I123039

5. Developing a multithreading - based project to Simulate a car parking management system with classes namely - RegistrearParking - Represents a parking request made by a car; parkingpool - Acts as a Shared synchronized queue; parking Agent - Represented a thread that Continously checks the pool and parks cars forom the queue and a Mainclass - Simulates N Cars arriving concurrently to request Parking.

Car ABC123 requested parking.

Car XYZ456 requested parking

Agent 1 parked car ABC123.

Agent 2 parked car XYZ456

...

Class Responsibilities

1. RegistrearParking (parking request)

- Represents a car requesting parking.
- Holds car number.

2. ParkingPool (Shared Resource)

- A synchronized queue.
- Cars are added by main thread.
- Cars are removed by ParkingAgent threads.

3. ParkingAgent (Consumer Thread)

- Continously checks the pool.
- parcks cars one by one.
- Multiple agents work concurrently.

4. Mainclass

- Simulates N car arriving concurrently.
- starts multiple parking agents.

1. RegistrarParking.java

```
class RegistrarParking {
    private String carNumber;
    public RegistrarParking (String carNumber) {
        this.carNumber = carNumber;
    }
    public String getcarNumber() {
        return carNumber;
    }
}
```

2. parkingPool.java

```
import java.util.LinkedList;
import java.util.Queue;

class parkingPool {
    private Queue<RegistrarParking> queue = new
                    LinkedList<>();
    public synchronized void addCar (RegistrarParking
                                        car) {
        queue.add (car);
    System.out.println("car "+car.getCarNumber()+"requested
                                        Parking.");
        notifyAll(); //notify waiting agents
    }
public synchronized RegistrarParking getCar(){
        while (queue.isEmpty()){
            try{ wait();
            } catch (InterruptedException e){
                e.printStackTrace();
            }
        return queue.poll();
    }}
```

3. parkingAgent.java

```
class parkingAgent extends Thread {
    private parkingPool pool;
    private int agentID;
public parkingAgent (parkingpool pool, int agentId){
        this.pool = pool;
        this.agentId = agentId;
}
@ Override
    public void run() {
      while (true) {
      Registrarparking car = pool.getcar();
System.out.println("Agent " + agentId + "parked car" +
                                car.getcarNumber() + ".");
try { Thread.sleep (1000); // simulate parking
  } catch (InterruptedException e){
      e.printStackTrace ();
  } }
}
```

4. Mainclass.java

```
Public class Mainclass {
  public static void main (String[] args){
    parking pool = new parkingPool ();
// start parking agents.
      ParkingAgent agent1 = new ParkingAgent(pool,1);
      ParkingAgent agent2 = new parkingAgent (pool,2)};
    agent1.Start();
    agent2.start();
String[] cars ={"ABC123","XY2456","LMN789","DEF456"
                "GHI999"};
for ( String carNumber : cars) {
   RegistrarParking car = new RegistrarParking (carNumber);
     pool.addCar (car);
try {
     Thread.sleep(500); // car arrival gap
   }
     catch (InterruptedExeception e) {
      e.printStackTrace();
```

```
    }
  }
}
}
```

Output :

```
Car ABC123 requested parking.
Car XYZ456 requested parking.
Agent 1 parked car ABC123.
Agent 2 parked car XYZ456.
Car LMN789 requested parking
Agent 1 parked car LMN789
car DEF456 requested parking
Agent 2 parked car DEF456
...
```

12. Describe how JDBC manages communicatio between a Java program application and a relational database. Outline the steps involved i executing a SELECT query and fetching results Include error handling with try-catch and finally blocks.

## How JDBC Manages Communication

JDBC (Java Database Connectivity) is an API that allows a Java application to connect, send SQL commands and receive results form a relational database.

JDBC works as a bridge between:

- Java Application
- JDBC API
- JDBC Driver
- Relational Database (MySQL / oracle / PostgreSQL)

Communication Flow :

Java Program
↓
JDBC API
↓
JDBC Driver
↓
Database
↓
ResultSet

The JDBC driver translates java method calls into database-specific SQL commands and returns the results back to java.

Steps to Execute a SELECT Query in JDBC

Step 1: Load JDBC Driver

Registers the driver with the DriverManager.

Step 2: Establish Connection

Connects java application to the database.

Step 3: Create Statement

used to send SQL queries.

Step 4: Execute SELECT Query

Executes SQL and returns a Resultset.

Step 5: process Resultset

Fetches rows column by column

Step 6: close Resources

prevents memory leaks.

JDBC SELECT Example with Error Handling

```
import java.sql.*;
public class jbs.JdbcSelectExample {
    public static void main (String[] args) {
        Connection con = null;
        Statement Stmt = null;
        Resultset rs = null;
    try {
        // Step: 1 Load JDBC Driven
    Class.forName ("com.mysql.cj.jdbc.Driver");
    // step 2 : Establish connection
```

```
Con = DriverManager.getConnection (
        "jdbc:Mysql://localhost:3306/studentdb",
         "root",
        "password"
    };
Stmt = con.createstatement ();
String SQL = " SELECT id, name, marks from
                        students";
 rs = stmt.executeQuery (sql);
while (rs.next ()) {
    int id = rs.getInt ("id");
    String name = rs.getString ("name");
    double marks = rs.getDouble ("marks");
   System.out.println (id + " " + name + " " + marks);
 } catch (ClassNotFoundExeception e) {
   System.out.println ("JDBC Driver not found!");
    e.printstackTrace ();
 } catch (SQLException e) {
System.out.println ("Database error occurred!");
    e.printStackTrace (); }
```

```
finally {
    try { if (rs! =null) rs.close();
          if (stmt!= null) stmt.close();
          if (con! =null) con.close();
        } catch (SQLException e) {
              e.printStackTrace();
    }
  }
}
```

try block

- Contains databases operation
- Risk of SQL or driver error

Catch block

- Handles:
- ClassNotfoundExpression → driver missing
- SQLException → Connection / query f. error.

finally block:

- always executes
- Ensures database resources are properly closed

Advantages of JDBC

(i) platform independent

(ii) Source secure database access

(iii) Support multiple databases

(iv) Handle large result sets

(v) Built-in exception handling.

(17) In a Java EE application, how does a servelet Controller manage the flow between the model and the view? provide a brief example the demonstrates forwarding data form a servlet to a JSP and rendering a response.

In a Java EE application:

Model:

- Contains business logic and data.
- Example: Java classes, database access.

Controller (servlet)

- Receives HTTP request
- Interacts with the model
- Places data into request
- Forwards the request to a JSP

View (JSP)

- Retrieves data form request
- Renders HTML response to the client

Request flow

Browser Request

↓

Servlet (Controller)

↓

Model (Business Logic / Data)

↓

Servlet sets attributes

↓

JSP (view)

↓

HTML Response

1. Servlet Controller (Controller)

```
import java.io.IOException;
import javax.servlet.*;
import javax.servlet.http.*;
public class StudentServlet extends HttpServlet {
  protected void doGet(HttpServletRequest request,
                       HttpServletResponse response)
    throws ServletException, IOException{
```

```
String studentName = "Afrin";
int marks = 85;
request.setAttribute("name", studentName);
request.setAttribute("marks", marks);
RequestDispatcher rd = request.getRequestDispatcher
("student.jsp");
rd.forward(request, response);
}
}
```

2. JSP View (student.jsp)

```
<html>
<head>
<title> student Result</title>
</head>
<body>
<h2> Student Details </h2>
<P> Name: ${name} </P>
<P> Mark: ${marks} </P>
```

```
<c:if test="${marks>=40}">
<p> status: pass </p>
</c:if>
</body>
</html>
```

[22] How does preparedStatement improve performance and security over Statement in JDBC? Write a short example to insert a record into a MySQL table using PreparedStatement.

PreparedStatement Improve performance.

- SQL query is precompiled by the database.
- Some query can be executed multiple times with different values.
- Reduces parsing and compilation overhead.
- Faster than statement for repeated execution.

PreparedStatement Improve Security.

- Uses parameterized queries.
- Prevents SQL Injection attacks.
- User input is treated as data, not SQL code.

Statement (unsafe)

```
String SQL = "INSERT INTO USER VALUES ('"+name+"','"+pass+"')
```

PreparedStatement (safe)

```
String SQL = "INSERT INTO user VALUES (?,?)";
```

Difference Between Statement and PreparedStatement.

| Feature | Statement | preparedstatement |
|---|---|---|
| SQL injection protection | No | Yes |
| Precompiled query | No | Yes |
| Performance | slower | faster |
| Parameter support | No | Yes |
| Reusability | Low | High |

M Insert Record using PreparedStatement (JDBC)

MySQL Table

```
CRETE TABLE Students (
    id INT,
    Name VARCHAR (50),
    marcks DOUBLE
);
```

```
import java.sql.*;
Publie class preparedstatementExample {
  Publie static void main (String[] args) {
        Connection con = nd null;
        preparedstatement ps = null;
      try {
            class.forName ("com.mysql.cj.jdbc.Driver");
            Con =DriverManagerr.getconnection(
                "jdbc:mysql://localhost:3306/studentdb",
                 "root"
                 "password"
              );
          String sql=INSERT INTO students(id,name,marks)VALUES
                                                  (?,?,?)";
          ps = con.preparedstatement(sql);

          ps.setInt (1, 101);
          ps.setString (2, "Afrin");
          ps.setDouble (3, 88.5);

          ps.executeupdate ();
System.out.println ("Record inserted Successfully!");
}.
```

```
catch(SQLException e){
    System.out.println("Database error!");
    e.printStackTrace();
}
finally{
    try{ if (ps!=null) ps.close();
         if (con!=null) con.close();
    }catch (SQLException e){
        e.printStackTrace();
    }
}
```

23. What is Resultset in JDBC and how is it used to retrieve data form a SQL MySQL database? Briefly explain the use of next(), getString(), and getINT() methods with an example.

Resultset in JDBC : A Resultset is an object in JDBC that holds the data returned by executing a SELECT query.

It works like a cursor that points to one row of data at a time.

- Initially, the cursor is positioned before the first row.
- We use next() to move row by row.
- Data is retrieved column-wise using getter methods.

Resultset is used:

next()

- Moves cursor to next row
- Result Return true if a row exists otherwise flase.

## Important Resultset Methods

### next()

- Move cursor to the next row.
- Returns true is a row exists, otherwise false.

### getString(columnName / columnIndex)

- Retrieves String data from the current row

### getInt(columnName / columnIndex)

- Retrieves integer data from the current row.

## Example: Retrieving Data from MySQL

MySQL Table (students)

| id | name | marks |
|---|---|---|
| 1 | Afrin | 85 |
| 2 | Mili | 80 |

```
import java.sql.*;
Public class ResultsetExample {
   public static void main (String [] args) {
      try {
         Class.forName ("com.mysql.cj.jdbc.Driver");
      Connection con = DriverManager.getConnection(
       "jdbc:mysql://localhost:3306/Studentdb",
        "root",
        "password".
      );
   Statement stmt = con.createStatement();
   ResultSet rs = stmt.executeQuery(
       "SELECT id, name, marks FROM Students"
    );
   While (rs.next())) {
        int id = rs.getInt ("id");
        String name = rs.getString ("name");
        int marks = rs.getInt ("marks");
   System.out.println (id + " " + name + " " + marks);
```