

**Layout Managers****PART-1**

Border, Grid, Flow, Card and Gridbag

**Layout Managers**

In Java AWT (Abstract Window Toolkit), a **Layout Manager** is responsible for **arranging** and **positioning** components (buttons, text fields, labels, etc.) inside a container (Frame, Panel, etc.).

**Use of Layout Managers**

- **Automatic Component Arrangement** → No need for manual positioning.
- **Resizes Components Dynamically** → Adjusts component sizes when the window is resized.
- **Platform Independence** → Works uniformly on different screen sizes.

**Types of Layout Managers**

Layout Manager	Description
<b>BorderLayout</b>	Divides container into five regions: NORTH, SOUTH, EAST, WEST, CENTER.
<b>FlowLayout</b>	Arranges components <b>in a row</b> (default for Panel).
<b>GridLayout</b>	Arranges components in <b>rows and columns</b> (equal size).
<b>CardLayout</b>	Allows switching between <b>multiple screens (cards)</b> .
<b>GridBagLayout</b>	Flexible <b>grid-based layout</b> allowing uneven cell sizes.

## 1. BorderLayout (Default for Frames)

BorderLayout is one of the **most commonly used layout managers** in **Java AWT (Abstract Window Toolkit)**. It divides the container into **five fixed regions** where components can be placed.

Region	Constant Used	Description
<b>NORTH</b>	BorderLayout.NORTH	Placed at the <b>top</b> of the container.
<b>SOUTH</b>	BorderLayout.SOUTH	Placed at the <b>bottom</b> of the container.
<b>EAST</b>	BorderLayout.EAST	Placed at the <b>right</b> side.
<b>WEST</b>	BorderLayout.WEST	Placed at the <b>left</b> side.
<b>CENTER</b>	BorderLayout.CENTER	Occupies the <b>remaining space</b> in the middle.

### Key Features of BorderLayout

- **Default Layout for Frame, Dialog, and Window.**
- **Expands components** to fill the region where they are placed.
- **Only one component per region** (use a Panel for multiple components).
- **Flexible resizing** when the window size changes.

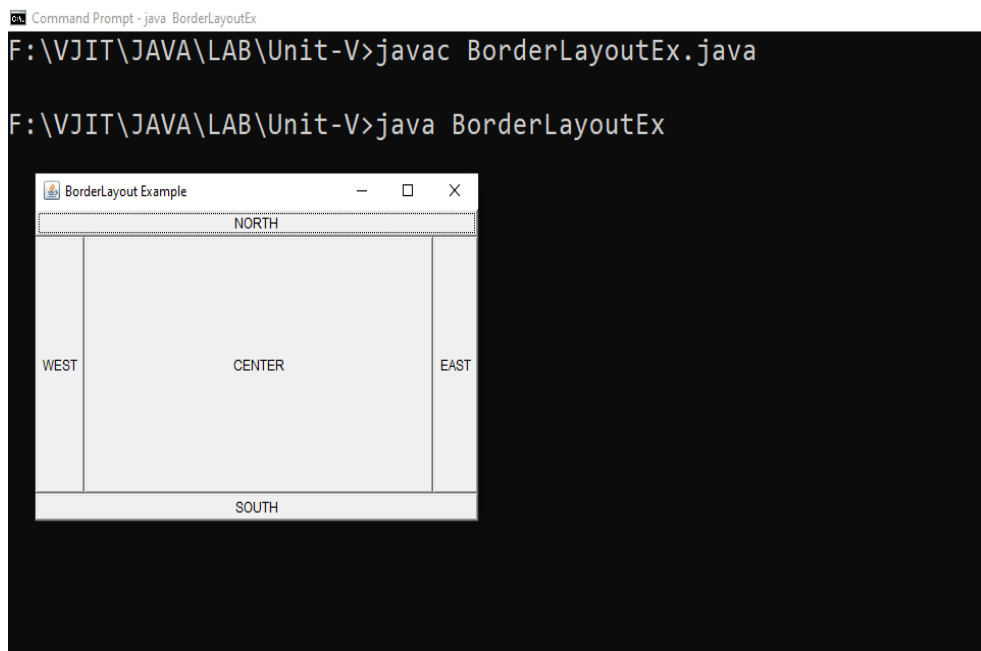
```
import java.awt.*;
```

```
public class BorderLayoutEx
{
    public static void main(String[] args)
    {
        // Creating a frame
        Frame frame = new Frame("BorderLayout Example");

        // Setting BorderLayout (default for Frame)
        frame.setLayout(new BorderLayout());
    }
}
```

```
// Adding buttons to different regions
frame.add(new Button("NORTH"), BorderLayout.NORTH);
frame.add(new Button("SOUTH"), BorderLayout.SOUTH);
frame.add(new Button("EAST"), BorderLayout.EAST);
frame.add(new Button("WEST"), BorderLayout.WEST);
frame.add(new Button("CENTER"), BorderLayout.CENTER);

// Frame settings
frame.setSize(450, 300);
frame.setVisible(true);
}
}
```



## 2. GridLayout

GridLayout is a layout manager that places components in a **rectangular grid** of rows and columns. Every component in the container is given the **same size**, and the container is divided evenly into the specified number of **rows and columns**.

### Key Features of GridLayout:

Feature	Description
<b>Equal Size Cells</b>	All components are given equal size regardless of their content.
<b>Row x Column Format</b>	You define the number of rows and columns in the grid.
<b>Fills Left to Right, Top to Bottom</b>	Components are placed row by row.
<b>No overlapping</b>	Each grid cell holds <b>only one component</b> .

### Constructor of GridLayout

```
GridLayout(int rows, int cols)
```

```
GridLayout(int rows, int cols, int hgap, int vgap)
```

- **rows** – number of rows
- **cols** – number of columns
- **hgap** – horizontal space between components
- **vgap** – vertical space between components

If rows = 0 → rows are determined by the number of components and columns.

```
import java.awt.*;

public class GridLayoutEx
{
    public static void main(String[] args)
    {
        Frame frame = new Frame("Calculator Layout");

        // 4 rows x 4 columns with spacing
        frame.setLayout(new GridLayout(4, 4, 5, 5));

        String[] buttons = {
            "7", "8", "9", "/",
            "4", "5", "6", "*",
            "1", "2", "3", "-",
            "0", ".", "=", "+"
        };

        for (String b : buttons)
        {
            frame.add(new Button(b));
        }

        frame.setSize(400, 300);
        frame.setVisible(true);
    }
}
```



### 3. FlowLayout

FlowLayout is the **simplest layout manager** in Java AWT. It arranges components **in a row**, and when no more space is left, it wraps them to the **next line** (like text in a paragraph).

#### Key Features of FlowLayout:

Feature	Description
<b>Alignment</b>	Left, center (default), or right alignment.
<b>Component Wrapping</b>	Automatically wraps components to a new line when space runs out.
<b>Spacing</b>	You can set horizontal and vertical gaps between components.
<b>Flexibility</b>	Easy to use and perfect for small layouts.

#### Constructors of FlowLayout:

```
FlowLayout()
```

```
FlowLayout(int alignment)
```

```
FlowLayout(int alignment, int hgap, int vgap)
```

- alignment — FlowLayout.LEFT, CENTER, or RIGHT
- hgap — Horizontal gap (space) between components
- vgap — Vertical gap between rows of components

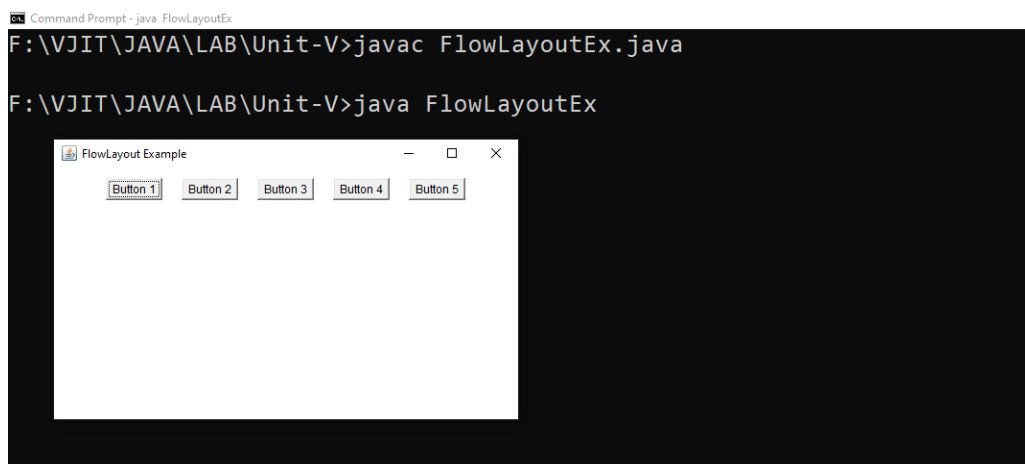
```
import java.awt.*;

public class FlowLayoutEx
{
    public static void main(String[] args)
    {
        // Create a new frame
        Frame frame = new Frame("FlowLayout Example");

        // Set FlowLayout with center alignment
        frame.setLayout(new FlowLayout(FlowLayout.CENTER, 20, 10));

        // Add some buttons
        frame.add(new Button("Button 1"));
        frame.add(new Button("Button 2"));
        frame.add(new Button("Button 3"));
        frame.add(new Button("Button 4"));
        frame.add(new Button("Button 5"));

        // Frame settings
        frame.setSize(500, 300);
        frame.setVisible(true);
    }
}
```



## 4. CardLayout

CardLayout is a layout manager in Java AWT that allows **multiple components (cards) to be stacked on top of each other**, with only **one visible at a time**. It is useful for creating **wizard-like interfaces, tabbed panels, and step-by-step forms**.

### Key Features of CardLayout

Feature	Description
<b>Stacked Layout</b>	Only one component (card) is visible at a time.
<b>Navigation Methods</b>	Allows switching between cards using <code>.next()</code> , <code>.previous()</code> , <code>.show()</code> , etc.
<b>Best for Wizards/Forms</b>	Useful for multi-step forms, tutorials, and tab-like interfaces.
<b>Flexible</b>	Can contain multiple panels or other components.

### Constructor of CardLayout

```
CardLayout()
CardLayout(int hgap, int vgap)
```

- hgap – Horizontal space between components.
- vgap – Vertical space between components.

```
import java.awt.*;
import java.awt.event.*;

public class CardLayoutEx
{
    public static void main(String[] args)
    {
        // Create a new frame
        Frame frame = new Frame("CardLayout Example");
        CardLayout cardLayout = new CardLayout();
```



```
// Create a Panel to hold the cards
Panel cardPanel = new Panel();
cardPanel.setLayout(cardLayout);

// Creating Cards (Panels with different components)
Panel card1 = new Panel();
card1.add(new Label("This is Card 1"));
Button next1 = new Button("Next");
card1.add(next1);

Panel card2 = new Panel();
card2.add(new Label("This is Card 2"));
Button next2 = new Button("Next");
Button previous2 = new Button("Previous");
card2.add(previous2);
card2.add(next2);

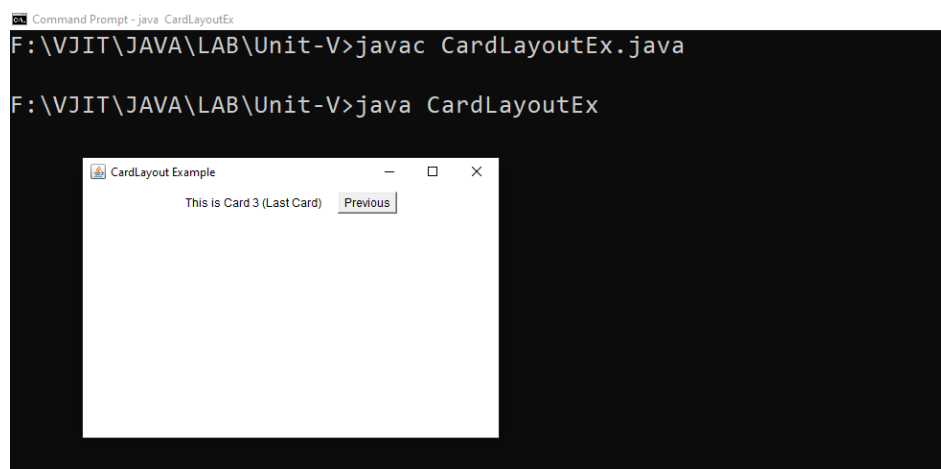
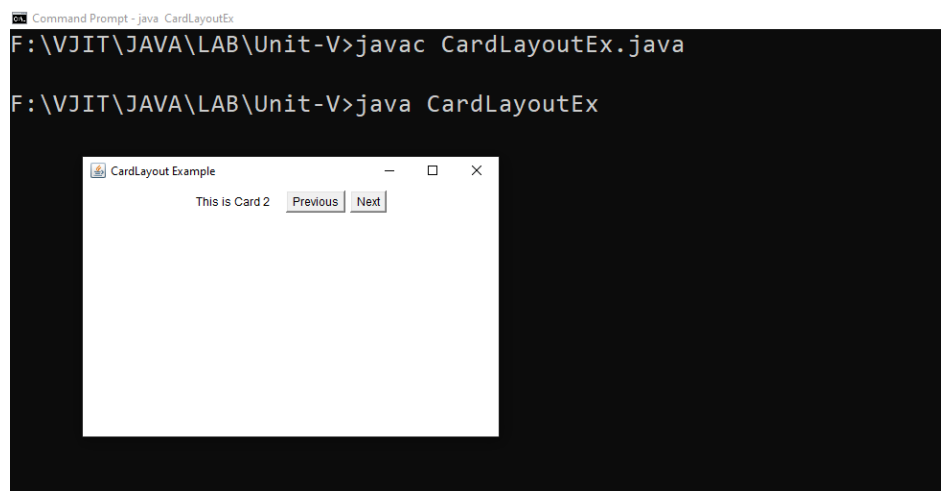
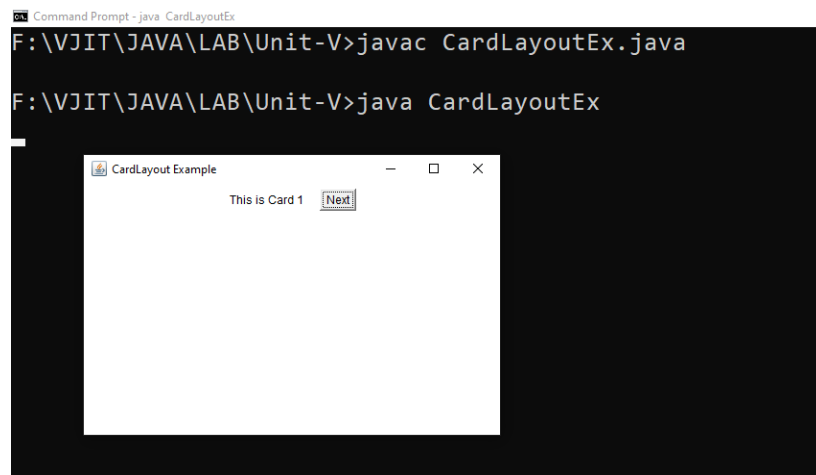
Panel card3 = new Panel();
card3.add(new Label("This is Card 3 (Last Card)"));
Button previous3 = new Button("Previous");
card3.add(previous3);

// Add cards to cardPanel
cardPanel.add(card1, "Card1");
cardPanel.add(card2, "Card2");
cardPanel.add(card3, "Card3");

// Event Listeners for Navigation
next1.addActionListener(e -> cardLayout.next(cardPanel));
next2.addActionListener(e -> cardLayout.next(cardPanel));
previous2.addActionListener(e -> cardLayout.previous(cardPanel));
previous3.addActionListener(e -> cardLayout.previous(cardPanel));

// Add cardPanel to Frame
frame.add(cardPanel);

// Frame Settings
frame.setSize(450, 300);
frame.setVisible(true);
}
}
```



## 5. GridBagLayout

GridBagLayout is the **most flexible and powerful layout manager** in Java AWT. It arranges components in a **grid** (like GridLayout), but **each cell can have a different size**, allowing precise control over positioning.

### Key Features of GridBagLayout

Feature	Description
<b>Flexible Layout</b>	Allows rows and columns of different sizes.
<b>Precise Positioning</b>	Components can span multiple rows/columns.
<b>Weighting System</b>	Components can grow/shrink dynamically.
<b>Constraints-Based</b>	Uses GridBagConstraints to control placement.
<b>Spacing Control</b>	Allows setting gaps between components.

### Constructor of GridBagLayout

GridBagLayout()

This constructor creates an instance of GridBagLayout, but components need **constraints (GridBagConstraints)** for proper alignment.

### GridBagConstraints

GridBagConstraints is a helper class used to specify **placement, size, and behavior** of components in GridBagLayout.

Constraint	Description
gridx, gridy	Specifies the row and column position.
gridwidth, gridheight	Determines how many columns/rows a component spans.
weightx, weighty	Controls how much space a component gets when resizing.
anchor	Defines the component's alignment (e.g., CENTER, EAST, WEST).
fill	Determines how the component should expand (HORIZONTAL, VERTICAL, BOTH).
insets	Specifies padding around the component.

```
import java.awt.*;
import java.awt.event.*;

public class GridBagEx
{
    public static void main(String[] args)
    {
        Frame frame = new Frame("Login Form");
        frame.setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();

        // Labels and TextFields
        Label uname = new Label("Username:");
        TextField user = new TextField(15);
        Label pwd = new Label("Password:");
        TextField password = new TextField(15);
        password.setEchoChar('*');

        // Buttons
        Button loginButton = new Button("Login");
        Button cancelButton = new Button("Cancel");

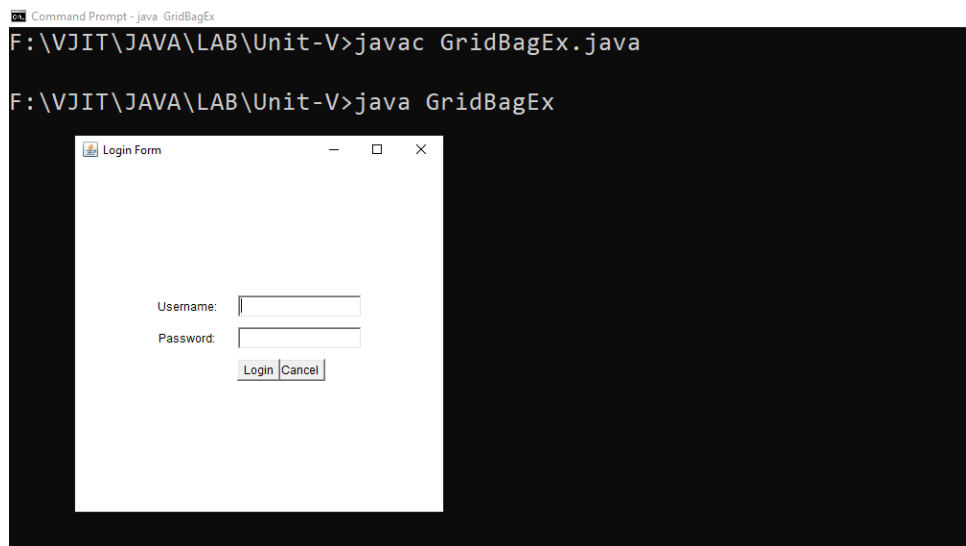
        // Constraints for Labels and Fields
        gbc.insets = new Insets(5, 5, 5, 5); // Padding

        gbc.gridx = 0; gbc.gridy = 0;
        frame.add(uname, gbc);
        gbc.gridx = 1; gbc.gridy = 0;
        frame.add(user, gbc);

        gbc.gridx = 0; gbc.gridy = 1;
        frame.add(pwd, gbc);
        gbc.gridx = 1; gbc.gridy = 1;
        frame.add(password, gbc);

        // Constraints for Buttons
        gbc.gridx = 0; gbc.gridy = 2;
```

```
gbc.gridwidth = 2;  
gbc.anchor = GridBagConstraints.CENTER; // Centering buttons  
frame.add(loginButton, gbc);  
  
gbc.gridx = 1; gbc.gridy = 2;  
frame.add(cancelButton, gbc);  
  
// Frame settings  
frame.setSize(400, 400);  
frame.setVisible(true);  
}  
}
```

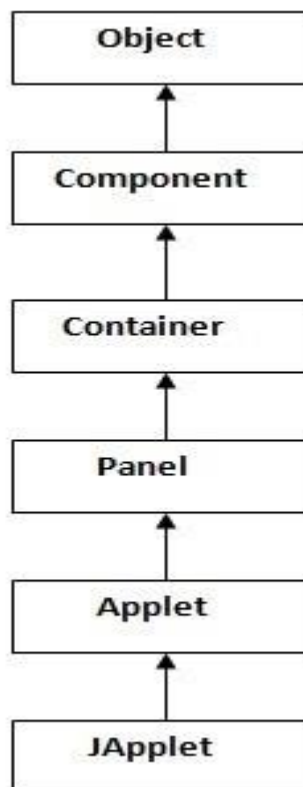


**Applets****PART-2**

Concepts of Applets, life cycle of an applet, creating applets, passing parameters to applets.

Applets are **small Java programs** that can be embedded in web pages and run in **Java-enabled web browser** or an **appletviewer tool**.

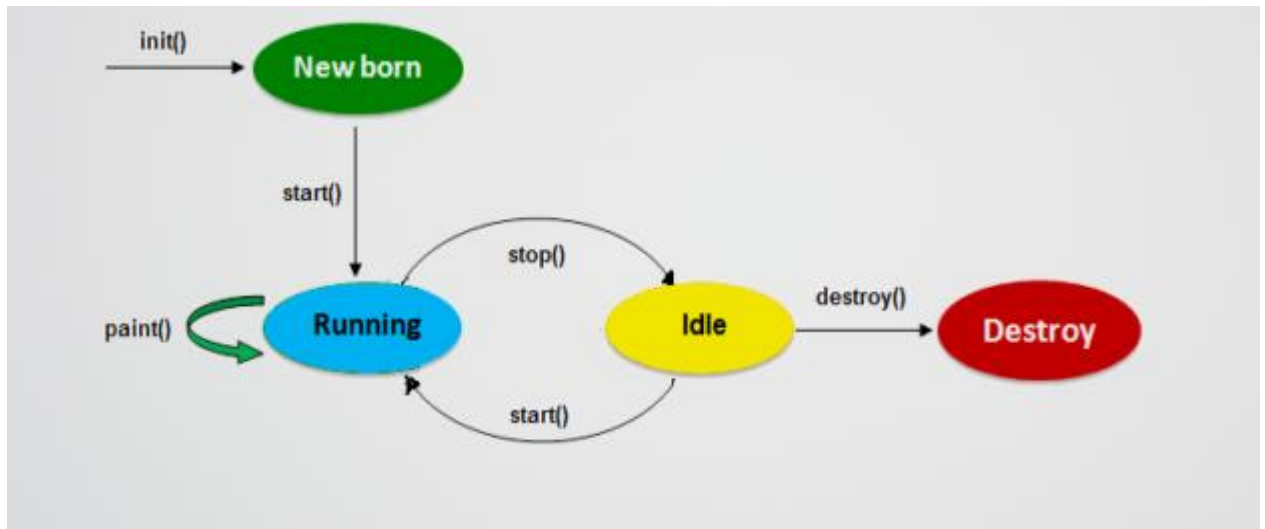
Although applets are now considered outdated due to security concerns and lack of browser support.

**Hierarchy of Applet****java.applet.Applet class**

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

## Lifecycle of an Applet

The applet life cycle can be defined as the process of how the object is created, started, stopped, and destroyed during the entire execution of its application. It basically has five core methods namely `init()`, `start()`, `stop()`, `paint()` and `destroy()`. These methods are invoked by the browser to execute.



The lifecycle of an applet is managed by the browser (or appletviewer tool) and involves the following methods:

Method	Description
<code>init()</code>	Called once when the applet is first loaded. Initialization tasks like setting up UI components are done here.
<code>start()</code>	Called after <code>init()</code> and also each time the applet is restarted (e.g., when the user revisits the page).
<code>paint(Graphics g)</code>	Called to redraw the content of the applet. All GUI drawing happens here.
<code>stop()</code>	Called when the user leaves the web page or switches to another tab.
<code>destroy()</code>	Called when the applet is being removed completely from memory. Cleanup is done here.

## Creating Applets, Passing Parameters to Applets

Applets are Java programs that extend the `java.applet.Applet` class

### Steps to Create a Simple Applet

1. Import necessary packages (`java.applet.*`, `java.awt.*`)
2. Extend the Applet class.
3. Override one or more lifecycle methods (`init()`, `start()`, `paint()`, etc.)
4. Compile the .java file.
5. Create an HTML file with the `<applet>` tag (or use `appletviewer` to test).

Program to illustrate **applet life cycle** and parameter passing.

```
import java.applet.Applet;
import java.awt.*;
import java.util.Date;
/*
<applet code="AppletLifeCycle" width=300 height=200>
  <param name="greeting" value="Welcome to Java Applet">
  <param name="author" value="VJIT">
</applet>
*/
public class AppletLifeCycle extends Applet
{
    // Instance variables to store parameter values
    private String greetingMessage;
    private String author;

    // Called when the applet is first loaded into memory
    @Override
    public void init()
    {
        // Retrieve parameters from the applet tag
        greetingMessage = getParameter("greeting"); // Get greeting message
        author = getParameter("author"); // Get author name
    }
}
```



```
        if (greetingMessage == null)
        {
            greetingMessage = "Hello, this is a default greeting!";
        }
        if (author == null)
        {
            author = "Unknown Author";
        }

        System.out.println("Applet Initialized");
    }

    // Called when the applet is started or brought to the foreground
    @Override
    public void start()
    {
        System.out.println("Applet Started");
    }

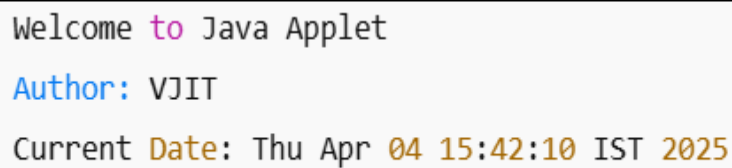
    // Called to display the applet on the screen
    @Override
    public void paint(Graphics g)
    {
        // Display greeting message and author name on the applet
        g.drawString(greetingMessage, 20, 50);
        g.drawString("Author: " + author, 20, 70);

        // Display current date and time
        g.drawString("Current Date: " + new Date(), 20, 90);
    }

    // Called when the applet is stopped (i.e., the user navigates away)
    @Override
    public void stop()
    {
        System.out.println("Applet Stopped");
    }
}
```

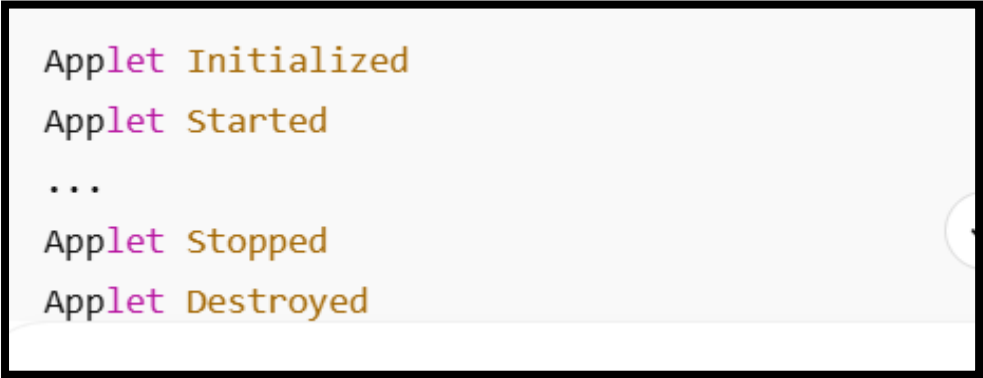
```
// Called when the applet is destroyed  
@Override  
public void destroy()  
{  
    System.out.println("Applet Destroyed");  
}
```

### In Browser



```
Welcome to Java Applet  
Author: VJIT  
Current Date: Thu Apr 04 15:42:10 IST 2025
```

### In Command Prompt



```
Applet Initialized  
Applet Started  
...  
Applet Stopped  
Applet Destroyed
```

**JDBC Connectivity****PART-3**

*JDBC Type 1 to 4 Drivers, connection establishment, Query Execution.*

**JDBC (Java Database Connectivity)** is a Java API that enables Java programs to **connect to and interact with databases** (such as MySQL, Oracle, SQL Server, PostgreSQL, etc.).

We can use JDBC API to access tabular data stored in any relational database. By the help of JDBC API, we can perform CRUD(Create, Read, Update, Delete) operations from the database. It is like Open Database Connectivity (ODBC) provided by Microsoft.

**JDBC Architecture**

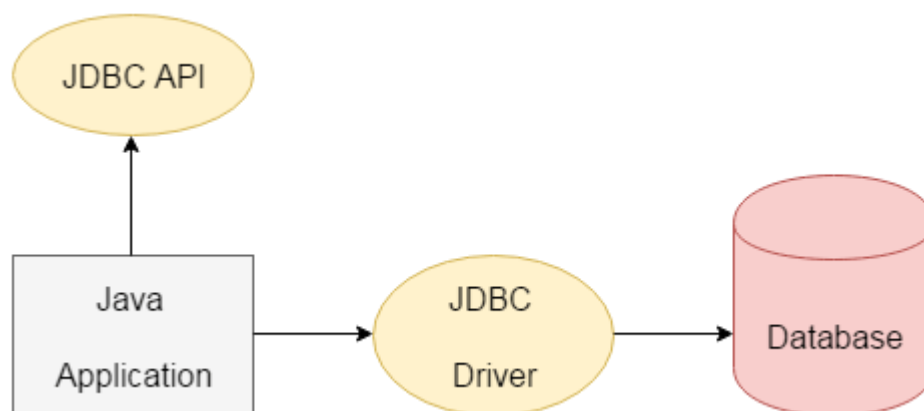
The JDBC architecture consists of two layers:

1. **JDBC API** (Application Level)

- Provides classes and interfaces like Connection, Statement, ResultSet, etc.
- Used by Java applications.

2. **JDBC Driver** (Implementation Level)

- Handles the communication between Java application and the database.
- Converts Java calls into DB-specific calls.



## JDBC Drivers

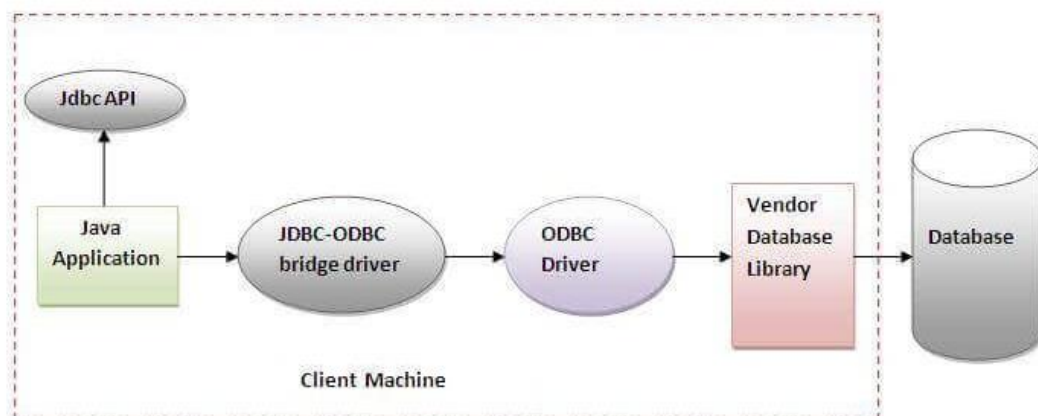
JDBC Driver is a software component that enables java application to interact with the database.

There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol Driver (Middleware Driver)
4. Thin Driver (Pure Java Driver)

### 1. Type 1: JDBC-ODBC Bridge Driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.



#### Advantages:

- Easy to use with ODBC-supported databases.

#### Disadvantages:

- Platform dependent (requires native ODBC driver).
- Performance is poor.
- Deprecated in Java 8 and removed in later versions.

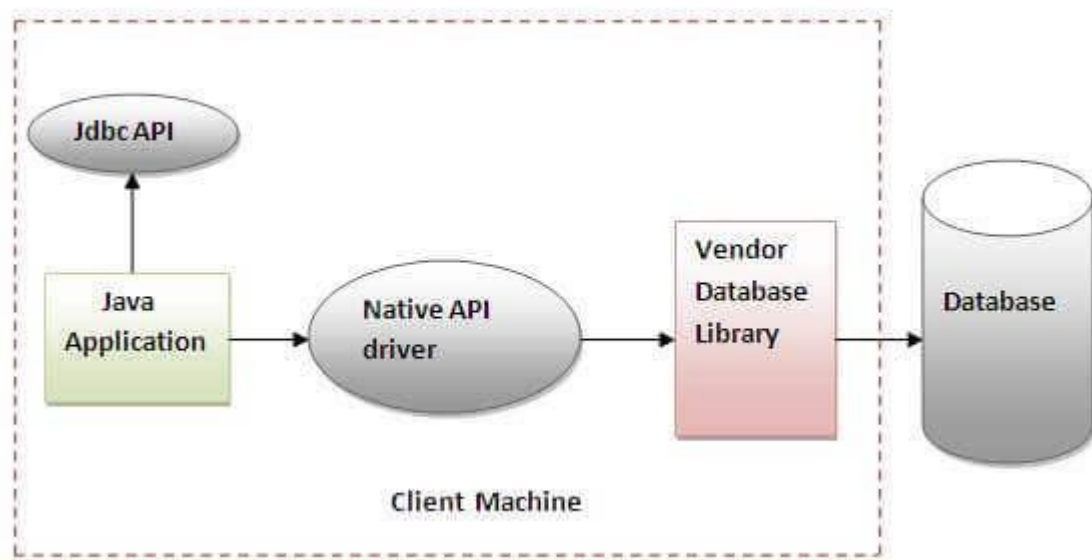
#### Example Driver Class:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

## 2. Type 2: Native-API Driver

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

- Uses **native C/C++ libraries** provided by the database vendor.
- JDBC calls → Native API calls → Database



### Advantages:

- Better performance than Type 1.
- Can use database-specific features.

### Disadvantages:

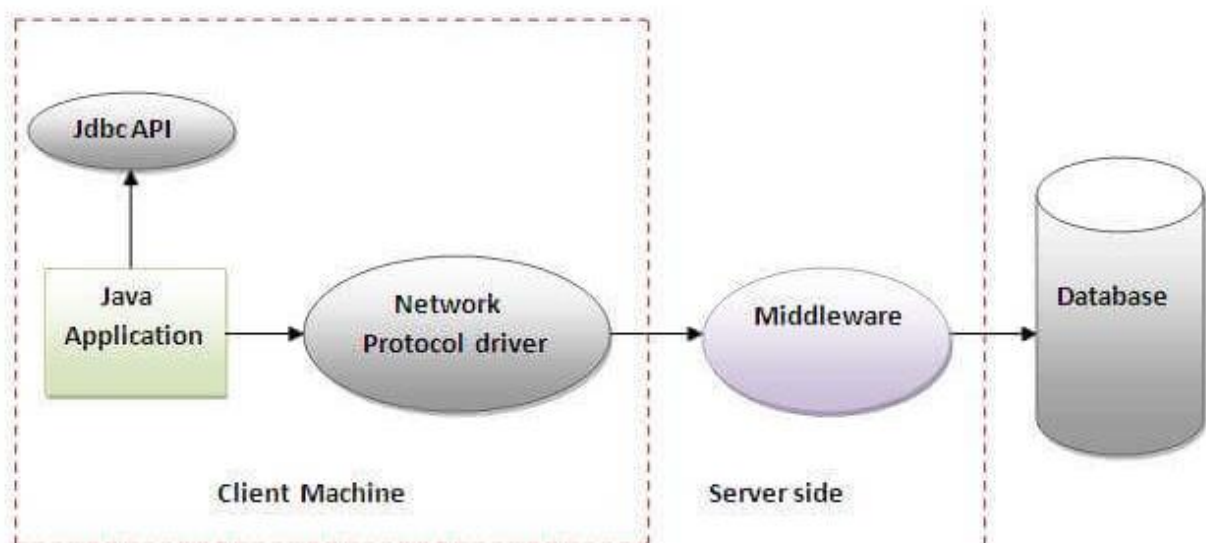
- Requires native libraries on client machine.
- Platform dependent.

**Example:** Oracle OCI (Oracle Call Interface) Driver

### 3. Type 3: Network Protocol Driver (Middleware Driver)

The Network Protocol Driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

- JDBC calls → Middleware Server → Database
- Uses a **Server-Based Middleware** that translates requests to DB calls.



#### Advantages:

- Platform independent.
- No Client-Side DB driver needed.
- Good for Intranet-Based Applications.

#### Disadvantages:

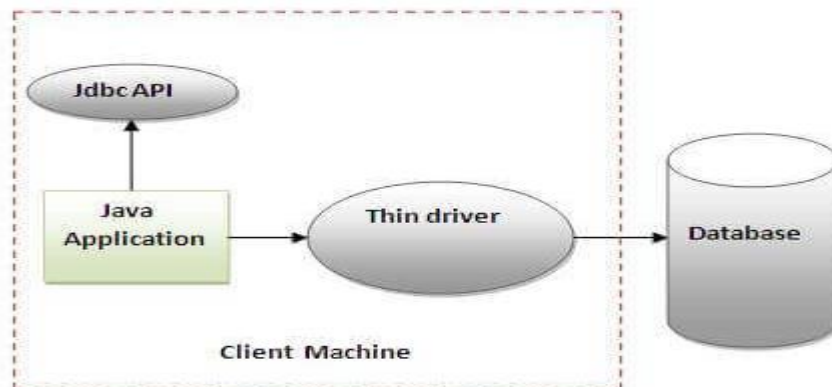
- Requires Middleware Server.
- Slightly complex setup.

**Example:** Informix Dynamic Server (IDS)

#### 4. Type 4: Thin Driver (Pure Java Driver)

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

- JDBC calls → Direct communication with the database using **Java Sockets**
- **No Native Libraries or Middleware** required



#### Advantages:

- Pure Java → Platform Independent.
- Best performance.
- Most commonly used today (e.g., MySQL Connector/J, Oracle Thin Driver).

#### Disadvantages:

- One driver per database type.

#### Example Driver Class:

```
Class.forName("com.mysql.cj.jdbc.Driver"); // For MySQL 8+
```

#### Comparison of JDBC Driver Types

Type	Name	Platform Dependent	Performance	Use in Modern Apps
Type-1	JDBC-ODBC Bridge	Yes	Low	✗Deprecated
Type-2	Native-API	Yes	Medium	✗Rare
Type-3	Network Protocol	No	Medium	✗Rare
Type-4	Thin Driver (Pure Java)	No	High	✓Recommended

## Common JDBC Classes & Interfaces

Class	Description
DriverManager	Manages the set of available JDBC drivers. Establishes a connection with a database using a driver.
Interfaces	Description
Connection	Represents an active connection to a database. Used to create statements and manage transactions.
Statement	Used to execute static SQL queries (e.g., SELECT * FROM table).
ResultSet	Represents the result of an SQL SELECT query. Allows data navigation and extraction.
PreparedStatement	Precompiled SQL statement (used for dynamic queries with parameters)
SQLException	Handles SQL errors

## Commonly Used Methods in JDBC Classes and Interfaces

### 1. DriverManager Class

Manages a list of database drivers and establishes connections to a database.

Method	Description
getConnection(String url)	Attempts to establish a connection using the provided URL.
getConnection(String url, String user, String password)	Connects with the specified username and password.
registerDriver(Driver driver)	Registers a JDBC driver with the DriverManager.

### Example:

```
Connection con = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/testdb", "root", "password");
```



## 2. Connection Interface

Represents a connection to the database.

Method	Description
createStatement()	Creates a basic SQL statement.
prepareStatement(String sql)	Creates a precompiled SQL statement.
commit()	Commits the current transaction.
rollback()	Rolls back the current transaction.
close()	Closes the connection.
isClosed()	Checks if the connection is closed.

## 3. Statement Interface

Used to execute static SQL statements (without parameters).

Method	Description
executeQuery(String sql)	Executes a SELECT query and returns a ResultSet.
executeUpdate(String sql)	Executes INSERT, UPDATE, DELETE and returns affected rows.
execute(String sql)	Executes any SQL command (returns boolean for result type).
close()	Closes the statement.

#### 4. PreparedStatement Interface

Used for executing **parameterized** queries (safer and faster).

Method	Description
setString(int paramIndex, String value)	Sets a String parameter at the specified index.
setInt(int paramIndex, int value)	Sets an int parameter.
setDouble(int paramIndex, double value)	Sets a double parameter.
executeQuery()	Executes SELECT statements.
executeUpdate()	Executes INSERT, UPDATE, or DELETE statements.

#### 5. ResultSet Interface

Used to retrieve and navigate results from a query.

Method	Description
next()	Moves the cursor to the next row (returns false if no more rows).
getString(String columnLabel)	Retrieves String data from a column.
getInt(String columnLabel)	Retrieves int data from a column.
getDouble(String columnLabel)	Retrieves double data.
wasNull()	Checks if last column read was SQL NULL.
close()	Closes the ResultSet.

#### 6. SQLException Class

Used to handle database access errors.

Method	Description
getMessage()	Returns the error message.
getErrorCode()	Returns vendor-specific error code.
getSQLState()	Returns the SQLState string.
printStackTrace()	Prints the exception trace.

## Connection Establishment to Data Base Server

### Steps to Use in JDBC to connect to MySql Server

#### 1. Load the Driver Class

```
Class.forName("com.mysql.jdbc.Driver");
```

#### 2. Establish the Connection

```
Connection con = DriverManager.getConnection(  
    "jdbc:mysql://localhost:3306/mydb", "username", "password");
```

#### 3. Create a Statement and Execute a Query

```
Statement stmt = con.createStatement();  
ResultSet rs = stmt.executeQuery("SELECT * FROM students");
```

#### 4. Process the Results

```
while (rs.next())  
{  
    System.out.println(rs.getString("name"));  
}
```

#### 5. Close the Connection

```
con.close();
```

**1. Java Program to CREATE Database Table in MySql Server**

*// This code is for establishing connection with MySQL database and retrieving data*

*// from database through JDBC*

*/\*1. import ---> java.sql*

*\*2. load and register the driver ---> com.jdbc.*

*\*3. create connection*

*\*4. create a statement*

*\*5. execute the query*

*\*6. process the results*

*\*7. close*

*\*/*

import java.io.\*;

import java.sql.\*;

class **create**

{

public static void **main**(String[] args) throws ClassNotFoundException, SQLException

{

String url= "jdbc:mysql://localhost:3306/vjit"; // Database details

String username = "root"; // MySQL credentials

String password = "vjit";

Class.forName("com.mysql.cj.jdbc.Driver"); // Driver name

Connection con = DriverManager.getConnection(url, username, password);

System.out.println("Connection Established successfully");

Statement st = con.createStatement();

```
String sql = "CREATE TABLE STUDENTS(Roll_No INTEGER NOT  
NULL,First_Name VARCHAR(255),Last_Name VARCHAR(255),Age  
INTEGER,PRIMARY KEY ( Roll_No ))";
```

```
st.execute(sql);
```

```
System.out.println("Created table in given database...");
```

```
st.close(); // close statement
```

```
con.close(); // close connection
```

```
System.out.println("Connection Closed....");
```

```
}
```

```
}
```

### Output:

MySQL 8.0 Command Line Client

```
mysql> show databases;  
+-----+  
| Database |  
+-----+  
| ceds     |  
| customer-sale |  
| employeedb |  
| information_schema |  
| mysql     |  
| performance_schema |  
| sakila    |  
| sys       |  
| vjit      |  
| world     |  
+-----+  
10 rows in set (0.01 sec)  
  
mysql> use vjit;  
Database changed  
mysql> ■
```

Command Prompt

```
F:\VJIT\JAVA\LAB\Week13>javac create.java
```

```
F:\VJIT\JAVA\LAB\Week13>java create
Connection Established successfully
Created table in given database...
Connection Closed....
```

```
F:\VJIT\JAVA\LAB\Week13>
```

MySQL 8.0 Command Line Client

```
mysql> show tables;
```

```
+-----+
| Tables_in_vjit |
+-----+
| allotment      |
| bill_details_view |
| book           |
| cassette       |
| daily_issues_view |
| daily_sales_view |
| employee       |
| employee_adding |
| employee_deletions |
| employee_net_salary |
| employee_pay_details |
| iss_rec_view   |
| issue_details  |
| issues_date_wise |
| salary_changes |
| student_machine_allocations |
| students       |
| thursday_machine_allocations |
+-----+
```

```
18 rows in set (0.00 sec)
```

```
mysql>
```

MySQL 8.0 Command Line Client

```
mysql> desc students;
```

```
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| Roll_No    | int       | NO   | PRI | NULL    |       |
| First_Name | varchar(255) | YES  |     | NULL    |       |
| Last_Name  | varchar(255) | YES  |     | NULL    |       |
| Age        | int       | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
```

```
4 rows in set (0.01 sec)
```

```
mysql>
```

## 2. Java Program to INSERT Record into Database Table in MySql Server

```
import java.sql.*;

class StudentInsertionApplication
{
    public static void main(String[] args) throws ClassNotFoundException,SQLException
    {
        String url= "jdbc:mysql://localhost:3306/vjit"; // Database details
        String username = "root"; // MySQL credentials
        String password = "vjit";
        Class.forName("com.mysql.cj.jdbc.Driver"); // Driver name
        Connection con = DriverManager.getConnection(url, username, password);
        System.out.println("Connection Established successfully");
        Statement st = con.createStatement();
        int c = st.executeUpdate("insert into students values(6703,'Rahul','HYD',20)");
        System.out.println(c + "\t Student Record inserted successfully");
        st.close();
        con.close();
    }
}
```

### Output:

Command Prompt

```
F:\VJIT\JAVA\LAB\Week13>javac insert.java
F:\VJIT\JAVA\LAB\Week13>java StudentInsertionApplication
Connection Established successfully
1      Student Record inserted successfully
F:\VJIT\JAVA\LAB\Week13>_
```

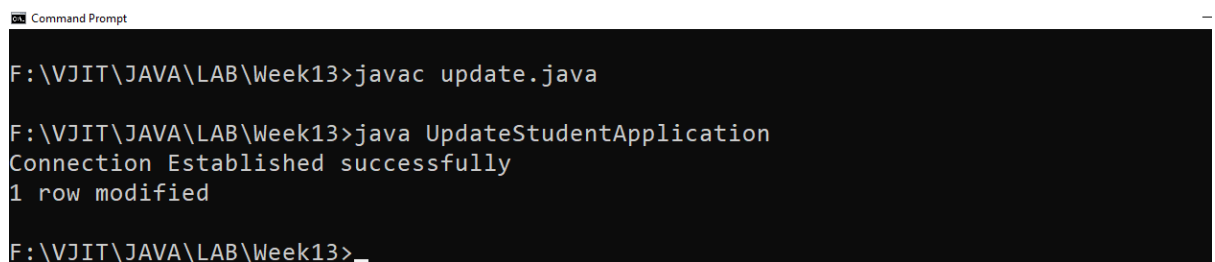
MySQL 8.0 Command Line Client

```
mysql> select * from students;
+-----+-----+-----+-----+
| Roll_No | First_Name | Last_Name | Age |
+-----+-----+-----+-----+
| 6703 | Rahul | HYD | 20 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

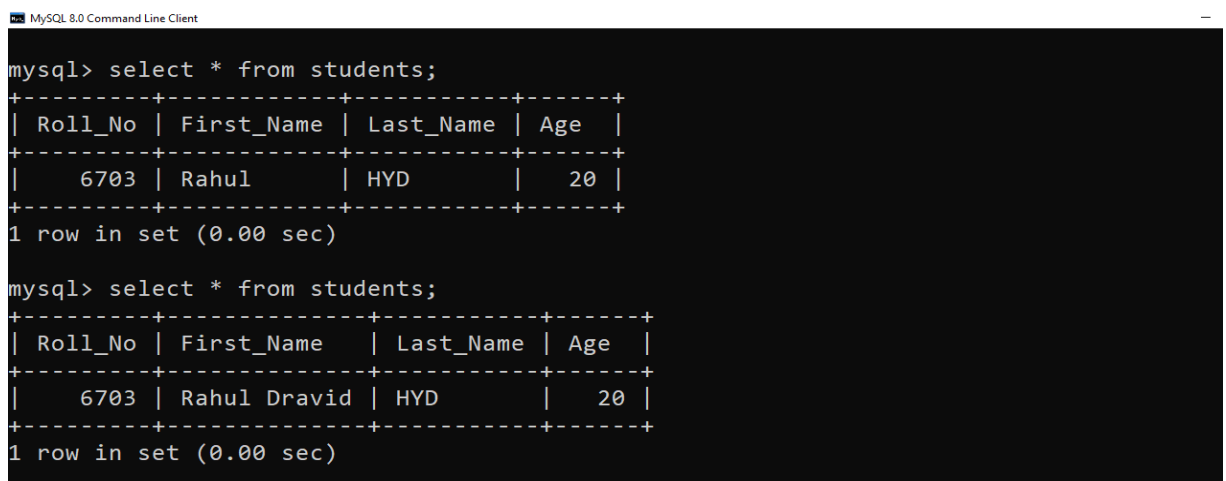
mysql>
```

**3. Java Program to UPDATE Record into Database Table in MySql Server**

```
import java.sql.*;
class UpdateStudentApplication
{
public static void main (String[]args) throws ClassNotFoundException, SQLException
{
    String url= "jdbc:mysql://localhost:3306/vjit"; // Database details
    String username = "root"; // MySQL credentials
    String password = "vjit";
    Class.forName("com.mysql.cj.jdbc.Driver"); // Driver name
    Connection con = DriverManager.getConnection(url, username, password);
    System.out.println("Connection Established successfully");
    Statement st = con.createStatement();
    int rows = st.executeUpdate("update students set First_Name = 'Rahul Dravid' where
    Roll_No=6703");
    System.out.println(rows + " row modified");
    st.close();
    con.close();
}
}
```

**Output:**

```
F:\VJIT\JAVA\LAB\Week13>javac update.java
F:\VJIT\JAVA\LAB\Week13>java UpdateStudentApplication
Connection Established successfully
1 row modified
F:\VJIT\JAVA\LAB\Week13>
```



```
mysql> select * from students;
+-----+-----+-----+-----+
| Roll_No | First_Name | Last_Name | Age |
+-----+-----+-----+-----+
| 6703 | Rahul | HYD | 20 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

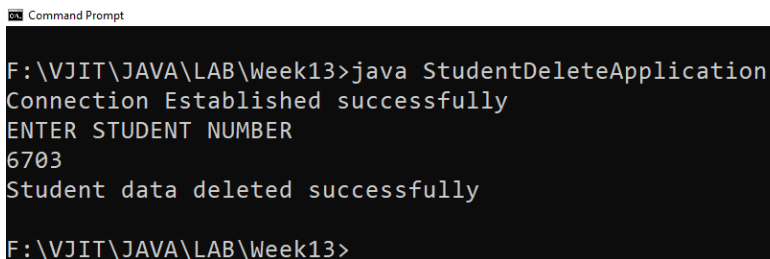
mysql> select * from students;
+-----+-----+-----+-----+
| Roll_No | First_Name | Last_Name | Age |
+-----+-----+-----+-----+
| 6703 | Rahul Dravid | HYD | 20 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```



**4. Java Program to DELETE Record from Database Table in MySql Server**

```
import java.sql.*;
import java.util.*;
class StudentDeleteApplication
{
public static void main (String[]args) throws ClassNotFoundException, SQLException
{
    String url= "jdbc:mysql://localhost:3306/vjit"; // Database details
    String username = "root"; // MySQL credentials
    String password = "vjit";
    Class.forName("com.mysql.cj.jdbc.Driver"); // Driver name
    Connection con = DriverManager.getConnection(url, username, password);
    System.out.println("Connection Established successfully");
        Statement st = con.createStatement();
        Scanner sc = new Scanner(System.in);
    System.out.println("ENTER STUDENT NUMBER");
    int rno = sc.nextInt();

    int c = st.executeUpdate("delete from students where Roll_No =" + rno);
    if (c == 0)
        System.out.println("Student data does not exist");
    else
        System.out.println("Student data deleted successfully");
    st.close();
    con.close();
}
}
```

**Output:**

```
Command Prompt
F:\VJIT\JAVA\LAB\Week13>java StudentDeleteApplication
Connection Established successfully
ENTER STUDENT NUMBER
6703
Student data deleted successfully
F:\VJIT\JAVA\LAB\Week13>
```

MySQL 8.0 Command Line Client

```
mysql> select * from students;
+-----+-----+-----+-----+
| Roll_No | First_Name | Last_Name | Age |
+-----+-----+-----+-----+
| 6703 | Rahul Dravid | HYD | 20 |
| 6706 | Ajay | TPT | 30 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

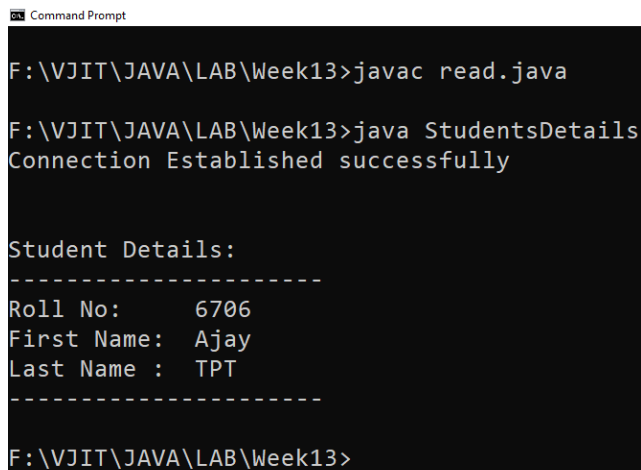
mysql> select * from students;
+-----+-----+-----+-----+
| Roll_No | First_Name | Last_Name | Age |
+-----+-----+-----+-----+
| 6706 | Ajay | TPT | 30 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

### 5. Java Program to **SELECT** Records from Database Table in MySql Server

```
import java.sql.*;
import java.util.*;
class StudentsDetails
{
public static void main (String[]args) throws ClassNotFoundException, SQLException
{
    String url= "jdbc:mysql://localhost:3306/vjit"; // Database details
    String username = "root"; // MySQL credentials
    String password = "vjit";
    Class.forName("com.mysql.cj.jdbc.Driver"); // Driver name
    Connection con = DriverManager.getConnection(url, username, password);
    System.out.println("Connection Established successfully");
    Statement st = con.createStatement();
    System.out.println("\n\nStudent Details: ");
    System.out.println("-----");
    ResultSet rs = st.executeQuery("select * from students");
    while(rs.next())
    {
        System.out.println("Roll No:   " + rs.getInt(1));
        System.out.println("First Name: " + rs.getString(2));
    }
}
```

```
        System.out.println("Last Name : " + rs.getString(3));
        System.out.println("-----");
    }
    rs.close();
    st.close();
    con.close();
}
}
```

### Output:



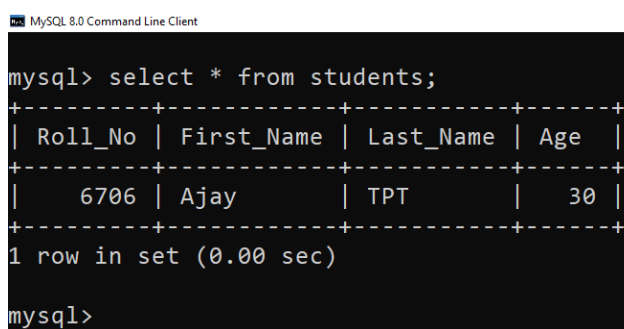
```
Command Prompt

F:\VJIT\JAVA\LAB\Week13>javac read.java

F:\VJIT\JAVA\LAB\Week13>java StudentsDetails
Connection Established successfully

Student Details:
-----
Roll No:      6706
First Name:   Ajay
Last Name :   TPT
-----

F:\VJIT\JAVA\LAB\Week13>
```



```
MySQL 8.0 Command Line Client

mysql> select * from students;
+-----+-----+-----+-----+
| Roll_No | First_Name | Last_Name | Age |
+-----+-----+-----+-----+
| 6706 | Ajay | TPT | 30 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>
```