

This simple example reads some **Student** records from a MySQL database and prints them on the console.

*Step 1: Create a simple SQL script to create a database, a table and some records in the table*

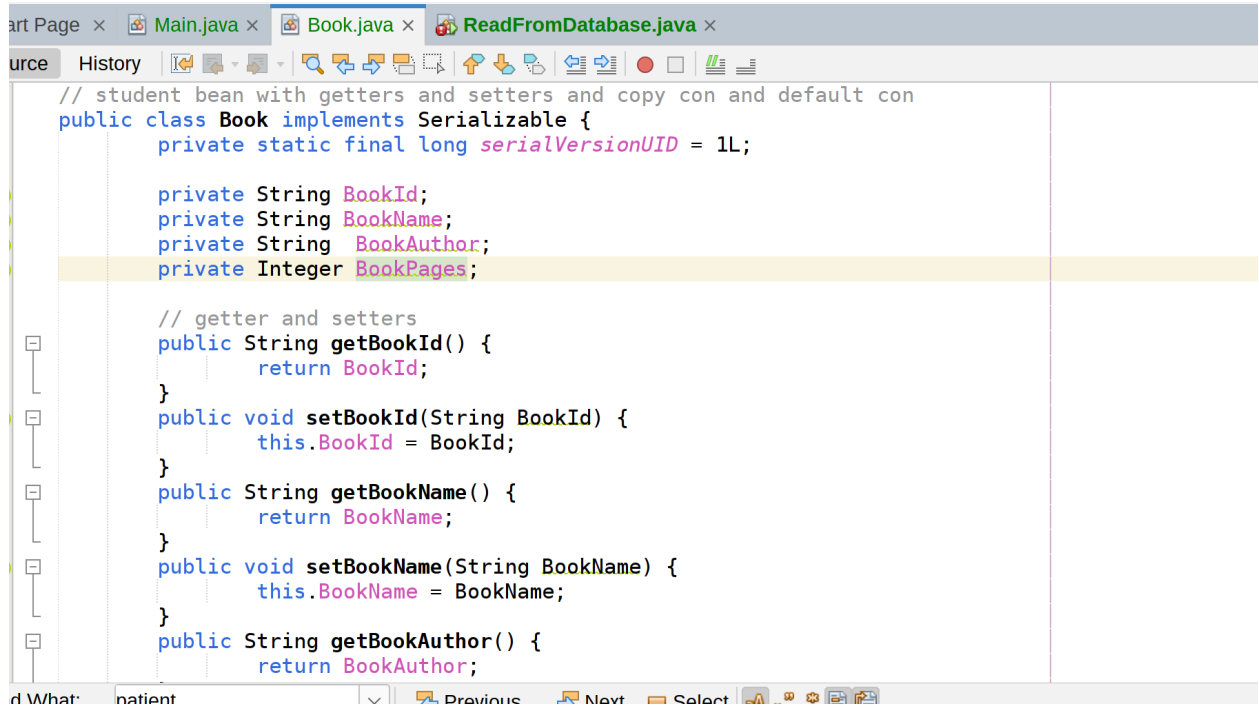
```
DROP SCHEMA IF EXISTS 'lib';
CREATE SCHEMA 'lib';
use lib;

CREATE TABLE 'Book_Details'(
  'BookID' int(2) NOT NULL,
  'BookName' varchar(32),
  'BookAuthor' varchar(32),
  'BookPages' int(4),

  CONSTRAINT PRIMARY KEY('BookID')
);

insert into Book_Details values(1,"Shuggie Bain","Douglas Stuart",289);
insert into Book_Details values(1,"Luster","Raven Leilani",300);
```

*Step 2: Create a Java POJO(Plain Old Java Object) Bean that resembles the 'Book\_Details' table we created in Step 2*

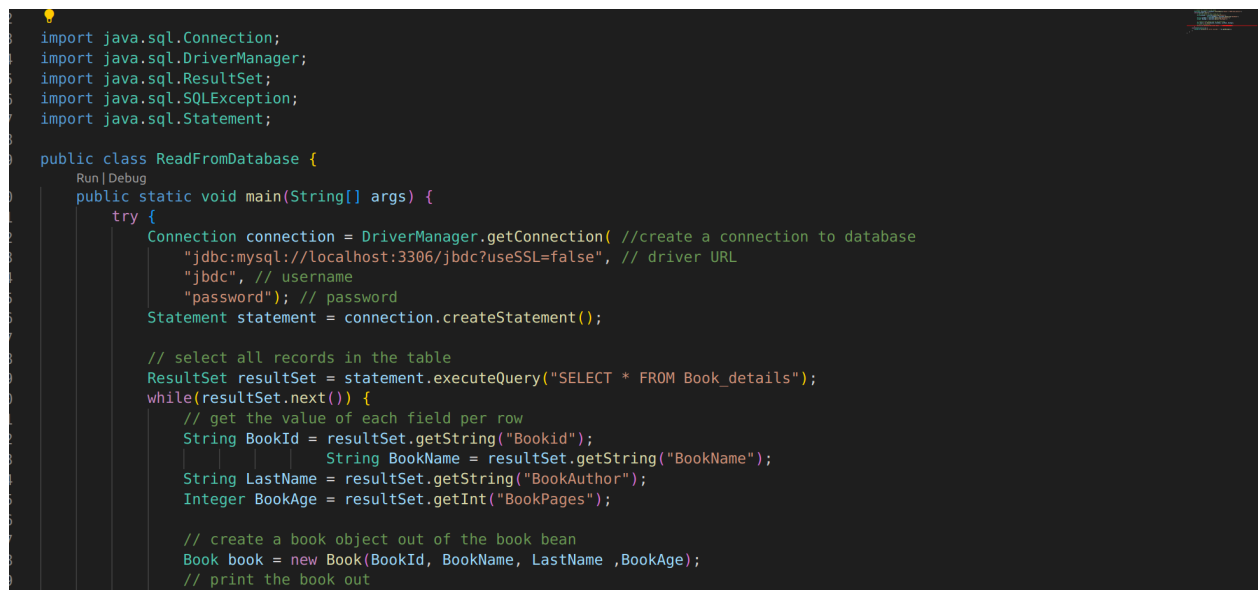


```
// student bean with getters and setters and copy con and default con
public class Book implements Serializable {
    private static final long serialVersionUID = 1L;

    private String BookId;
    private String BookName;
    private String BookAuthor;
    private Integer BookPages;

    // getter and setters
    public String getBookId() {
        return BookId;
    }
    public void setBookId(String BookId) {
        this.BookId = BookId;
    }
    public String getBookName() {
        return BookName;
    }
    public void setBookName(String BookName) {
        this.BookName = BookName;
    }
    public String getBookAuthor() {
        return BookAuthor;
    }
}
```

**Step 3: Create JDBC code to access the Book records from the table and print them out to the console**



```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class ReadFromDatabase {
    public static void main(String[] args) {
        try {
            Connection connection = DriverManager.getConnection( //create a connection to database
                "jdbc:mysql://localhost:3306/jdbc?useSSL=false", // driver URL
                "jdbc", // username
                "password"); // password
            Statement statement = connection.createStatement();

            // select all records in the table
            ResultSet resultSet = statement.executeQuery("SELECT * FROM Book_details");
            while(resultSet.next()) {
                // get the value of each field per row
                String BookId = resultSet.getString("Bookid");
                String BookName = resultSet.getString("BookName");
                String LastName = resultSet.getString("BookAuthor");
                Integer BookAge = resultSet.getInt("BookPages");

                // create a book object out of the book bean
                Book book = new Book(BookId, BookName, LastName, BookAge);
                // print the book out
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

## 2. JavaBeans

20 marks

### JavaBeans

Java beans are Java classes that meet the following conditions:

They have a no-argument constructor

It implements the serializable interface

It provides methods to get and set the properties of the class, known as **getters** and **setters**

**Example:**

*In the simple example below, we create a **Book** bean that we can award some marks and print them out to the console*

**Step 1: Create the *Book* Bean**

```
package com.maxwel.bookjavabean;

import java.io.Serializable;

/**
 * @author maxwel-ochieng
 */

public class Book implements Serializable {
    private static final long serialVersionUID = 1L;

    // encapsulated class properties(use 'private' keyword)
    private String bookId;
    private String bookName;
    private String authorName;
    private String bookgenre;
    private Integer bookpages;

    // getters help retrieve details for the object
    public String getBookId() {
        return bookId;
    }
    public void setBookId(String bookId) {
        this.bookId = bookId;
    }
    public String getBookName() {
```

**Step 2: Create getters and setters for the bean Add the default constructor**

```

package com.maxwel.bookjavabean;

import java.io.Serializable;

/**
 * @author maxwel-ochieng
 */

public class Book implements Serializable {
    private static final long serialVersionUID = 1L;

    // encapsulated class properties(use 'private' keyword)
    private String bookId;
    private String bookName;
    private String authorName;
    private String bookgenre;
    private Integer bookpages;

    // getters help retrieve details for the object
    public String getBookId() {
        return bookId;
    }
    public void setBookId(String bookId) {
        this.bookId = bookId;
    }
    public String getBookName() {
    }
    public String getAuthorName() {
        return authorName;
    }
    public void setAuthorName(String bookName) {
        this.authorName = authorName;
    }
    public String getBookGenre() {
        return bookgenre;
    }
    public void setBookGenre(String bookgenre) {
        this.bookgenre = bookgenre;
    }
    public Integer getBookPages() {
        return bookpages;
    }

    // setters help 'set' properties of the object
    public void setBookPages(Integer bookpages) {
        this.bookpages = bookpages;
    }

    // the default constructor takes no arguments
    public Book() {}
}

```

**Step 3: Create an instance of the Student class using the default constructor and use the getters and setters on it.**

```

public class JavaBeanTester {

    public static void main(String[] args)
    {
        // creating a instance of the class(object) using default constructor
        Book book = new Book();

        // using getters and setters to populate the object properties
        // using the getters of the bean
        book.setBookName( bookName: "Kidagaa");
        book.setAuthorName( bookName: "Paula");
        book.setBookId( bookId: "29987");
        book.setBookGenre( bookgenre: "Drama");
        book.setBookPages( bookpages: 657);

        // we can print out some details using the setters of the bean
        System.out.println( x: "Welcome To Maxwell's Library UseCase for JavaBeans");
        System.out.println( x: "*****");
        System.out.println("Book id: " + book.getBookId());
        System.out.println("Book name is: " + book.getBookName());
        System.out.println("Author name is : " + book.getAuthorName());
        System.out.println("Book Genre is " + book.getBookGenre());
        System.out.println("Book pages : " + book.getBookPages());

        System.out.println( x: "*****BYE*****");
    }
}

```

**Step 5: Check console for output:**

```

l --- exec-maven-plugin:3.0.0:exec (default-cli) @ javabean ---
Welcome To Maxwell's Library UseCase for JavaBeans
*****
Book id: 29987
Book name is: Kidagaa
Author name is : null
Book Genre is Drama
Book pages :657
*****BYE*****
*****

```

### 3. Remote Method Invocation

**20 marks**

Java RMI(**Remote Method Invocation**) is a technology that allows the methods of one Object Oriented Programming language make calls and receive responses from methods of another class running on a different machine. It is the OOP equivalent of RPCs(**Remote Procedure Calls**) that is used by functional programming languages like C.

**Step 1: Create an interface to house all the methods exposed in the RMI communication**

```

package com.maxwel.rmijava;

import java.rmi.Remote;
import java.rmi.RemoteException;

/**
 * @author maxwel
 * This interface will be implemented by both the client and server class.
 * It extends Remote.
 */
public interface RMIInterface extends Remote {
    /**
     * All methods defined here that would be used in the RMI operations and must be public and throw a
     */
    public String hello(String name) throws RemoteException;
}

```

### Step 2: Create the server that implements the interface

```

package com.maxwel.rmijava;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
// the server class extends UnicastRemoteObject and implements the interface we
// created earlier
public class RMIServer extends UnicastRemoteObject implements RMIInterface {
    protected RMIServer() throws RemoteException {
        super();
    }
    private static final long serialVersionUID = 1L;
    /**
     * Overriding the method defined in the RMIInterface class
     */
    public String hello(String name) throws RemoteException {
        System.err.println(name+ " is trying to connect");
        return "Server says hello to: " + name;
    }
    public static void main(String[] args) {
        try {
            // we bind the server to local host with the name 'rmi-server'
            Naming.rebind( name: "//localhost/rmi-server/", new RMIServer());
            System.out.println( x: "Server ready for use...");
        } catch (Exception ex) {
            // and print out any exceptions that may be thrown when the server is starting
            System.err.println("RMI Server Exception! Details: " + ex.getMessage());
        }
    }
}

```

### Step 3: Create a client that also implements the interface but connects to the server on the specified port

```

package com.maxwel.rmijava;

import java.rmi.Naming;
import javax.swing.JOptionPane;

public class RMIClient {
    public static void main(String[] args) {
        try {
            System.setProperty( key: "java.rmi.server.hostname", value: "192.168.0.103");

            RMIInterface lookUp = (RMIInterface) Naming.lookup( name: "//localhost/rmi-server");
            String text = JOptionPane.showInputDialog( message: "What is your name?");

            String response = lookUp.hello( name: text);
            JOptionPane.showMessageDialog( parentComponent: null, message: response);
        } catch (Exception ex) {
            System.err.println("An Error Occured Connecting to Client: " + ex.getMessage());
        }
    }
}

```

#### 4. Network programming

20 marks

Network programming allows two Java programs to communicate via a network, even if they are on different machines. This can be in the form of HTTP packets, WebSockets, TCP connections or other protocols supported by the language.

In this example, we have created a simple socket program that runs on the terminal. The server is started and waits for a client program to connect and then displays the message that the client sends to it in the terminal.

**Step 1: Server code that establishes the socket connection and a port for clients to connect to**

```

package com.maxwel.networkingjava;
import java.io.DataInputStream;
import java.io.IOException;
import java.net.*;

public class NetworkingServer {
    public static void main(String[] args) throws IOException {
        try {
            // Create the server port
            ServerSocket server = new ServerSocket(port: 10000);
            // authorize to accept in that port
            Socket socket = server.accept();
            // create an input stream to receive messages
            DataInputStream messageInputStream = new DataInputStream( in: socket.getInputStream());

            // variable to hold incoming message
            String incomingMessage = messageInputStream.readUTF().toString();

            // print out the message sent by server
            System.out.println("Message received: " + incomingMessage);

            // close the connection
            server.close();
        } catch (IOException e) {
            System.out.println("Error occured: " + e.getMessage());
        }
    }
}

```

**Step 2: Create client code that connects to the socket port opened by the server**



```

package com.maxwel.networkingjava;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
public class NetworkingClient {

    public static void main(String[] args) {
        try {
            // establish socket
            Socket socket = new Socket("localhost", 10000);

            // data output stream to send messages out
            DataOutputStream messageOutputStream = new DataOutputStream( socket.getOutputStream());

            // client sends message to the server
            messageOutputStream.writeUTF("Hello , Welcome to networking.");

            // flush the output stream
            messageOutputStream.flush();

            // close the socket
            socket.close();
        } catch (IOException ex) {
            System.out.println("Error occurred: " + ex.getMessage());
        }
    }
}

```

### Step 3: Run the server and client side by side and see the output

```

aloha-tech@alohatech-HP-EliteBook-Folio-1020-G1:~/Documents/Advanced Java/javac$ /usr/bin/env /usr/lib/jvm/java-11-openjdk-amd64/bin/java -cp /home/aloha-tech/Documents/Advanced Java/javac/javanetworking/target/classes com.maxwel.networkingjava.NetworkingClient
Error occurred: Connection refused (Connection refused)
aloha-tech@alohatech-HP-EliteBook-Folio-1020-G1:~/Documents/Advanced Java/javac$ /usr/bin/env /usr/lib/jvm/java-11-openjdk-amd64/bin/java -cp /home/aloha-tech/Documents/Advanced Java/javac/javanetworking/target/classes com.alvin.javanetworking.NetworkingServer
Error: Could not find or load main class com.alvin.javanetworking.NetworkingServer
Caused by: java.lang.ClassNotFoundException: com.alvin.javanetworking.NetworkingServer
aloha-tech@alohatech-HP-EliteBook-Folio-1020-G1:~/Documents/Advanced Java/javac$ cd /home/aloha-tech/Documents/Advanced Java/javac ; /usr/bin/env /usr/lib/jvm/java-11-openjdk-amd64/bin/java -cp /home/aloha-tech/Documents/Advanced Java/javac/javanetworking/target/classes com.maxwel.networkingjava.NetworkingServer
Message received: Hello , Welcome to networking.

```

## 5. Multimedia programming

20 marks

The multimedia library allows a java application to play audio or video media. The easiest way to get an audio playing is by using the JavaFX media classes as in the simple example below:

### Step 1: Create the code

```

package javafxmultimedia;

import java.io.File;

import javafx.application.Application;
import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;
import javafx.stage.Stage;

public class Main extends Application {
    @Override
    public void start(Stage stage) throws Exception {
        // path to the Music file
        String pathToFile =
            "/home/aloha-tech/Downloads/Poems/dist:

        // instantiate the JavaFX media class
        Media media = new Media(new File( pathname: pathToFile).toURI().toString());

        // instantiate JavaFX media player class
        MediaPlayer mediaPlayer = new MediaPlayer(media);

        // set the auto play property to true for the audio to be played
        mediaPlayer.setAutoplay(true);

        //set the stage
        stage.setTitle("Audio Player");
    }
}

```

**Step 2: Play the media by running the code above. You should see a window similar to this one with the media you linked to playing in the background.**

