1. Write a Java program that takes two integers as input from the user and performs division, handling division by zero and invalid input types.

```java
package Javaprograms;
import java.util.Scanner;
public class Q18
{
public static void main(String[] args)
{
Scanner sc = new Scanner(System.in);
try {
System.out.print("Enter the first integer: ");
int num1 = getValidInteger(sc);
System.out.print("Enter the second integer: ");
int num2 = getValidInteger(sc);
if (num2 == 0)
{
System.out.println("Error: Cannot divide by zero.");
}
else
{
int result = num1 / num2;
System.out.println("Result: " + num1 + " / " + num2 + " = " + result);
}
}
catch (Exception e)
{
System.out.println("Invalid input. \nPlease enter valid integers.");
}
finally
{
sc.close();
}
}
private static int getValidInteger(Scanner scanner)
{
while (!scanner.hasNextInt())
{
System.out.println("Invalid input. \nPlease enter a valid integer."); scanner.next();
}
return scanner.nextInt();
}
}
```

2. Create a Java program that reads from a user-specified file, implementing exception handling for file not found and I/O errors

```java
package Javaprograms;
import java.io.*;
import java.util.Scanner;
public class Q2
{
public static void main(String[] args)
{
Scanner scanner = new Scanner(System.in);
System.out.print("Enter the file path: ");
String filePath = scanner.nextLine();
try {
FileReader fileReader = new FileReader(filePath);
BufferedReader bufferedReader = new BufferedReader(fileReader); String line;
System.out.println("File contents:");
while ((line = bufferedReader.readLine()) != null)
{
System.out.println(line);
}
bufferedReader.close(); fileReader.close();
}
catch (FileNotFoundException e)
{
System.out.println("Error: The file '" + filePath + "' was not found.");
}
catch (IOException e)
{
System.out.println("Error: An I/O error occurred while reading the file.");
}
finally
{
scanner.close();
}
}
}
```

3. Create a class hierarchy for animals. Design a base class Animal with properties like name and age. Then, create two subclasses: Dog and Cat. Each subclass should have a method sound() that returns the sound the animal makes

```
package Javaprograms;
class Animal
{
protected String name;
protected int age;
public Animal(String name, int age)
{
this.name = name;
this.age = age;
}
public String sound()
{
return "Some generic animal sound";
}
public String getName()
{
return name;
}
public int getAge()
{
return age;
}
public void displayInfo()
{
System.out.println("Animal Name: " + name);
System.out.println("Animal Age: " + age); System.out.println("Animal Sound: " + sound());
}
}
class Cat extends Animal
{
public Cat(String name, int age)
{
super(name, age);
}
@Override
public String sound()
{
return "Meow";
}
}
class Dog extends Animal
{
public Dog(String name, int age)
{
super(name, age);
}
public String sound()
{
return "Bark";
}
}
class Q3
{
public static void main(String[] args)
{
Animal dog = new Dog("Buddy", 3);
Animal cat = new Cat("Whiskers", 2);
System.out.println("Dog Info:");
dog.displayInfo();
System.out.println();
System.out.println("Cat Info:");
cat.displayInfo();
}
}
```

4. Design a class hierarchy for bank accounts. Create a base class BankAccount with properties like accountNumber and balance. Then, create two subclasses: SavingsAccount and CurrentAccount. Implement methods to deposit and withdraw money, and override a method to display account details specific to each account type.

```
package Javaprograms;
class BankAccount {
   protected String accountNumber;
   protected double balance;
   // Constructor
   public BankAccount(String accountNumber, double balance) {
       this.accountNumber = accountNumber;
       this.balance = balance;
   }
   // Method to deposit money
   public void deposit(double amount) {
       if (amount > 0) {
           balance += amount;
           System.out.println("Deposited: " + amount);
       } else {
           System.out.println("Invalid deposit amount!");
       }
   }
   // Method to withdraw money
   public void withdraw(double amount) {
       if (amount > 0 && amount <= balance) {
           balance -= amount;
           System.out.println("Withdrew: " + amount);
       } else {
           System.out.println("Insufficient balance or invalid amount!");
       }
   }
   // Method to display account details (to be overridden by subclasses)
   public void displayAccountDetails() {
       System.out.println("Account Number: " + accountNumber);
```

```java
        System.out.println("Balance: " + balance);
    }
}
// Subclass: SavingsAccount
class SavingsAccount extends BankAccount {
    private double interestRate;
    // Constructor
    public SavingsAccount(String accountNumber, double balance, double interestRate) {
        super(accountNumber, balance);
        this.interestRate = interestRate;
    }
    // Method to calculate interest
    public void calculateInterest() {
        double interest = balance * (interestRate / 100);
        System.out.println("Interest earned: " + interest);
    }
    // Override displayAccountDetails
    @Override
    public void displayAccountDetails() {
        super.displayAccountDetails();
        System.out.println("Account Type: Savings Account");
        System.out.println("Interest Rate: " + interestRate + "%");
    }
}
// Subclass: CurrentAccount
class CurrentAccount extends BankAccount {
    private double overdraftLimit;
    // Constructor
    public CurrentAccount(String accountNumber, double balance, double overdraftLimit) {
        super(accountNumber, balance);
        this.overdraftLimit = overdraftLimit;
    }
    // Override withdraw method to account for overdraft
    @Override
    public void withdraw(double amount) {
        if (amount > 0 && (balance - amount) >= -overdraftLimit) {
            balance -= amount;
            System.out.println("Withdrew: " + amount);
        } else {
            System.out.println("Overdraft limit exceeded or invalid amount!");
        }
    }
    // Override displayAccountDetails
    @Override
    public void displayAccountDetails() {
        super.displayAccountDetails();
        System.out.println("Account Type: Current Account");
        System.out.println("Overdraft Limit: " + overdraftLimit);
    }
}
// Main class to demonstrate functionality
public class Q4 {
    public static void main(String[] args) {
        // Create a SavingsAccount
        SavingsAccount savings = new SavingsAccount("SA12345", 1000.0, 5.0);
        savings.deposit(500);
        savings.withdraw(300);
        savings.calculateInterest();
        savings.displayAccountDetails();
        System.out.println();
        // Create a CurrentAccount
        CurrentAccount current = new CurrentAccount("CA12345", 2000.0, 1000.0);
        current.deposit(1000);
        current.withdraw(2500);
        current.displayAccountDetails();
    }
}
```

Q5) Develop a class hierarchy for geometric shapes. Create a base class Shape with a method area(). Then, implement two subclasses: Circle and Rectangle. Each subclass should have a constructorto initialize its dimensions and override the area() method tocalculate the area ofthe shape.

```
package Javaprograms;
abstract class Shape
{
public abstract double area();
}
class Circle extends Shape
{
private double radius; public
Circle(double radius)
{
this.radius = radius;
}
@Override
public double area()
{
return Math.PI * radius * radius;
}
}
class Rectangle extends Shape
{
private double width;
private double height;
public Rectangle(double width, double height)
{
this.width = width;
this.height = height;
}
@Override
public double area()
{
return width * height;
}
}
public class Q5
{
public static void main(String[] args)
{
Shape circle = new Circle(5.0);
Shape rectangle = new Rectangle(4.0, 6.0);
System.out.println("Area of Circle: " + circle.area());
System.out.println("Area ofRectangle: " + rectangle.area());
}
}
```

Q6) Implement a Java program demonstrating the use of abstract classes and interfaces in a banking application scenario. Define classes Account (abstract class),SavingsAccount, and CurrentAccount implementing different interfaces for operations like deposit, withdraw, and calculateInterest.

```
package Javaprograms;
//Interface for deposit operation
interface Depositable {
void deposit(double amount);
}
//Interface for withdraw operation
interface Withdrawable {
void withdraw(double amount);
}
//Interface for calculating interest
interface InterestCalculable {
void calculateInterest();
}
//Abstract class: Account
abstract class Account implements Depositable, Withdrawable {
protected String accountNumber;
protected double balance;
// Constructor
public Account(String accountNumber, double balance) {
    this.accountNumber = accountNumber;
    this.balance = balance;
}
// Abstract method to display account details
public abstract void displayAccountDetails();
}
//SavingsAccount class implementing InterestCalculable
class SavingsAccount extends Account implements InterestCalculable {
private double interestRate;
// Constructor
public SavingsAccount(String accountNumber, double balance, double interestRate) {
    super(accountNumber, balance);
    this.interestRate = interestRate;
}
// Implement deposit
@Override
public void deposit(double amount) {
    if (amount > 0) {
        balance += amount;
        System.out.println("Deposited: " + amount);
    } else {
        System.out.println("Invalid deposit amount!");
    }
}
// Implement withdraw
@Override
public void withdraw(double amount) {
    if (amount > 0 && amount <= balance) {
        balance -= amount;
        System.out.println("Withdrew: " + amount);
    } else {
        System.out.println("Insufficient balance or invalid amount!");
    }
}
// Implement calculateInterest
@Override
public void calculateInterest() {
    double interest = balance * (interestRate / 100);
    System.out.println("Interest earned: " + interest);
}
// Override displayAccountDetails
@Override
public void displayAccountDetails() {
    System.out.println("Savings Account");
    System.out.println("Account Number: " + accountNumber);
    System.out.println("Balance: " + balance);
    System.out.println("Interest Rate: " + interestRate + "%");
}
}
//CurrentAccount class
class CurrentAccount extends Account {
private double overdraftLimit;
// Constructor
public CurrentAccount(String accountNumber, double balance, double overdraftLimit) {
    super(accountNumber, balance);
    this.overdraftLimit = overdraftLimit;
}
// Implement deposit
@Override
public void deposit(double amount) {
    if (amount > 0) {
        balance += amount;
        System.out.println("Deposited: " + amount);
    } else {
        System.out.println("Invalid deposit amount!");
    }
}
// Implement withdraw
@Override
public void withdraw(double amount) {
    if (amount > 0 && (balance - amount) >= -overdraftLimit) {
        balance -= amount;
        System.out.println("Withdrew: " + amount);
    } else {
        System.out.println("Overdraft limit exceeded or invalid amount!");
    }
}
// Override displayAccountDetails
@Override
public void displayAccountDetails() {
    System.out.println("Current Account");
```

```java
    System.out.println("Account Number: " + accountNumber);
    System.out.println("Balance: " + balance);
    System.out.println("Overdraft Limit: " + overdraftLimit);
}
}
//Main class to demonstrate functionality
public class Q6_1 {
public static void main(String[] args) {
    // Create a SavingsAccount
    SavingsAccount savings = new SavingsAccount("SA12345", 1000.0, 5.0);
    savings.deposit(500);
    savings.withdraw(300);
    savings.calculateInterest();
    savings.displayAccountDetails();
    System.out.println();
    // Create a CurrentAccount
    CurrentAccount current = new CurrentAccount("CA12345", 2000.0, 1000.0);
    current.deposit(1000);
    current.withdraw(2500);
    current.displayAccountDetails();
}
}
```

Q7) Implement a Java program to demonstrate multithreading using the Runnable interface forprinting numbers 1 to 10 using two threads.

```java
package Javaprograms;
class PrintNumbers implements Runnable {
   private int start;
   // Constructor to initialize the starting number
   public PrintNumbers(int start) {
       this.start = start;
   }
   @Override
   public void run() {
       for (int i = start; i <= 10; i += 2) {
           System.out.println(i);
           try {
               Thread.sleep(1000); // Pause for 1 second
           } catch (InterruptedException e) {
               System.out.println("Thread interrupted: " + e);
           }
       }
   }
}
public class Q6 {
   public static void main(String[] args) {
       // Create Runnable objects for odd and even numbers
       Runnable oddNumbers = new PrintNumbers(1); // Print odd numbers
       Runnable evenNumbers = new PrintNumbers(2); // Print even numbers
       // Create threads using the Runnable objects
       Thread thread1 = new Thread(oddNumbers);
       Thread thread2 = new Thread(evenNumbers);
       // Start both threads
       thread1.start();
       thread2.start();
       // Wait for both threads to complete
       try {
           thread1.join();
           thread2.join();
       } catch (InterruptedException e) {
           System.out.println("Main thread interrupted: " + e);
       }
       System.out.println("Finished printing numbers.");
   }
}
```

Q8) Write a Java program that creates two threads. The first thread should printnumbers from 1 to 10 with a delay of 500milliseconds between each number.Thesecond thread should printthe lettersfrom 'A' to'J' with adelay of 700millisecondsbetween each letter.Use theThread classto create the threads.

```
package Javaprograms;
class LetterThread extends Thread {
@Override
public void run() {
for (char letter = 'A'; letter <= 'J'; letter++) {System.out.println(letter);
try {
// Sleep for 700milliseconds between prints
Thread.sleep(700);
} catch (InterruptedException e) {
System.out.println(e);
}
}
}
}
class NumberThread extends Thread
{
@Override
public void run()
{
for (int i = 1; i <= 10; i++)
{
System.out.println(i);try
{
Thread.sleep(500);
}
catch (InterruptedException e)
{
System.out.println(e);
}}}}
public class Q8
{
public static void main(String[] args)
{
Thread numberThread = new NumberThread();
Thread letterThread = new LetterThread();
letterThread.start();
numberThread.start();
try
{
numberThread.join();
letterThread.join();
}
catch (InterruptedException e)
{
System.out.println(e);
}
}}
```

Q9) Create a Java program that uses multiple threads to increment a shared counter.Implement a classCounterwith a synchronized method increment()that increases the counter by 1.Create three threads that each increment the counter 1000 times.After all threads finish, print the final value of the counter to ensure it is correct.

```
package Javaprograms;
class Counter {
   private int count = 0;
   // Synchronized method to increment the counter
   public synchronized void increment() {
       count++;
   }
   // Method to get the current value of the counter
   public int getCount() {
       return count;
   }
}
public class Q9 {
   public static void main(String[] args) {
       Counter counter = new Counter();
       // Define a Runnable task for incrementing the counter
       Runnable task = () -> {
           for (int i = 0; i < 1000; i++) {
               counter.increment();
           }
       };
       // Create three threads
       Thread thread1 = new Thread(task);
       Thread thread2 = new Thread(task);
       Thread thread3 = new Thread(task);
       // Start the threads
       thread1.start();
       thread2.start();
       thread3.start();
       // Wait for all threads to complete
       try {
           thread1.join();
           thread2.join();
           thread3.join();
       } catch (InterruptedException e) {
           System.out.println("Thread interrupted: " + e);
       }
       // Print the final value of the counter
       System.out.println("Final counter value: " + counter.getCount());
   }
}
```

Q10)Design a simpleGUI application usingSwing components that includes a JFramewith a JLabel, a JTextField, and a JButton. When the button is clicked, the text entered in the text field should be displayed in the labelCreate a JFrame. Add a JLabel to display instructions. Add a JTextField for user input. Add a JButton to trigger the action. Implement an ActionListener for the button to update the label with the text from the textfield.

```
package Javaprograms;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Q10{
public static void main(String[] args)
{
JFrame frame = new JFrame("Simple GUI Application");frame.setSize(400,200);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setLayout(new FlowLayout());
JLabel label = new JLabel("Enter text and click the button:");
JTextField textField =new JTextField(20);
JButton button = new JButton("Display Text");
button.addActionListener(new ActionListener()
{
@Override
public void actionPerformed(ActionEvent e)
{
label.setText("You entered: "+textField.getText());
}
});
frame.add(label);
frame.add(textField);
frame.add(button);
frame.setVisible(true);
}}
```

Q11) Experiment with different layout managers in Java to understand their behavior. Create aJFrame with multiple JButtons arranged using different layout managers such as BorderLayout, FlowLayout, GridLayout, and BoxLayout.

```
package Javaprograms;
import javax.swing.*;
import java.awt.*;
public class Q11 {
  public static void main(String[] args) {
     JFrame frame = new JFrame("Layout Manager Experiment");
     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     frame.setSize(600, 400);
     frame.setLayout(new GridLayout(2, 2)); // A grid layout for main frame to display different layouts
     // Panel with BorderLayout
     JPanel borderPanel = new JPanel(new BorderLayout());
     borderPanel.add(new JButton("North"), BorderLayout.NORTH);
     borderPanel.add(new JButton("South"), BorderLayout.SOUTH);
     borderPanel.add(new JButton("East"), BorderLayout.EAST);
     borderPanel.add(new JButton("West"), BorderLayout.WEST);
     borderPanel.add(new JButton("Center"), BorderLayout.CENTER);
     // Panel with FlowLayout
     JPanel flowPanel = new JPanel(new FlowLayout());
     flowPanel.add(new JButton("Button 1"));
     flowPanel.add(new JButton("Button 2"));
     flowPanel.add(new JButton("Button 3"));
     flowPanel.add(new JButton("Button 4"));
     flowPanel.add(new JButton("Button 5"));
     // Panel with GridLayout
     JPanel gridPanel = new JPanel(new GridLayout(3, 2)); // 3 rows and 2 columns
     gridPanel.add(new JButton("Button 1"));
     gridPanel.add(new JButton("Button 2"));
     gridPanel.add(new JButton("Button 3"));
     gridPanel.add(new JButton("Button 4"));
     gridPanel.add(new JButton("Button 5"));
     gridPanel.add(new JButton("Button 6"));
     // Panel with BoxLayout
     JPanel boxPanel = new JPanel();
     boxPanel.setLayout(new BoxLayout(boxPanel, BoxLayout.Y_AXIS));
     boxPanel.add(new JButton("Button 1"));
     boxPanel.add(new JButton("Button 2"));
     boxPanel.add(new JButton("Button 3"));
     boxPanel.add(new JButton("Button 4"));
     boxPanel.add(new JButton("Button 5"));
     // Add panels to the main frame
     frame.add(borderPanel);
     frame.add(flowPanel);
     frame.add(gridPanel);
     frame.add(boxPanel);
     frame.setVisible(true);
  }
}
```

Q12) Develop a menu-driven GUI application using Swing components. The application should include amenu barwith optionsforFile (with sub-options New,Open, Save, SaveAs,Exit) andEdit(with sub-optionsCut,Copy, Paste).Implement basic functionalitiesfor each menu option. Create a JFrame. Add a JMenuBar. Add JMenu itemsforFile andEdit. Add JMenuItems for the sub-options under each menu. ImplementActionListenersfor eachmenu itemto performthe respective actions(e.g., displaya dialog for New/Open,save a file for Save, exit the application forExit, etc.).

```
package Javaprograms;
import javax.swing.*;
import java.awt.event.*;
public class Q12 {
  public static void main(String[] args) {
      // Create JFrame
      JFrame frame = new JFrame("Menu-Driven Application");
      frame.setSize(400, 400);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      // Create JMenuBar
      JMenuBar menuBar = new JMenuBar();
      // File menu
      JMenu fileMenu = new JMenu("File");
      JMenuItem newItem = new JMenuItem("New");
      JMenuItem openItem = new JMenuItem("Open");
      JMenuItem saveItem = new JMenuItem("Save");
      JMenuItem exitItem = new JMenuItem("Exit");

      fileMenu.add(newItem);
      fileMenu.add(openItem);
      fileMenu.add(saveItem);
      fileMenu.addSeparator();
      fileMenu.add(exitItem);
      // Edit menu
      JMenu editMenu = new JMenu("Edit");
      JMenuItem cutItem = new JMenuItem("Cut");
      JMenuItem copyItem = new JMenuItem("Copy");
      JMenuItem pasteItem = new JMenuItem("Paste");
      editMenu.add(cutItem);
      editMenu.add(copyItem);
      editMenu.add(pasteItem);
      // Add menus to menu bar
      menuBar.add(fileMenu);
      menuBar.add(editMenu);
      // Set the menu bar to the frame
      frame.setJMenuBar(menuBar);
      // ActionListeners for each menu item
      newItem.addActionListener(e -> JOptionPane.showMessageDialog(frame, "New option
selected"));
      openItem.addActionListener(e -> JOptionPane.showMessageDialog(frame, "Open option
selected"));
      saveItem.addActionListener(e -> JOptionPane.showMessageDialog(frame, "Save option
selected"));
      exitItem.addActionListener(e -> System.exit(0));
      cutItem.addActionListener(e -> JOptionPane.showMessageDialog(frame, "Cut option selected"));
```

```
        copyItem.addActionListener(e -> JOptionPane.showMessageDialog(frame, "Copy option
selected"));
        pasteItem.addActionListener(e -> JOptionPane.showMessageDialog(frame, "Paste option
selected"));
        // Make the frame visible
        frame.setVisible(true);
    }
}
```

Q13) Develop a Java program that demonstrates basic event handling using buttons. Create a JFramewith twobuttonslabeled "Button 1" and "Button 2".When "Button1" is clicked, displayamessage saying "Button 1 clicked!" andwhen "Button 2" is clicked, display amessage saying"Button 2 clicked!" Create a JFrame. Add two JButtons with labels "Button 1" and "Button 2". ImplementActionListenersfor each button to handle the click events.Display appropriatemessages when each button is clicked.

```
package Javaprograms;
import javax.swing.*;
import java.awt.event.*;
public class Q13{
    public static void main(String[] args) {
        // Create JFrame
        JFrame frame = new JFrame("Button Event Handling");
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(null);
        // Create two buttons
        JButton button1 = new JButton("Button 1");
        button1.setBounds(50, 50, 100, 30);
        JButton button2 = new JButton("Button 2");
        button2.setBounds(150, 50, 100, 30);
        // Add ActionListener for Button 1
        button1.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(frame, "Button 1 clicked!");
            }
        });
        // Add ActionListener for Button 2
        button2.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(frame, "Button 2 clicked!");
            }
        });
        // Add buttons to the frame
        frame.add(button1);
        frame.add(button2);
        // Set frame visibility
        frame.setVisible(true);
```

```
  }}
```

Q14) Develop a Java programthat demonstrates customevents and listeners.Createa scenario where an alarmsystemistriggeredwhen a button is pressed.Implement customevent classes and listenersto handle the alarmeventCreate a JFrame. Add a JButton labeled "Trigger Alarm". Define a customevent class(e.g.,AlarmEvent) and a corresponding listenerinterface(e.g., AlarmListener). ImplementtheAlarmListenerinterface in a classresponsible for handlingthe alarmevent. Trigger the custom event when the "Trigger Alarm" button is pressed. Display a message or perform an action when the alarm event is triggered.

```java
package Javaprograms;
import javax.swing.*;  // For JFrame, JButton, and other Swing components
import java.awt.event.*;  // For ActionListener and ActionEvent
// 1. Define the custom AlarmEvent class
class AlarmEvent extends java.util.EventObject {
  public AlarmEvent(Object source) {
     super(source);
  }
}
// 2. Define the AlarmListener interface
interface AlarmListener extends java.util.EventListener {
  void alarmTriggered(AlarmEvent event);
}
// 3. Create the AlarmHandler class that implements the AlarmListener interface
class AlarmHandler implements AlarmListener {
  @Override
  public void alarmTriggered(AlarmEvent event) {
     JOptionPane.showMessageDialog(null, "ALARM TRIGGERED!");
  }
}
// 4. Main application that creates the JFrame and handles the event
public class Q14 {
  public static void main(String[] args) {
     JFrame frame = new JFrame("Alarm System");  // JFrame to display the window
     JButton triggerButton = new JButton("Trigger Alarm");  // Button to trigger the alarm
     // Create the AlarmHandler object to handle the event
     AlarmHandler handler = new AlarmHandler();

     // ActionListener for the button that triggers the alarm
     triggerButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
           // Trigger the custom alarm event
           AlarmEvent event = new AlarmEvent(this);
           handler.alarmTriggered(event);
        }
     });  // Set up the JFrame layout and visibility
     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     frame.setSize(300, 150);  // Set the window size
     frame.setLayout(null);  // Set layout to null for manual component placement
     triggerButton.setBounds(50, 50, 200, 40);  // Set button position and size
     frame.add(triggerButton);  // Add the button to the frame
     frame.setVisible(true);  // Make the window visible
  }}
```
15. Develop a Java application to perform CRUD operations on a student database using JDBC.Create a database schema for a student table with fields like student_id, name, age, and grade. Establish a JDBC connection to the database. Write SQL queries to create the student table, insert sample data, update records, and delete records.Implement exception handling to manage SQL exceptions.Execute the Java program to demonstrate CRUD operations.

```java
package Q15;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
public class StudentCRUDApp {
  // Database connection details datadase name with logged user and password
  private static final String DB_URL = "jdbc:mysql://localhost:3306/studentdb";
  private static final String DB_USER = "root";
  private static final String DB_PASSWORD = "system"; // Change as needed
  // Student Model Class
  static class Student {
     private int studentId;
     private String name;
     private int age;
     private String grade;
     public Student(int studentId, String name, int age, String grade) {
        this.studentId = studentId;
        this.name = name;
        this.age = age;
        this.grade = grade;
     }
     public int getStudentId() {
        return studentId;
     }
     public void setStudentId(int studentId) {
        this.studentId = studentId;
     }
     public String getName() {
        return name;
     }
     public void setName(String name) {
        this.name = name;
     }
     public int getAge() {
        return age;
     }
     public void setAge(int age) {
        this.age = age;
     }
     public String getGrade() {
        return grade;
     }
```

```
    public void setGrade(String grade) {
        this.grade = grade;
    }
}
// Database Connection Utility
public static Connection getConnection() throws SQLException {
    try {
        // Load the JDBC driver for MySQL
        Class.forName("com.mysql.cj.jdbc.Driver");
        // Establish connection to the database
        return DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
    } catch (ClassNotFoundException e) {
        throw new SQLException("JDBC Driver not found", e);
    }
}
// CRUD Operations for Student
public static class StudentDAO {
    private Connection connection;
    public StudentDAO() {
        try {
            connection = getConnection();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    // Create a new student
    public void createStudent(Student student) {
        String query = "INSERT INTO student (name, age, grade) VALUES (?, ?, ?)";
        try (PreparedStatement ps = connection.prepareStatement(query)) {
            ps.setString(1, student.getName());
            ps.setInt(2, student.getAge());
            ps.setString(3, student.getGrade());
            ps.executeUpdate();
            System.out.println("Student added successfully!");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    // Read all students
    public List<Student> getAllStudents() {
        List<Student> students = new ArrayList<>();
        String query = "SELECT * FROM student";
        try (Statement stmt = connection.createStatement();
             ResultSet rs = stmt.executeQuery(query)) {
            while (rs.next()) {
                int id = rs.getInt("student_id");
                String name = rs.getString("name");
                int age = rs.getInt("age");
                String grade = rs.getString("grade");
                students.add(new Student(id, name, age, grade));
            }
        } catch (SQLException e) {
```

```java
            e.printStackTrace();
        }
        return students;
    }
    // Update student information
    public void updateStudent(Student student) {
        String query = "UPDATE student SET name = ?, age = ?, grade = ? WHERE student_id = ?";
        try (PreparedStatement ps = connection.prepareStatement(query)) {
            ps.setString(1, student.getName());
            ps.setInt(2, student.getAge());
            ps.setString(3, student.getGrade());
            ps.setInt(4, student.getStudentId());
            ps.executeUpdate();
            System.out.println("Student updated successfully!");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    // Delete student by ID
    public void deleteStudent(int studentId) {
        String query = "DELETE FROM student WHERE student_id = ?";
        try (PreparedStatement ps = connection.prepareStatement(query)) {
            ps.setInt(1, studentId);
            ps.executeUpdate();
            System.out.println("Student deleted successfully!");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
// Main Application for interacting with the user
public static void main(String[] args) {
    StudentDAO studentDAO = new StudentDAO();
    Scanner scanner = new Scanner(System.in);
    int choice;
    do {
        System.out.println("\n1. Add Student");
        System.out.println("2. View All Students");
        System.out.println("3. Update Student");
        System.out.println("4. Delete Student");
        System.out.println("5. Exit");
        System.out.print("Enter your choice: ");
        choice = scanner.nextInt();
        switch (choice) {
            case 1:
                // Add a new student
                System.out.print("Enter Name: ");
                String name = scanner.next();
                System.out.print("Enter Age: ");
                int age = scanner.nextInt();
                System.out.print("Enter Grade: ");
                String grade = scanner.next();
```

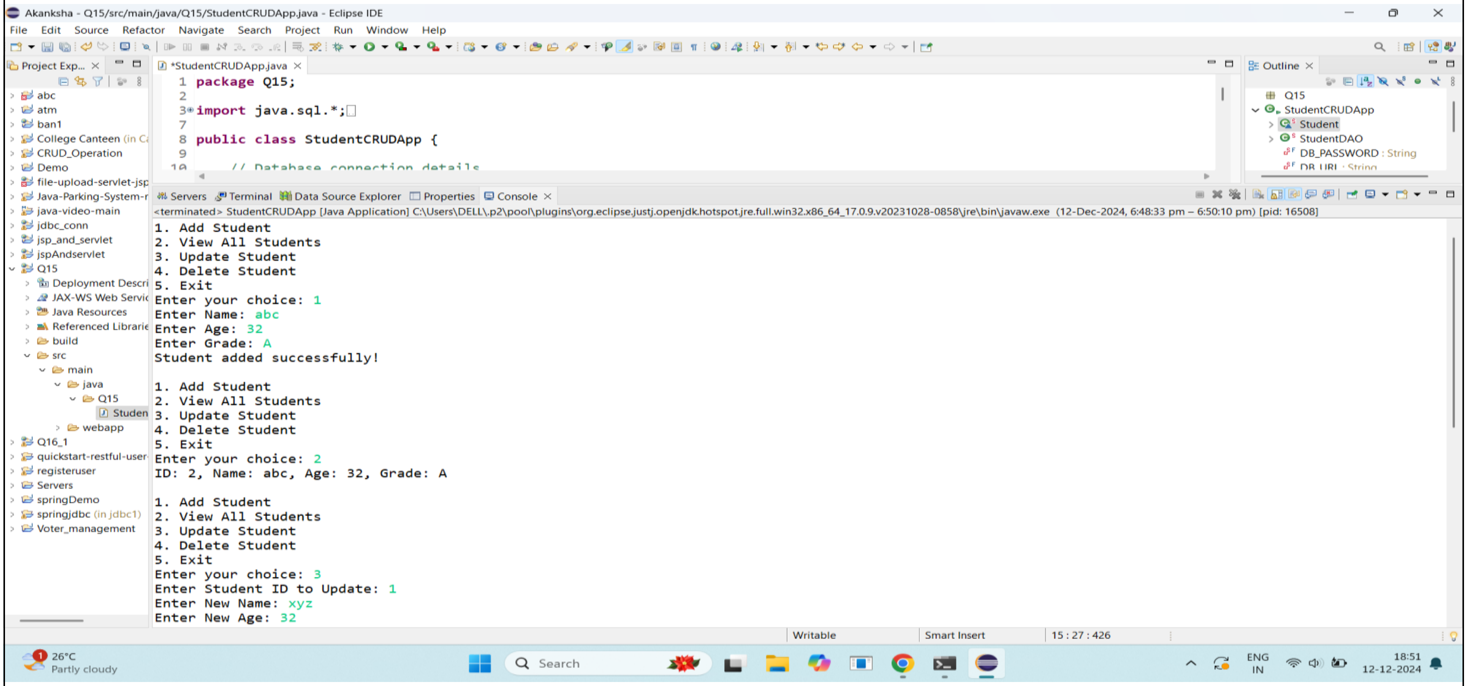
```
                    Student student = new Student(0, name, age, grade);
                    studentDAO.createStudent(student);
                    break;
                case 2:
                    // View all students
                    List<Student> students = studentDAO.getAllStudents();
                    for (Student s : students) {
                        System.out.println("ID: " + s.getStudentId() + ", Name: " + s.getName() + ", Age: " +
s.getAge() + ", Grade: " + s.getGrade());
                    }
                    break;
                case 3:
                    // Update student information
                    System.out.print("Enter Student ID to Update: ");
                    int studentIdToUpdate = scanner.nextInt();
                    System.out.print("Enter New Name: ");
                    String newName = scanner.next();
                    System.out.print("Enter New Age: ");
                    int newAge = scanner.nextInt();
                    System.out.print("Enter New Grade: ");
                    String newGrade = scanner.next();
                    Student updatedStudent = new Student(studentIdToUpdate, newName, newAge,
newGrade);
                    studentDAO.updateStudent(updatedStudent);
                    break;
                case 4:
                    // Delete a student
                    System.out.print("Enter Student ID to Delete: ");
                    int studentIdToDelete = scanner.nextInt();
                    studentDAO.deleteStudent(studentIdToDelete);
                    break;
                case 5:
                    // Exit
                    System.out.println("Exiting...");
                    break;
                default:
                    System.out.println("Invalid choice! Please try again.");
            }
        } while (choice != 5);
        scanner.close();
    }
}
```

15. Develop a Java application to perform CRUD operations on a student database using JDBC.Create a database schema for a student table with fields like student_id, name, age, and grade. Establish a JDBC connection to the database. Write SQL queries to create the student table, insert sample data, update records, and delete records.Implement exception handling to manage SQL exceptions.Execute the Java program to demonstrate CRUD operations.

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
public class StudentCRUDApp {
 // Database connection details
   private static final String DB_URL = "jdbc:mysql://localhost:3306/studentdb";
   private static final String DB_USER = "root";
   private static final String DB_PASSWORD = "system"; // Change as needed
  // Student Model Class
   static class Student {
       private int studentId;
       private String name;
       private int age;
       private String grade;
  public Student(int studentId, String name, int age, String grade) {
          this.studentId = studentId;
          this.name = name;
          this.age = age;
          this.grade = grade;}
       public int getStudentId() {
          return studentId;}
public void setStudentId(int studentId) {
          this.studentId = studentId;}
       public String getName() {
          return name }
 public void setName(String name) {
          this.name = name; }
public int getAge() {
 return age;}
 public void setAge(int age) {
          this.age = age;}
   public String getGrade() {
          return grade;
      }
 public void setGrade(String grade) {
          this.grade = grade;
      }
   }
// Database Connection Utility
   public static Connection getConnection() throws SQLException {
      try {
          // Load the JDBC driver for MySQL
          Class.forName("com.mysql.cj.jdbc.Driver");
          // Establish connection to the database
          return DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
      } catch (ClassNotFoundException e) {
          throw new SQLException("JDBC Driver not found", e);
      }
   }  // CRUD Operations for Student
   public static class StudentDAO {
      private Connection connection;
 public StudentDAO() {
          try {
              connection = getConnection();
          } catch (SQLException e) {
              e.printStackTrace();}}
 // Create a new student
      public void createStudent(Student student) {
          String query = "INSERT INTO student (name, age, grade) VALUES (?, ?, ?)";
```

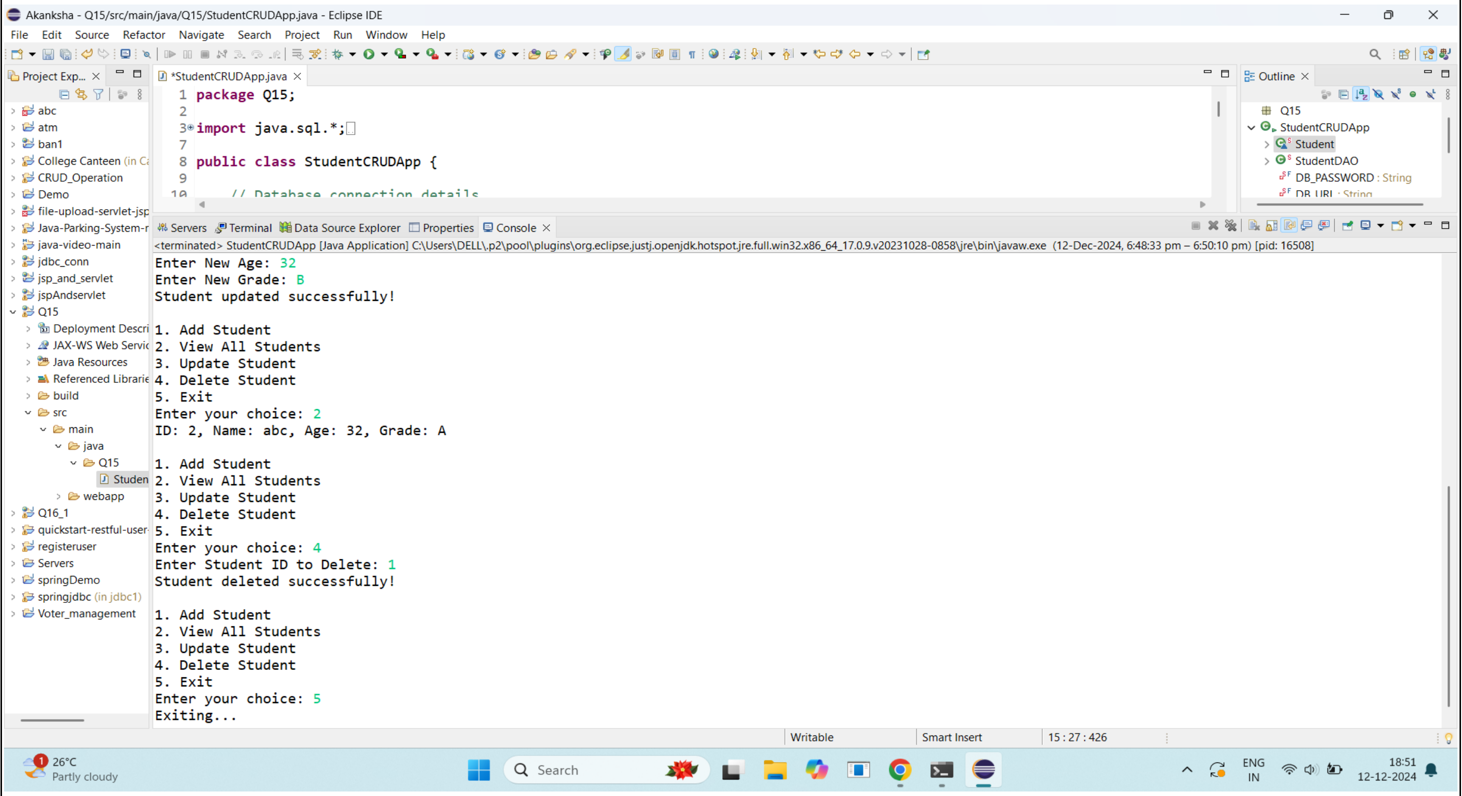
```
        try (PreparedStatement ps = connection.prepareStatement(query)) {
            ps.setString(1, student.getName());
            ps.setInt(2, student.getAge());
            ps.setString(3, student.getGrade());
            ps.executeUpdate();
            System.out.println("Student added successfully!");
        } catch (SQLException e) {
            e.printStackTrace();}}
    // Read all students
    public List<Student> getAllStudents() {
        List<Student> students = new ArrayList<>();
        String query = "SELECT * FROM student";
        try (Statement stmt = connection.createStatement();
             ResultSet rs = stmt.executeQuery(query)) {
     while (rs.next()) {
                int id = rs.getInt("student_id");
                String name = rs.getString("name");
                int age = rs.getInt("age");
                String grade = rs.getString("grade");
                students.add(new Student(id, name, age, grade));
            }
        } catch (SQLException e) {
            e.printStackTrace();}
       return students;
    }
// Update student information
    public void updateStudent(Student student) {
        String query = "UPDATE student SET name = ?, age = ?, grade = ? WHERE student_id = ?";
        try (PreparedStatement ps = connection.prepareStatement(query)) {
            ps.setString(1, student.getName());
            ps.setInt(2, student.getAge());
            ps.setString(3, student.getGrade());
            ps.setInt(4, student.getStudentId());
            ps.executeUpdate();
            System.out.println("Student updated successfully!");
        } catch (SQLException e) {
            e.printStackTrace();}}
    // Delete student by ID
    public void deleteStudent(int studentId) {
        String query = "DELETE FROM student WHERE student_id = ?";
        try (PreparedStatement ps = connection.prepareStatement(query)) {
            ps.setInt(1, studentId);
            ps.executeUpdate();
            System.out.println("Student deleted successfully!");
        } catch (SQLException e) {
            e.printStackTrace();}}}
// Main Application for interacting with the user
  public static void main(String[] args) {
    StudentDAO studentDAO = new StudentDAO();
    Scanner scanner = new Scanner(System.in);
    int choice;
    do {
        System.out.println("\n1. Add Student");
        System.out.println("2. View All Students");
        System.out.println("3. Update Student");
        System.out.println("4. Delete Student");
        System.out.println("5. Exit");
        System.out.print("Enter your choice: ");
        choice = scanner.nextInt();
         switch (choice) {
            case 1:
// Add a new student
```

```
                System.out.print("Enter Name: ");
                String name = scanner.next();
                System.out.print("Enter Age: ");
                int age = scanner.nextInt();
                System.out.print("Enter Grade: ");
                String grade = scanner.next();
                Student student = new Student(0, name, age, grade);
                studentDAO.createStudent(student);
                break;

            case 2:
                // View all students
                List<Student> students = studentDAO.getAllStudents();
                for (Student s : students) {
                    System.out.println("ID: " + s.getStudentId() + ", Name: " + s.getName() + ", Age: " +
s.getAge() + ", Grade: " + s.getGrade());
                }
                break;

            case 3:
                // Update student information
                System.out.print("Enter Student ID to Update: ");
                int studentIdToUpdate = scanner.nextInt();
                System.out.print("Enter New Name: ");
                String newName = scanner.next();
                System.out.print("Enter New Age: ");
                int newAge = scanner.nextInt();
                System.out.print("Enter New Grade: ");
                String newGrade = scanner.next();
                Student updatedStudent = new Student(studentIdToUpdate, newName, newAge,
newGrade);
                studentDAO.updateStudent(updatedStudent);
            break;

        case 4:
            // Delete a student
            System.out.print("Enter Student ID to Delete: ");
            int studentIdToDelete = scanner.nextInt();
            studentDAO.deleteStudent(studentIdToDelete);
            break;

        case 5:
            // Exit
            System.out.println("Exiting...");
            break;

        default:
            System.out.println("Invalid choice! Please try again.");}
    } while (choice != 5);
    scanner.close();}}
```



```
package Q15;

import java.sql.*;

public class StudentCRUDApp {

    // Database connection details
```

```
<terminated> StudentCRUDApp [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858\jre\bin\javaw.exe  (12-Dec-2024, 6:48:33 pm – 6:50:10 pm) [pid: 16508]
1. Add Student
2. View All Students
3. Update Student
4. Delete Student
5. Exit
Enter your choice: 1
Enter Name: abc
Enter Age: 32
Enter Grade: A
Student added successfully!

1. Add Student
2. View All Students
3. Update Student
4. Delete Student
5. Exit
Enter your choice: 2
ID: 2, Name: abc, Age: 32, Grade: A

1. Add Student
2. View All Students
3. Update Student
4. Delete Student
5. Exit
Enter your choice: 3
Enter Student ID to Update: 1
Enter New Name: xyz
Enter New Age: 32
```

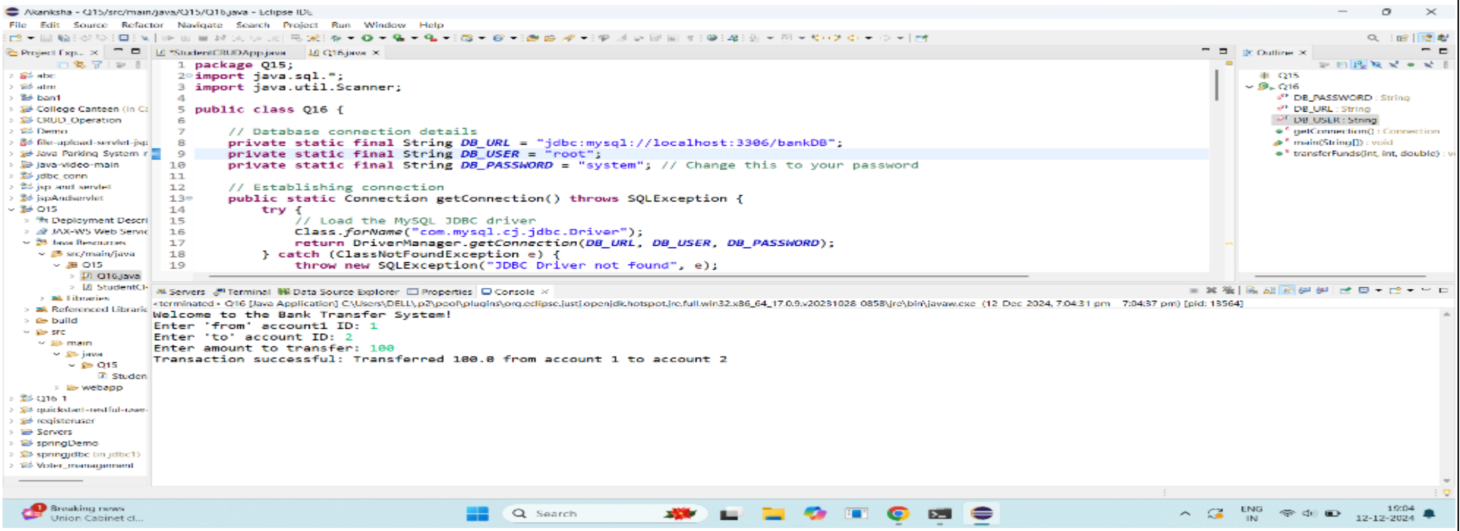


```
package Q15;

import java.sql.*;

public class StudentCRUDApp {

    // Database connection details
```

```
<terminated> StudentCRUDApp [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858\jre\bin\javaw.exe  (12-Dec-2024, 6:48:33 pm – 6:50:10 pm) [pid: 16508]
Enter New Age: 32
Enter New Grade: B
Student updated successfully!

1. Add Student
2. View All Students
3. Update Student
4. Delete Student
5. Exit
Enter your choice: 2
ID: 2, Name: abc, Age: 32, Grade: A

1. Add Student
2. View All Students
3. Update Student
4. Delete Student
5. Exit
Enter your choice: 4
Enter Student ID to Delete: 1
Student deleted successfully!

1. Add Student
2. View All Students
3. Update Student
4. Delete Student
5. Exit
Enter your choice: 5
Exiting...
```

16.Create a Java program to demonstrate transaction management and rollbacks using JDBC. Establish a connection to a database that supports transactions.Write Java code to perform multiple SQL operations within a transaction, such as transferring funds between bank accounts.Implement commit and rollback operations based on specific conditions (e.g., if a transaction fails).Use SQL exceptions to handle errors and ensure data integrity. Execute the program and observe the effect of commit and rollback operations on the database.

```
import java.sql.*;
import java.util.Scanner;
public class DynamicBankTransactionDemo {
 // Database connection details
 private static final String DB_URL = "jdbc:mysql://localhost:3306/bankDB";
  private static final String DB_USER = "root";
    private static final String DB_PASSWORD = "system"; // Change this to your password
 // Establishing connection
    public static Connection getConnection() throws SQLException {
        try {
            // Load the MySQL JDBC driver
            Class.forName("com.mysql.cj.jdbc.Driver");
            return DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
        } catch (ClassNotFoundException e) {
            throw new SQLException("JDBC Driver not found", e);}}
  // Method to perform the bank transfer (transaction management)
    public static void transferFunds(int fromAccountId, int toAccountId, double amount) {
        Connection connection = null;
        PreparedStatement withdrawStmt = null;
        PreparedStatement depositStmt = null;
        try {
            // Step 1: Establish a connection
            connection = getConnection();
            // Step 2: Set auto-commit to false
            connection.setAutoCommit(false);
           // Step 3: Prepare the SQL queries for withdrawal and deposit
            String withdrawQuery = "UPDATE accounts SET balance = balance - ? WHERE account_id = ?";
            String depositQuery = "UPDATE accounts SET balance = balance + ? WHERE account_id = ?";
            withdrawStmt = connection.prepareStatement(withdrawQuery);
            depositStmt = connection.prepareStatement(depositQuery);
            // Step 4: Withdraw money from the 'from' account
            withdrawStmt.setDouble(1, amount);
            withdrawStmt.setInt(2, fromAccountId);
            int withdrawRowsAffected = withdrawStmt.executeUpdate();
            if (withdrawRowsAffected == 0) {
                throw new SQLException("Insufficient funds or invalid 'from' account");}
             // Step 5: Deposit money into the 'to' account
            depositStmt.setDouble(1, amount);
            depositStmt.setInt(2, toAccountId);
            int depositRowsAffected = depositStmt.executeUpdate();
            if (depositRowsAffected == 0) {
                throw new SQLException("Invalid 'to' account");}
             // Step 6: Commit the transaction if no issues
            connection.commit();
            System.out.println("Transaction successful: Transferred " + amount + " from account " +
fromAccountId + " to account " + toAccountId);}
catch (SQLException e) {
            // Step 7: Rollback the transaction in case of any exception
            try {
                if (connection != null) {
                    connection.rollback();}}
               catch (SQLException ex) {
                System.out.println("Failed to rollback transaction: " + ex.getMessage());}
            System.out.println("Transaction failed: " + e.getMessage());
        } finally {
            // Step 8: Close resources
            try {
                if (withdrawStmt != null) withdrawStmt.close();
                if (depositStmt != null) depositStmt.close();
                if (connection != null) connection.close();
            } catch (SQLException e) {
                e.printStackTrace();}}}
    public static void main(String[] args) {
        // Create a Scanner object to take user input
        Scanner scanner = new Scanner(System.in);
        // Ask for user input
        System.out.println("Welcome to the Bank Transfer System!");
      // Get from account ID, to account ID, and the amount to transfer
        System.out.print("Enter 'from' account1 ID: ");
        int fromAccountId = scanner.nextInt();
       System.out.print("Enter 'to' account ID: ");
        int toAccountId = scanner.nextInt();
        System.out.print("Enter amount to transfer: ");
        double amount = scanner.nextDouble();
       // Validate that the transfer amount is positive
        if (amount <= 0) {
            System.out.println("Transfer amount must be greater than zero.");
            return;}
         // Perform the transaction
        transferFunds(fromAccountId, toAccountId, amount);
        // Close the scanner
        scanner.close();}}
```



```
package Q15;
import java.sql.*;
import java.util.Scanner;

public class Q16 {

    // Database connection details
    private static final String DB_URL = "jdbc:mysql://localhost:3306/bankDB";
    private static final String DB_USER = "root";
    private static final String DB_PASSWORD = "system"; // Change this to your password

    // Establishing connection
    public static Connection getConnection() throws SQLException {
        try {
            // Load the MySQL JDBC driver
            Class.forName("com.mysql.cj.jdbc.Driver");
            return DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
        } catch (ClassNotFoundException e) {
            throw new SQLException("JDBC Driver not found", e);
```

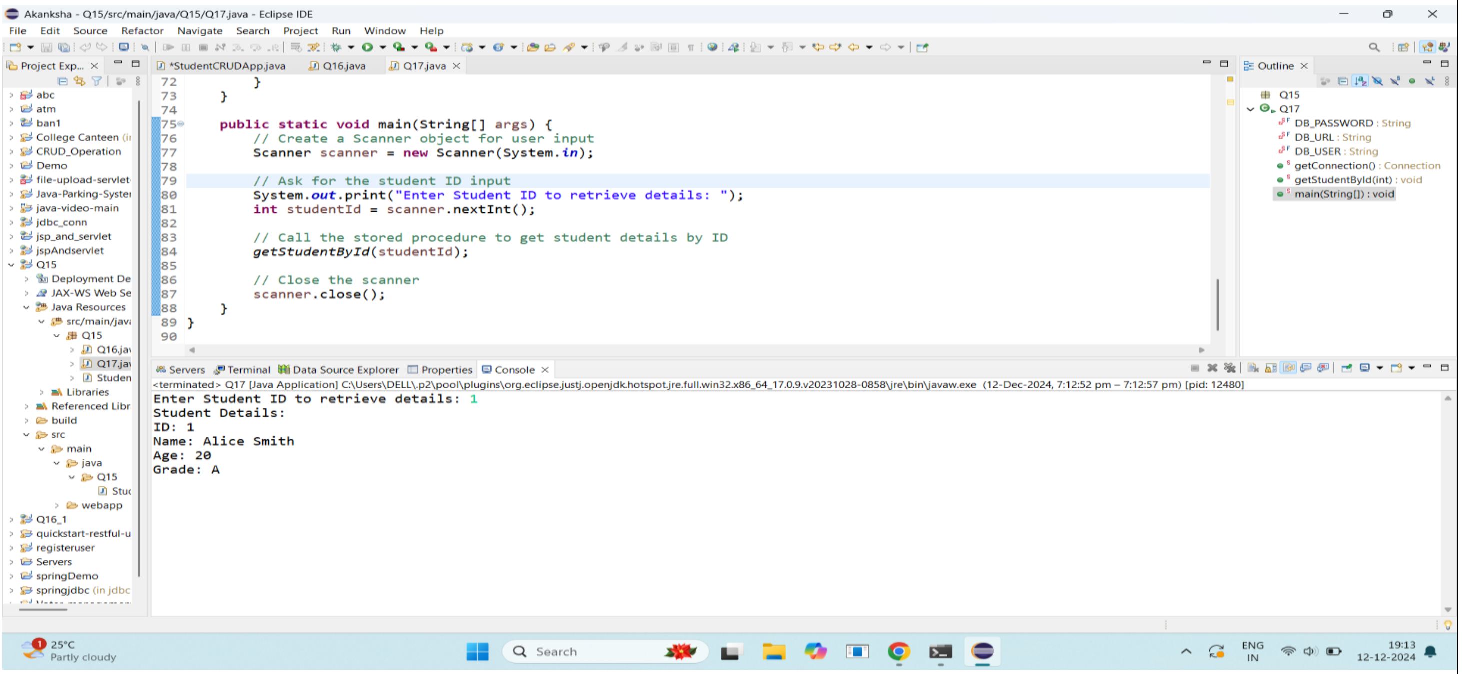
```
<terminated> Q16 [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028 0858\jre\bin\javaw.exe (12 Dec 2024, 7:04:31 pm   7:04:37 pm) [pid: 13564]
Welcome to the Bank Transfer System!
Enter 'from' account1 ID: 1
Enter 'to' account ID: 2
Enter amount to transfer: 100
Transaction successful: Transferred 100.0 from account 1 to account 2
```

17.Create a database schema named "University" with tables for storing student records. d. Create a stored procedure named "getStudentById" that accepts a student ID as input and returns the corresponding student details. e. Populate the student table with sample data. f. Establish a JDBC connection to the "University" database. g. Write a Java method to invoke the "getStudentById" stored procedure using CallableStatement. h. Prompt the user to input a student ID. i. Pass the input student ID to the CallableStatement as a parameter. j. Execute the CallableStatement to retrieve the student details. k. Display the retrieved student details (e.g., ID, name, age, etc.) to the user.

```
import java.sql.*;
import java.util.Scanner;
import java.sql.*;
import java.util.Scanner;
public class UniversityDatabaseApp {
// Database connection details
  private static final String DB_URL = "jdbc:mysql://localhost:3306/University";
  private static final String DB_USER = "root";
  private static final String DB_PASSWORD = "system"; // Change this to your password
 // Method to establish the database connection
  public static Connection getConnection() throws SQLException {
    try {
      // Load the MySQL JDBC driver
      Class.forName("com.mysql.cj.jdbc.Driver");
      return DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);}
     catch (ClassNotFoundException e) {
      throw new SQLException("JDBC Driver not found", e);}}
  // Method to invoke the "getStudentById" stored procedure
  public static void getStudentById(int studentId) {
    Connection connection = null;
    CallableStatement callableStatement = null;
    ResultSet resultSet = null;
 try {
      // Establish connection
      connection = getConnection();
// Prepare the stored procedure call
      String sql = "{CALL getStudentById(?)}";
      callableStatement = connection.prepareCall(sql);
```

```
          callableStatement.setInt(1, studentId); // Set the input parameter
         // Execute the stored procedure
          resultSet = callableStatement.executeQuery();
        // Process the result set
          if (resultSet.next()) {
              int id = resultSet.getInt("student_id");
              String name = resultSet.getString("student_name");
              int age = resultSet.getInt("age");
              String grade = resultSet.getString("grade");
        // Display the student details
              System.out.println("Student Details:");
              System.out.println("ID: " + id);
              System.out.println("Name: " + name);
              System.out.println("Age: " + age);
              System.out.println("Grade: " + grade);
          } else {
              System.out.println("Student with ID " + studentId + " not found.");}
      } catch (SQLException e) {
          e.printStackTrace();
      } finally {
          // Close resources
          try {
              if (resultSet != null) resultSet.close();
              if (callableStatement != null) callableStatement.close();
              if (connection != null) connection.close();
          } catch (SQLException e) {
              e.printStackTrace();}}}
  public static void main(String[] args) {
      // Create a Scanner object for user input
      Scanner scanner = new Scanner(System.in);
```

```
// Ask for the student ID input

System.out.print("Enter Student ID to retrieve details: ");

int studentId = scanner.nextInt();

// Call the stored procedure to get student details by ID

getStudentById(studentId);

// Close the scanner

scanner.close();}}
```



Akanksha - Q15/src/main/java/Q15/Q17.java - Eclipse IDE

```
        }
    }

    public static void main(String[] args) {
        // Create a Scanner object for user input
        Scanner scanner = new Scanner(System.in);

        // Ask for the student ID input
        System.out.print("Enter Student ID to retrieve details: ");
        int studentId = scanner.nextInt();

        // Call the stored procedure to get student details by ID
        getStudentById(studentId);

        // Close the scanner
        scanner.close();
    }
}
```

```
<terminated> Q17 [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858\jre\bin\javaw.exe (12-Dec-2024, 7:12:52 pm – 7:12:57 pm) [pid: 12480]
Enter Student ID to retrieve details: 1
Student Details:
ID: 1
Name: Alice Smith
Age: 20
Grade: A
```

18. Develop a servlet that handles form submission from a web page. The servlet should extract form parameters (such as name, email, etc.), process them, and display the submitted data back to the user.Create a servlet class that extends HttpServlet.Implement the necessary methods (e.g., doGet or doPost) to handle HTTP requests. Read form parameters using the request object.Process the form data (e.g., validate inputs, perform calculations).Generate an HTML response to display the submitted data back to the user.

Formservelt.java

```
package com.servlet;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

/**

 * Servlet implementation class FormServlet

 */

@WebServlet("/FormServlet")

public class FormServlet extends HttpServlet {

        private static final long serialVersionUID = 1L;

     /**

   * @see HttpServlet#HttpServlet()

   */

  public FormServlet() {

    super();

    // TODO Auto-generated constructor stub}

     /**

         * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

         */

        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

                // TODO Auto-generated method stub

                response.getWriter().append("Served at: ").append(request.getContextPath());}

        /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
         */
   @Override
   protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
  // Set response content type
       response.setContentType("text/html");
 // Get the PrintWriter to write the response
       PrintWriter out = response.getWriter();
 // Extract form parameters from the request
       String name = request.getParameter("name");
       String email = request.getParameter("email");
       String age = request.getParameter("age");
// Process the form data (basic validation for age)
       boolean isValid = true;
       StringBuilder errorMessages = new StringBuilder();
  try {
           int ageValue = Integer.parseInt(age);
           if (ageValue <= 0) {
               isValid = false;
               errorMessages.append("Age must be a positive number.<br>");}
            } catch (NumberFormatException e) {
           isValid = false;
           errorMessages.append("Age must be a valid number.<br>");}
         // Generate the response
       out.println("<html><body>");
       if (isValid) {
           // Display the submitted data
           out.println("<h2>Form Submitted Successfully!</h2>");
           out.println("<p><b>Name:</b> " + name + "</p>");
           out.println("<p><b>Email:</b> " + email + "</p>");
```

```
        out.println("<p><b>Age:</b> " + age + "</p>");
    } else {
        out.println("<h2>Form Submission Failed</h2>");
        out.println("<p><b>Errors:</b></p>");
        out.println("<p>" + errorMessages.toString() + "</p>");}
       out.println("<a href='form.html'>Go Back</a>");
        out.println("</body></html>");}}
```

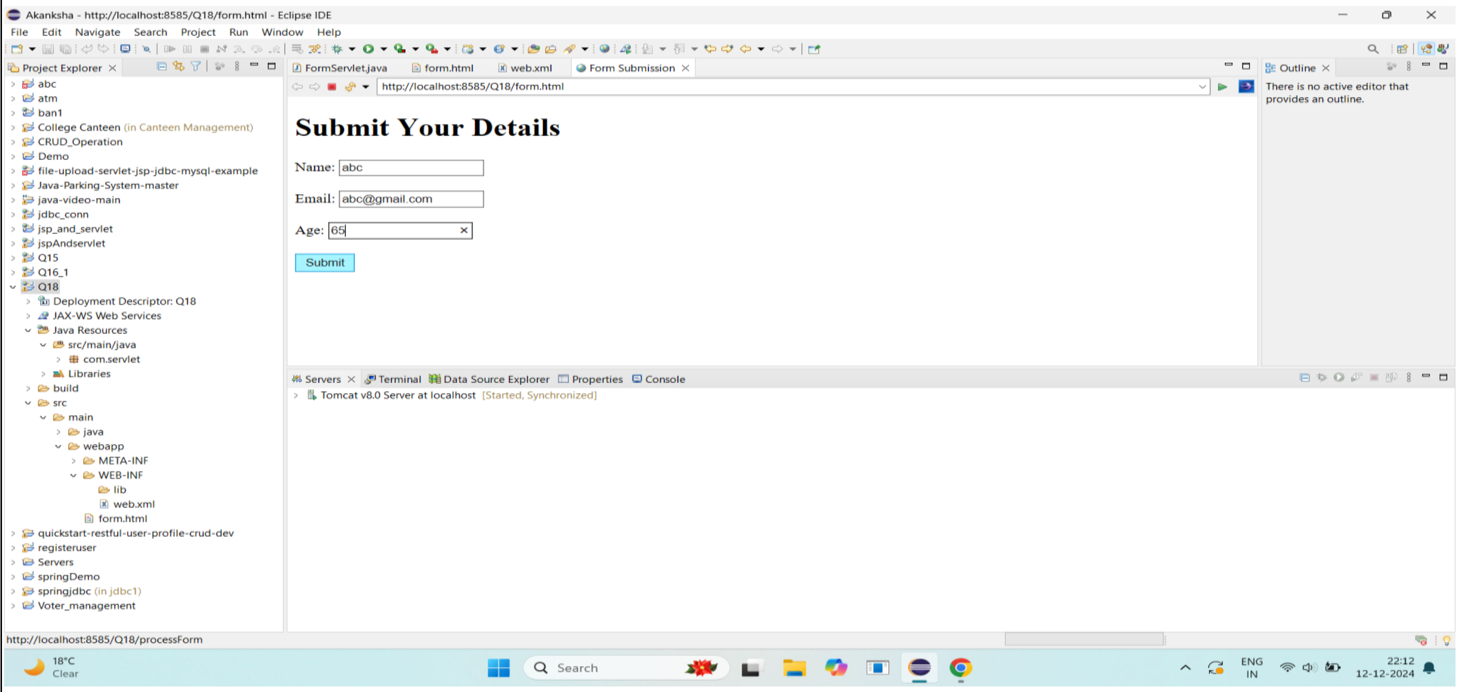
form.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Form Submission</title>
</head>
<body>
   <h1>Submit Your Details</h1>
   <form action="processForm" method="post">
       <label for="name">Name:</label>
       <input type="text" id="name" name="name" required><br><br>
       <label for="email">Email:</label>
       <input type="email" id="email" name="email" required><br><br>
        <label for="age">Age:</label>
       <input type="text" id="age" name="age" required><br><br>
      <input type="submit" value="Submit">
   </form>
</body>
</html>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?><web-app
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-
app_3_1.xsd" id="WebApp_ID" version="3.1">
 <servlet>
        <servlet-name>FormServlet</servlet-name>
        <servlet-class>com.servlet.FormServlet</servlet-class>
    </servlet>
    <!-- Servlet Mapping -->
    <servlet-mapping>
        <servlet-name>FormServlet</servlet-name>
        <url-pattern>/processForm</url-pattern>
    </servlet-mapping
    <display-name>Q18</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.jsp</welcome-file>
    <welcome-file>default.htm</welcome-file>
  </welcome-file-list>
</web-app>
```


Akanksha - http://localhost:8585/Q18/form.html - Eclipse IDE

File Edit Navigate Search Project Run Window Help

Project Explorer

- abc
- atm
- ban1
- College Canteen (in Canteen Management)
- CRUD_Operation
- Demo
- file-upload-servlet-jsp-jdbc-mysql-example
- Java-Parking-System-master
- java-video-main
- jdbc_conn
- jsp_and_servlet
- jspAndservlet
- Q15
- Q16_1
- Q18
  - Deployment Descriptor: Q18
  - JAX-WS Web Services
  - Java Resources
    - src/main/java
      - com.servlet
    - Libraries
  - build
  - src
    - main
      - java
      - webapp
        - META-INF
        - WEB-INF
          - lib
          - web.xml
        - form.html
- quickstart-restful-user-profile-crud-dev
- registeruser
- Servers
- springDemo
- springjdbc (in jdbc1)
- Voter_management

FormServlet.java | form.html | web.xml | Form Submission

http://localhost:8585/Q18/form.html

Submit Your Details

Name: abc

Email: abc@gmail.com

Age: 65

Submit

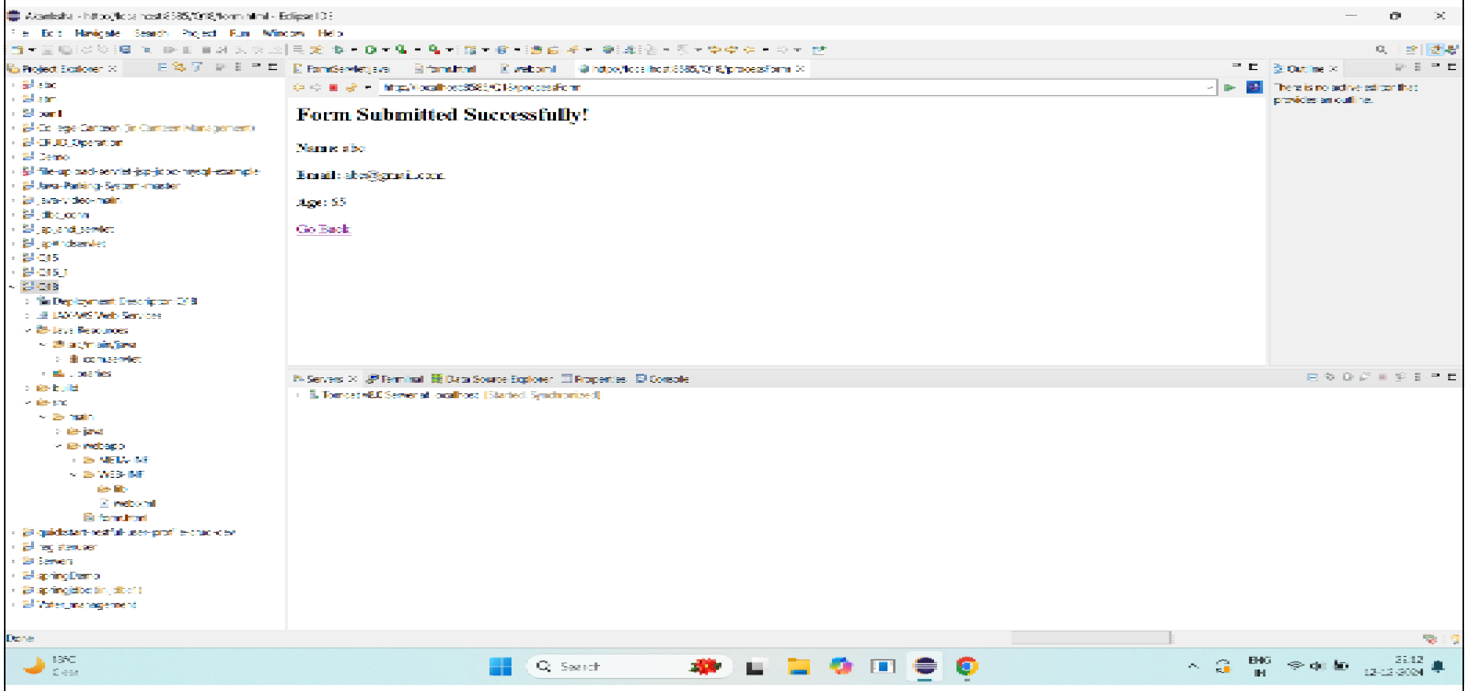
Outline: There is no active editor that provides an outline.

Servers | Terminal | Data Source Explorer | Properties | Console

Tomcat v8.0 Server at localhost [Started, Synchronized]

http://localhost:8585/Q18/processForm

18°C Clear | Search | ENG IN | 22:12 12-12-2024


Akanksha - http://localhost:8585/Q18/form.html - Eclipse IDE

FormServlet.java | form.html | web.xml | http://localhost:8585/Q18/processForm

http://localhost:8585/Q18/processForm

Form Submitted Successfully!

Name: abc

Email: abc@gmail.com

Age: 65

Go Back

Outline: There is no active editor that provides an outline.

Servers | Terminal | Data Source Explorer | Properties | Console

Tomcat v8.0 Server at localhost [Started, Synchronized]

Done

18°C Clear | Search | ENG IN | 22:12 12-12-2024

19. Develop a web application that includes user authentication using servlets and JavaServer Pages (JSP). Users should be able to log in with a username and password, and upon successful authentication, they should be redirected to a welcome pageCreate a servlet to handle user authentication.Implement a login form using JSP. Use session management to keep track of authenticated users. Validate user credentials against a predefined set (e.g., in-memory storage or database).Upon successful authentication, redirect the user to a welcome page using JSP.

Loginservlet.java

```
package com.servlet;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

/**

 * Servlet implementation class LoginServlet

 */

@WebServlet("/LoginServlet")

public class LoginServlet extends HttpServlet {

   // Predefined username and password for demo purposes

   private static final String VALID_USERNAME = "admin";

  private static final String VALID_PASSWORD = "password123";

 @Override

  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

 String username = request.getParameter("username");

  String password = request.getParameter("password");

// Validate credentials

if (VALID_USERNAME.equals(username) && VALID_PASSWORD.equals(password)) {

  // Credentials are correct, create a session for the user

   HttpSession session = request.getSession();

 session.setAttribute("username", username);

 // Redirect to the welcome page

  response.sendRedirect("welcome.jsp");
 } else {
 // Invalid credentials, show error message
request.setAttribute("errorMessage", "Invalid username or password.");
request.getRequestDispatcher("login.jsp").forward(request, response);}}
```

logoutservlet.java

```
package com.servlet;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
/**
 * Servlet implementation class LogoutServlet
 */
@WebServlet(name = "LogoutServlet1", urlPatterns = { "/LogoutServlet1" })
public class LogoutServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;
     /**
    * @see HttpServlet#HttpServlet()
    */
   public LogoutServlet() {
     super();
     // TODO Auto-generated constructor stub}
   /**
         * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
response)
         */
   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
```

```
                    // Invalidate the session
                    HttpSession session = request.getSession(false);
                    if (session != null) {
                       session.invalidate();}
                    // Redirect to login page
                    response.sendRedirect("login.jsp");}
                 /**
                          * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
response)
                          */
                         protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                                    // TODO Auto-generated method stub
                                    doGet(request, response);}}
```

login.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
   <h2>Login</h2>
   <form action="login" method="post">
  <label for="username">Username:</label>
  <input type="text" id="username" name="username" required><br><br>
 <label for="password">Password:</label>
 <input type="password" id="password" name="password" required><br><br>
```

```
<input type="submit" value="Login">
  </form>
</body>
</body>
</html>
```

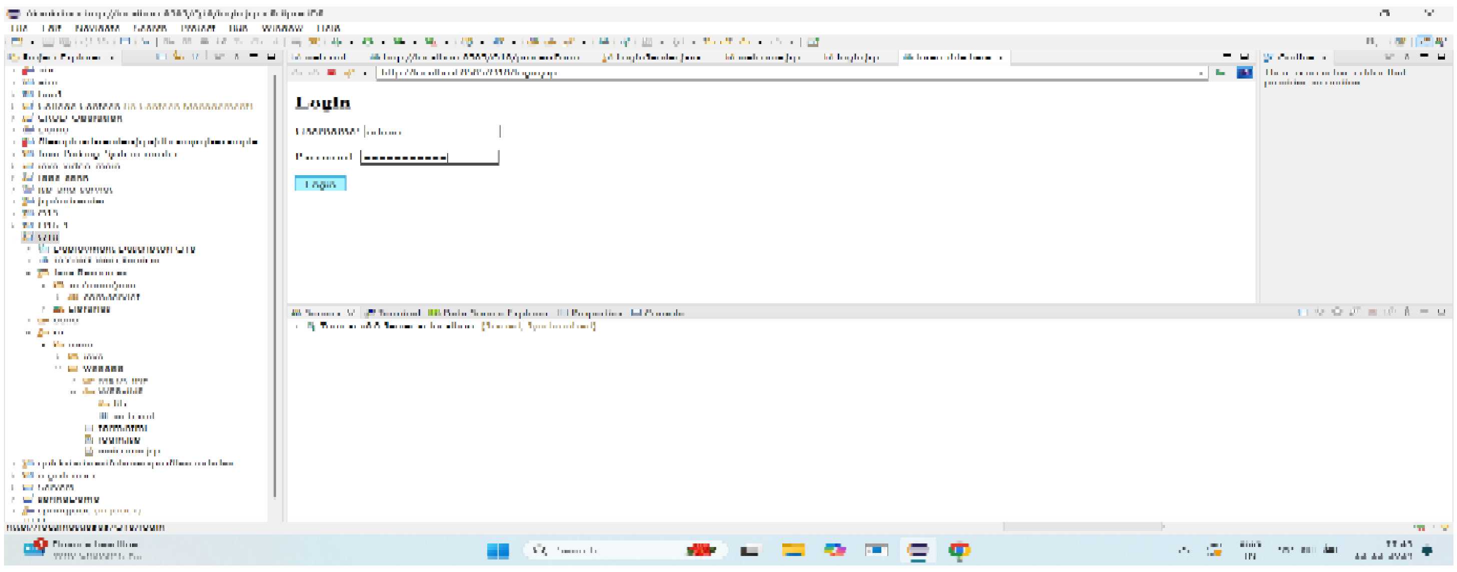
welcome.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
   <title>Welcome</title>
</head>
<body>
   <h2>Welcome, <%= session.getAttribute("username") %>!</h2>
   <p>You have successfully logged in.</p>
   <a href="logout">Logout</a>
</body>
</html>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-
app_3_1.xsd" id="WebApp_ID" version="3.1">
   <servlet>
      <servlet-name>LoginServlet</servlet-name>
      <servlet-class>com.servlet.LoginServlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>LoginServlet</servlet-name>
```

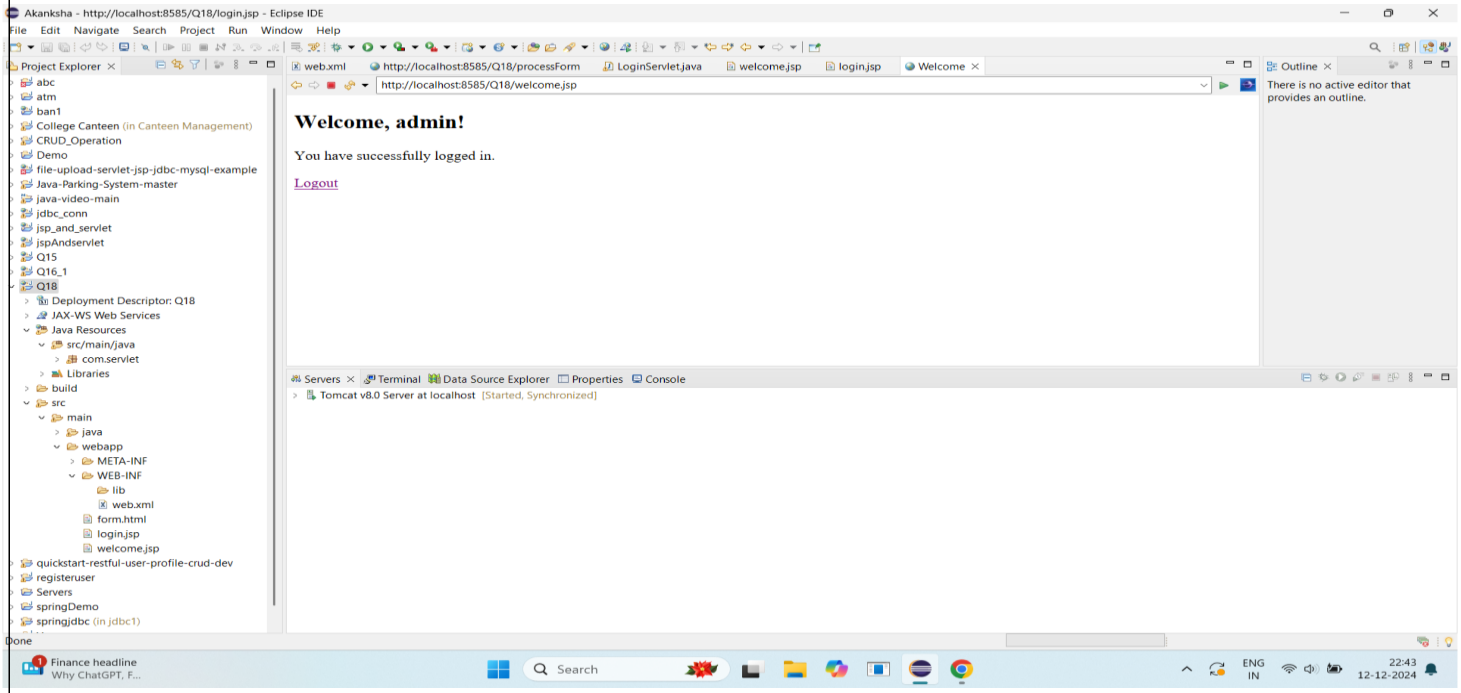
```
        <url-pattern>/login</url-pattern>
    </servlet-mapping>
<!-- Logout Servlet -->
    <servlet>
        <servlet-name>LogoutServlet</servlet-name>
        <servlet-class>com.servlet.LogoutServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>LogoutServlet</servlet-name>
        <url-pattern>/logout</url-pattern>
    </servlet-mapping>
  <display-name>Q18</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.jsp</welcome-file>
    <welcome-file>default.htm</welcome-file>
  </welcome-file-list>
    </web-app>
```



Login

Username: [illegible]

Password: ●●●●●●●●●●●●


Akanksha - http://localhost:8585/Q18/login.jsp - Eclipse IDE

File Edit Navigate Search Project Run Window Help

Project Explorer

- abc
- atm
- ban1
- College Canteen (in Canteen Management)
- CRUD_Operation
- Demo
- file-upload-servlet-jsp-jdbc-mysql-example
- Java-Parking-System-master
- java-video-main
- jdbc_conn
- jsp_and_servlet
- jspAndservlet
- Q15
- Q16_1
- Q18
  - Deployment Descriptor: Q18
  - JAX-WS Web Services
  - Java Resources
    - src/main/java
      - com.servlet
    - Libraries
  - build
  - src
    - main
      - java
      - webapp
        - META-INF
        - WEB-INF
          - lib
          - web.xml
        - form.html
        - login.jsp
        - welcome.jsp
- quickstart-restful-user-profile-crud-dev
- registeruser
- Servers
- springDemo
- springjdbc (in jdbc1)

web.xml | http://localhost:8585/Q18/processForm | LoginServlet.java | welcome.jsp | login.jsp | Welcome

http://localhost:8585/Q18/welcome.jsp

Welcome, admin!

You have successfully logged in.

Logout

Outline

There is no active editor that provides an outline.

Servers | Terminal | Data Source Explorer | Properties | Console

Tomcat v8.0 Server at localhost [Started, Synchronized]

Done

Finance headline
Why ChatGPT, F...

Search

ENG IN 22:43 12-12-2024

20. Create a dynamic web application for performing CRUD (Create, Read, Update, Delete) operations using servlets and JSP. The application should allow users to interact with a database to manipulate data records.Design a database schema for storing data records (e.g., user information, product details).Implement servlets to handle CRUD operations (e.g., adding new records, retrieving records, updating records, deleting records). Develop JSP pages to interact with users (e.g., display data, input forms for adding/updating records).Use JDBC (Java Database Connectivity) to connect to the database and perform database operations.Implement error handling and validation for user inputs.

productmanager.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
   <%@ page import="com.servlet.Product" %>
   <!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
   <h2>Product Management</h2>
 <!-- Form for adding/editing products -->
   <h3><%= request.getAttribute("product") != null ? "Edit Product" : "Add Product" %></h3>
   <form action="ProductServlet" method="POST">
      <input type="hidden" name="action" value="<%= request.getAttribute("product") != null ?
"update" : "create" %>">
      <% if (request.getAttribute("product") != null) { %>
         <input type="hidden" name="id" value="<%= ((Product)
request.getAttribute("product")).getId() %>">
      <% } %>
      <label for="name">Product Name:</label><br>
      <input type="text" name="name" value="<%= request.getAttribute("product") != null ?
((Product) request.getAttribute("product")).getName() : "" %>" required><br><br>
      <label for="description">Description:</label><br>
      <textarea name="description" required><%= request.getAttribute("product") != null ?
((Product) request.getAttribute("product")).getDescription() : "" %></textarea><br><br>
      <label for="price">Price:</label><br>
      <input type="number" step="0.01" name="price" value="<%= request.getAttribute("product")
!= null ? ((Product) request.getAttribute("product")).getPrice() : "" %>" required><br><br>
```

```
 <button type="submit"><%= request.getAttribute("product") != null ? "Update" : "Add" %>
Product</button>
   </form>
   <br><br>
 <!-- Display Product List -->
   <h3>Product List</h3>
   <table border="1">
       <tr>
           <th>Name</th>
           <th>Description</th>
           <th>Price</th>
           <th>Actions</th>
       </tr>
       <c:forEach var="product" items="${products}">
           <tr>
               <td>${product.name}</td>
               <td>${product.description}</td>
               <td>${product.price}</td>
               <td>
                   <a href="ProductServlet?action=edit&id=${product.id}">Edit</a> |
                   <a href="ProductServlet?action=delete&id=${product.id}">Delete</a>
               </td>
           </tr>
       </c:forEach>
   </table>
</body>
</html>
```

Productservlet.java

```
package com.servlet;
import java.io.IOException;
import java.sql.SQLException;
import java.util.List;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
/**
 * Servlet implementation class ProductServlet
 */
@WebServlet("/ProductServlet")
public class ProductServlet extends HttpServlet {
            private static final long serialVersionUID = 1L;
        /**
     * @see HttpServlet#HttpServlet()
     */
    public ProductServlet() {
        super();
        // TODO Auto-generated constructor stub}
    /**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
             */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        String action = request.getParameter("action");
        ProductDAO productDAO = new ProductDAO();
  if ("delete".equals(action)) {
            int id = Integer.parseInt(request.getParameter("id"));
            try {
                productDAO.deleteProduct(id);
                response.sendRedirect("productManager.jsp");
            } catch (SQLException e) {
                e.printStackTrace();
                response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);}}
            else if ("edit".equals(action)) {
            int id = Integer.parseInt(request.getParameter("id"));
            try {
                Product product = productDAO.getProductById(id);
                request.setAttribute("product", product);
                RequestDispatcher dispatcher = request.getRequestDispatcher("productManager.jsp");
                dispatcher.forward(request, response);
            } catch (SQLException e) {
                e.printStackTrace();
                response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);}
          } else {
            try {
                List<Product> products = productDAO.getAllProducts();
                request.setAttribute("products", products);
                RequestDispatcher dispatcher = request.getRequestDispatcher("productManager.jsp");
                dispatcher.forward(request, response);
            } catch (SQLException e) {
                e.printStackTrace();
                response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);}}}
            protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        String name = request.getParameter("name");
        String description = request.getParameter("description");
        double price = Double.parseDouble(request.getParameter("price"));
        String action = request.getParameter("action");
        Product product = new Product();
        product.setName(name);
        product.setDescription(description);
```

```
        product.setPrice(price);
        ProductDAO productDAO = new ProductDAO();
         try {
            if ("create".equals(action)) {
                productDAO.addProduct(product);
            } else if ("update".equals(action)) {
                int id = Integer.parseInt(request.getParameter("id"));
                product.setId(id);
                productDAO.updateProduct(product);}
            response.sendRedirect("productManager.jsp");}
         catch (SQLException e) {
            e.printStackTrace();
            response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);}}}
```

productDAO.java

```
package com.servlet;
import java.sql.*;
import java.util.*;
public class ProductDAO {
  public List<Product> getAllProducts() throws SQLException {
        List<Product> productList = new ArrayList<>();
        Connection connection = DBUtil.getConnection();
        String sql = "SELECT * FROM products";
        Statement statement = connection.createStatement();
        ResultSet rs = statement.executeQuery(sql);
          while (rs.next()) {
            Product product = new Product();
            product.setId(rs.getInt("id"));
            product.setName(rs.getString("name"));
            product.setDescription(rs.getString("description"));
            product.setPrice(rs.getDouble("price"));
            productList.add(product);}
```

```
   rs.close();
  statement.close();
  connection.close();
  return productList;}
public void addProduct(Product product) throws SQLException {
  Connection connection = DBUtil.getConnection();
  String sql = "INSERT INTO products (name, description, price) VALUES (?, ?, ?)";
  PreparedStatement stmt = connection.prepareStatement(sql);
  stmt.setString(1, product.getName());
  stmt.setString(2, product.getDescription());
  stmt.setDouble(3, product.getPrice());
  stmt.executeUpdate();
  stmt.close();
  connection.close();}
 public void updateProduct(Product product) throws SQLException {
  Connection connection = DBUtil.getConnection();
  String sql = "UPDATE products SET name = ?, description = ?, price = ? WHERE id = ?";
  PreparedStatement stmt = connection.prepareStatement(sql);
  stmt.setString(1, product.getName());
  stmt.setString(2, product.getDescription());
  stmt.setDouble(3, product.getPrice());
  stmt.setInt(4, product.getId());
  stmt.executeUpdate();
  stmt.close();
  connection.close();}
public void deleteProduct(int id) throws SQLException {
  Connection connection = DBUtil.getConnection();
  String sql = "DELETE FROM products WHERE id = ?";
  PreparedStatement stmt = connection.prepareStatement(sql);
  stmt.setInt(1, id);
  stmt.executeUpdate();
```

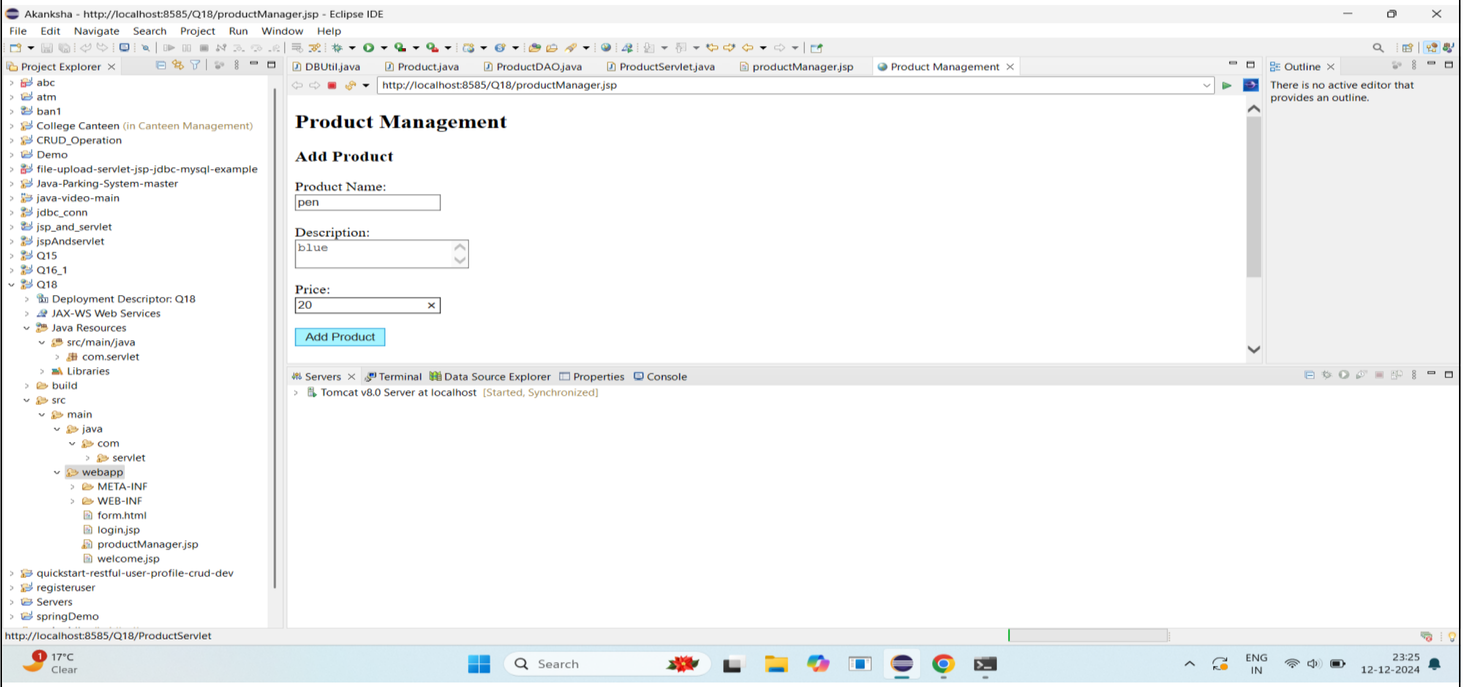
```
        stmt.close();
        connection.close();}
    public Product getProductById(int id) throws SQLException {
        Connection connection = DBUtil.getConnection();
        String sql = "SELECT * FROM products WHERE id = ?";
        PreparedStatement stmt = connection.prepareStatement(sql);
        stmt.setInt(1, id);
        ResultSet rs = stmt.executeQuery();
        Product product = null;
 if (rs.next()) {
            product = new Product();
            product.setId(rs.getInt("id"));
            product.setName(rs.getString("name"));
            product.setDescription(rs.getString("description"));
            product.setPrice(rs.getDouble("price"));}
           rs.close();
        stmt.close();
        connection.close();
        return product;}}
```

product.java

```
package com.servlet;
public class Product {
    private int id;
    private String name;
    private String description;
    private double price;
 // Getters and Setters
    public int getId() {
        return id;}
     public void setId(int id) {
        this.id = id;}
     public String getName() {
        return name;}
       public void setName(String name) {
        this.name = name;}
       public String getDescription() {
        return description;}
       public void setDescription(String description) {
        this.description = description;}
    public double getPrice() {
        return price;}
     public void setPrice(double price) {
        this.price = price;}}
```

DBUtil.java

```
package com.servlet;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class DBUtil {
   private static final String URL = "jdbc:mysql://localhost:3306/productdb";
   private static final String USER = "root";
   private static final String PASSWORD = "system";
public static Connection getConnection() throws SQLException {
       try {
          Class.forName("com.mysql.cj.jdbc.Driver");
          return DriverManager.getConnection(URL, USER, PASSWORD);
       } catch (Exception e) {
          e.printStackTrace();
          throw new SQLException("Database connection error", e);}}}
```


Akanksha - http://localhost:8585/Q18/productManager.jsp - Eclipse IDE
File Edit Navigate Search Project Run Window Help
Project Explorer
DBUtil.java
Product.java
ProductDAO.java
ProductServlet.java
productManager.jsp
Product Management
Outline
http://localhost:8585/Q18/productManager.jsp

There is no active editor that provides an outline.

abc
atm
ban1
College Canteen (in Canteen Management)
CRUD_Operation
Demo
file-upload-servlet-jsp-jdbc-mysql-example
Java-Parking-System-master
java-video-main
jdbc_conn
jsp_and_servlet
jspAndservlet
Q15
Q16_1
Q18
  Deployment Descriptor: Q18
  JAX-WS Web Services
  Java Resources
    src/main/java
      com.servlet
    Libraries
  build
  src
    main
      java
        com
          servlet
      webapp
        META-INF
        WEB-INF
        form.html
        login.jsp
        productManager.jsp
        welcome.jsp
quickstart-restful-user-profile-crud-dev
registeruser
Servers
springDemo

Product Management

Add Product

Product Name: pen

Description: blue

Price: 20

Add Product

Tomcat v8.0 Server at localhost [Started, Synchronized]

http://localhost:8585/Q18/ProductServlet

21. Develop a simple Java application to demonstrate the usage of Spring IOC container and Dependency Injection (DI) features.Configure a Spring IOC container using XML-based configuration.Define two POJO classes: Employee and Address, with appropriate attributes and methods.Implement Dependency Injection using Setter Injection to inject Address object into the Employee class.Write a Java program to retrieve an Employee object from the Spring IOC container and display its details along with the associated Address.Test the application to ensure proper DI and object creation.

MainApp.java

```
package springDemo;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

   public static void main(String[] args) {

      // Load the Spring configuration file

      ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

     // Retrieve the Employee bean from the IoC container

      emp emp = (emp) context.getBean("employee");

      // Display employee details

      emp.displayDetails();}}
```

Address.java

```
package springDemo;

public class Address {

   private String street;

   private String city;

   private String zipCode;

 // Getters and Setters

   public String getStreet() {

      return street;}

   public void setStreet(String street) {

      this.street = street; }

   public String getCity() {

      return city;   }

 public void setCity(String city) {

      this.city = city;}

    public String getZipCode() {
        return zipCode;}
     public void setZipCode(String zipCode) {
        this.zipCode = zipCode;}}
```

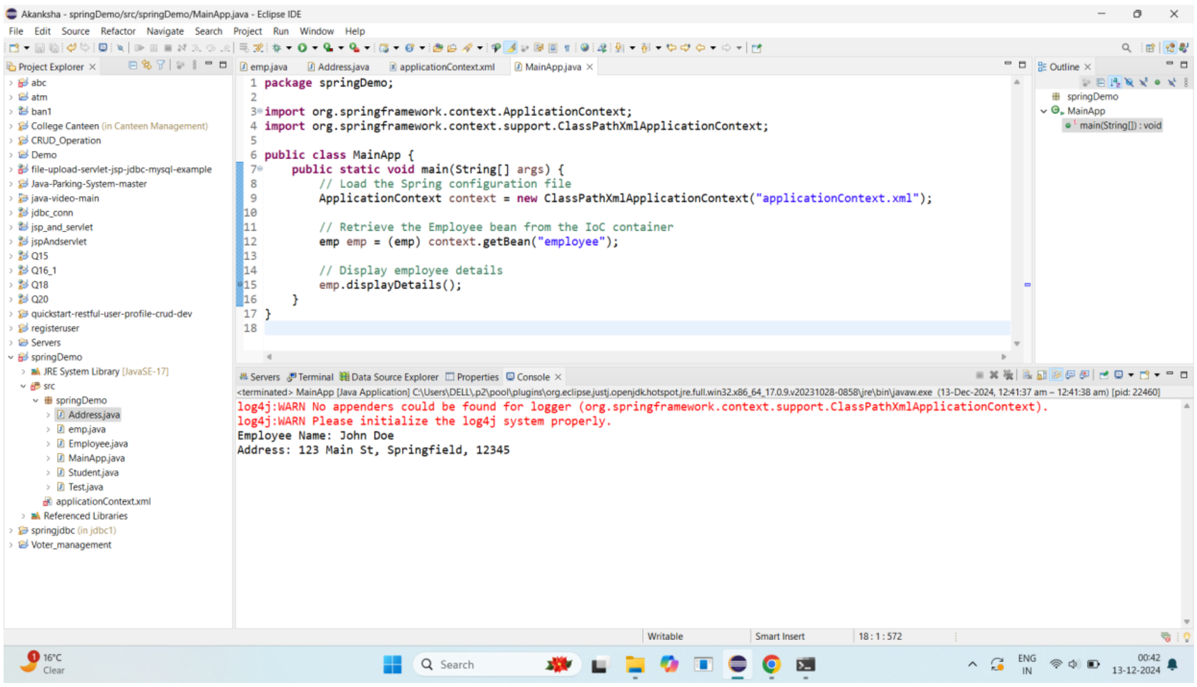
emp.java

```
package springDemo;
public class emp {
    private String name;
    private Address address;
// Setter Injection: Injecting the Address object using setter method
    public void setName(String name) {
        this.name = name;}
    public void setAddress(Address address) {
        this.address = address;}
    public void displayDetails() {
        System.out.println("Employee Name: " + name);
        System.out.println("Address: " + address.getStreet() + ", " + address.getCity() + ", " +
address.getZipCode());}}
```

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
    <beans xmlns="http://www.springframework.org/schema/beans"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns:context="http://www.springframework.org/schema/context"
         xmlns:aop="http://www.springframework.org/schema/aop"
         xsi:schemaLocation="
         http://www.springframework.org/schema/beans
           classpath:/org/springframework/beans/factory/xml/spring-beans-3.0.xsd
         http://www.springframework.org/schema/context
         classpath:/org/springframework/context/config/spring-context-3.0.xsd
        http://www.springframework.org/schema/aop
        classpath:/org/springframework/aop/config/spring-aop-3.0.xsd
         ">
```

```xml
    <bean id="address" class="springDemo.Address">
        <property name="street" value="123 Main St"/>
        <property name="city" value="Springfield"/>
        <property name="zipCode" value="12345"/>
    </bean>
<!-- Bean definition for Employee -->
    <bean id="employee" class="springDemo.emp">
        <property name="name" value="John Doe"/>
        <property name="address" ref="address"/>
    </bean>
 </beans>
```



```java
package springDemo;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
    public static void main(String[] args) {
        // Load the Spring configuration file
        ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

        // Retrieve the Employee bean from the IoC container
        emp emp = (emp) context.getBean("employee");

        // Display employee details
        emp.displayDetails();
    }
}
```

```
<terminated> MainApp [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858\jre\bin\javaw.exe  (13-Dec-2024, 12:41:37 am – 12:41:38 am) [pid: 22460]
log4j:WARN No appenders could be found for logger (org.springframework.context.support.ClassPathXmlApplicationContext).
log4j:WARN Please initialize the log4j system properly.
Employee Name: John Doe
Address: 123 Main St, Springfield, 12345
```

22. Implement a simple Java application using Spring Framework that demonstratesDependency Injection (DI) using constructor injection.Instructions:1. Create an interface MessageService with a method sendMessage().2. Create a class EmailService implementing MessageService that prints "Email message sent".3. Create a class SMSService implementing MessageService that prints "SMS message sent".4. Create a class MessageProcessor that depends on MessageService for sending messages.5. Configure Spring to inject EmailService into MessageProcessor using constructor injection.6. Test the application by creating an instance of MessageProcessor in main method and invoking sendMessage().

MainApp.java

```
import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

    public static void main(String[] args) {

        // Set the Spring profile dynamically

        System.setProperty("spring.profiles.active", "email");  // Change this to "sms" for SMS service

 // Load the Spring context from the configuration file

        ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
// Retrieve the MessageProcessor bean

    MessageProcessor messageProcessor = context.getBean("messageProcessor",
MessageProcessor.class);

  // Call the method to send the message

        messageProcessor.processMessage();  // Will print "Email message sent" or "SMS message
sent"}}
```

MessageProcessor.java

```
package com.example;

public class MessageProcessor {

    private MessageService messageService;

    // Constructor-based DI

    public MessageProcessor(MessageService messageService) {

        this.messageService = messageService;}

    public void processMessage() {

        messageService.sendMessage();}}
```

EmailService.java

```
package com.example;

public class EmailService implements MessageService {

    @Override
    public void sendMessage() {
        System.out.println("Email message sent");}}
```

MessageService.java

```
package com.example;
public interface MessageService {
    void sendMessage();}
```

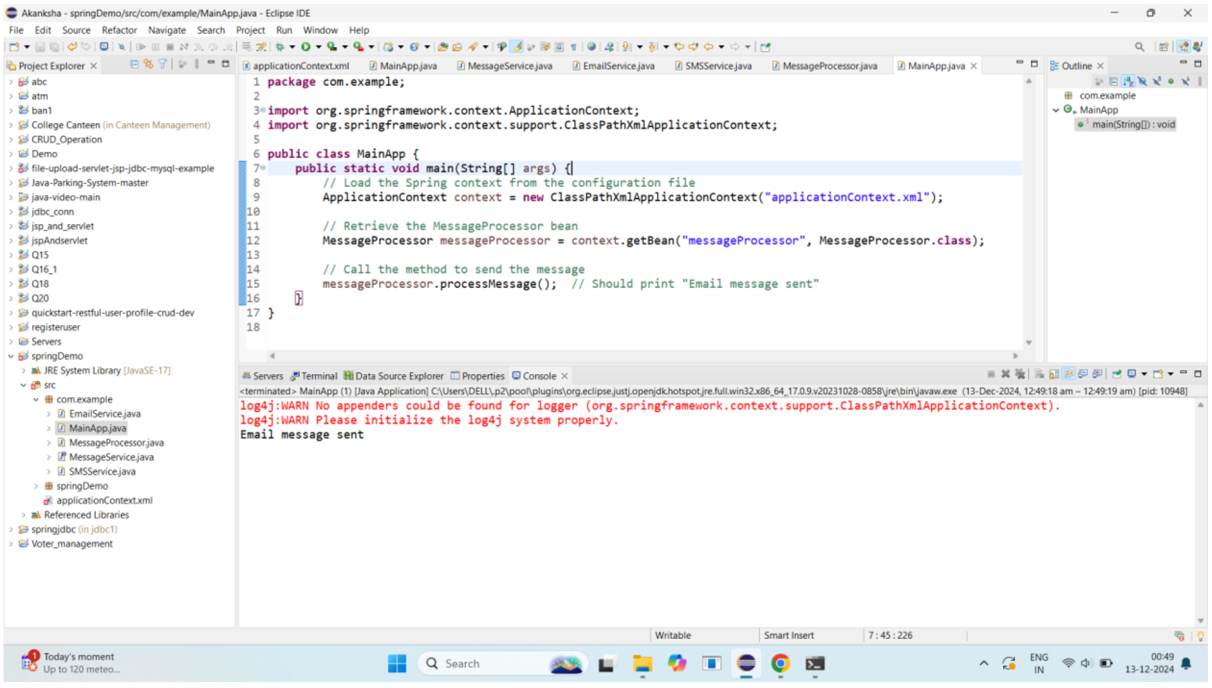
ApplicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
    <beans xmlns="http://www.springframework.org/schema/beans"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns:context="http://www.springframework.org/schema/context"
         xmlns:aop="http://www.springframework.org/schema/aop"
         xsi:schemaLocation="
         http://www.springframework.org/schema/beans
           classpath:/org/springframework/beans/factory/xml/spring-beans-3.0.xsd
         http://www.springframework.org/schema/context
         classpath:/org/springframework/context/config/spring-context-3.0.xsd
        http://www.springframework.org/schema/aop
        classpath:/org/springframework/aop/config/spring-aop-3.0.xsd
 ">
    <bean id="emailService" class="com.example.EmailService" />
  <!-- Bean definition for MessageProcessor with constructor injection -->
    <bean id="messageProcessor" class="com.example.MessageProcessor">
        <constructor-arg ref="emailService"/>
    </bean>
  </beans>
```

smsservice.java

```
<?xml version="1.0" encoding="UTF-8"?>
    <beans xmlns="http://www.springframework.org/schema/beans"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
      xmlns:context="http://www.springframework.org/schema/context"
      xmlns:aop="http://www.springframework.org/schema/aop"
      xsi:schemaLocation="
      http://www.springframework.org/schema/beans
        classpath:/org/springframework/beans/factory/xml/spring-beans-3.0.xsd
      http://www.springframework.org/schema/context
      classpath:/org/springframework/context/config/spring-context-3.0.xsd
     http://www.springframework.org/schema/aop
     classpath:/org/springframework/aop/config/spring-aop-3.0.xsd
      ">
  <bean id="emailService" class="com.example.EmailService" />
<!-- Bean definition for MessageProcessor with constructor injection -->
  <bean id="messageProcessor" class="com.example.MessageProcessor">
     <constructor-arg ref="emailService"/>
  </bean>
</beans>
```



```java
package com.example;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
    public static void main(String[] args) {
        // Load the Spring context from the configuration file
        ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

        // Retrieve the MessageProcessor bean
        MessageProcessor messageProcessor = context.getBean("messageProcessor", MessageProcessor.class);

        // Call the method to send the message
        messageProcessor.processMessage();  // Should print "Email message sent"
    }
}
```

```
<terminated> MainApp (1) [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858\jre\bin\javaw.exe (13-Dec-2024, 12:49:18 am – 12:49:19 am) [pid: 10948]
log4j:WARN No appenders could be found for logger (org.springframework.context.support.ClassPathXmlApplicationContext).
log4j:WARN Please initialize the log4j system properly.
Email message sent
```

24. Develop a Spring MVC application to handle a simple "Hello World" requestresponse.Instructions:1. Create a controller HelloController with a method sayHello() mapped to URL /hello.2. Configure Spring MVC to handle this request and respond with a view displaying "Hello, World!".3. Implement a simple JSP view hello.jsp that displays the greeting message.4. Test the application by accessing http://localhost:8080/hello in web browser.

HelloController.java

```
package com.example.controller;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

@Controller

public class HelloController {

@RequestMapping("/hello")

   public String sayHello() {

      return "hello"; // This returns the view name 'hello.jsp'}}
```

hello.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

   pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>

<html>

<head>

   <title>Hello World</title>

</head>

<body>

   <p>Hello, World!</p>

</body>

</html>
```

servlet-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

      xsi:schemaLocation="http://www.springframework.org/schema/beans

                  http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
```

```
  <!-- Enable Spring MVC -->
  <context:component-scan base-package="com.example.controller" />
 <!-- View Resolver for JSP pages -->
  <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/views/" />
    <property name="suffix" value=".jsp" />
  </bean>
</beans>
```

web.xml

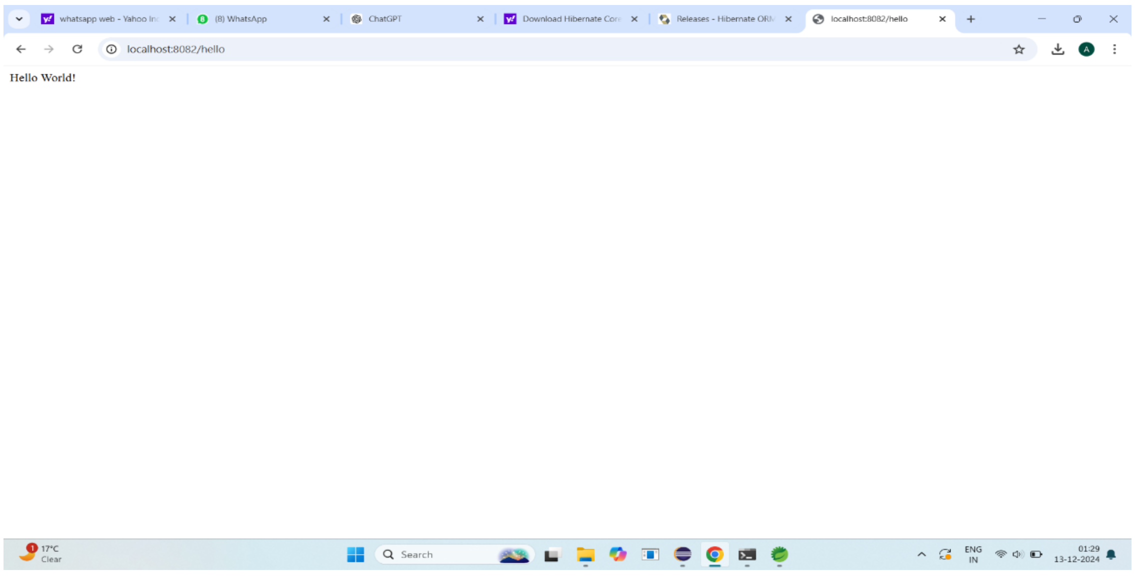
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
     version="3.0">
  <!-- Spring DispatcherServlet configuration -->
  <servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
 <groupId>com.example</groupId>
   <artifactId>spring-mvc-hello-world</artifactId>
   <version>1.0-SNAPSHOT</version>
   <packaging>war</packaging>
  <dependencies>
       <!-- Spring Web MVC -->
       <dependency>
           <groupId>org.springframework</groupId>
           <artifactId>spring-webmvc</artifactId>
           <version>5.3.25</version>
       </dependency>
  <!-- JSTL for JSP Views -->
       <dependency>
           <groupId>javax.servlet</groupId>
           <artifactId>javax.servlet-api</artifactId>
           <version>4.0.1</version>
           <scope>provided</scope>
       </dependency>
<!-- Spring Core -->
       <dependency>
           <groupId>org.springframework</groupId>
           <artifactId>spring-core</artifactId>
           <version>5.3.25</version>
       </dependency>
 <!-- Log4j for logging -->
       <dependency>
           <groupId>org.apache.logging.log4j</groupId>
           <artifactId>log4j-api</artifactId>
```

```
<version>2.14.1</version>

      </dependency>

   </dependencies>

<build>

      <plugins>

         <plugin>

            <groupId>org.apache.maven.plugins</groupId>

            <artifactId>maven-war-plugin</artifactId>

            <version>3.3.1</version>

         </plugin>

      </plugins>

   </build>s

</project>
```



Hello World!