Object - Oriented Programming 2

Week 8. GUI - Layout Managers, Introduction to Java Database Connectivity(JDBC)

By Elubu Joseph - MSc.IS

Lecturer

Department of Information Technology

Kumi University

Email: josebulinda@gmail.com

jose@kumiuniversity.ac.ug

Agenda

1. GUI - Layout Manager Classes,
2. Introduction to Java Database Connectivity-JDBC

GUI - LayoutManager

Layout Manager is an interface used for arranging the components in order of preference. LayoutManager interface is implemented by all its classes.

Below are some of the classes which are used for the representation of layout manager.

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. javax.swing.BoxLayout

BorderLayout Class

is used when one wants to arrange the components in five regions of he Frame. The five regions can be Top, Bottom, Left, Center and the Right.

Delcaration

public class BorderLayout extends Object implements LayoutManager2, Serializable

There are 5 types of constants in BorderLayout. They are as following:

1. public static final int NORTH
2. public static final int SOUTH
3. public static final int EAST
4. public static final int WEST
5. public static final int CENTER

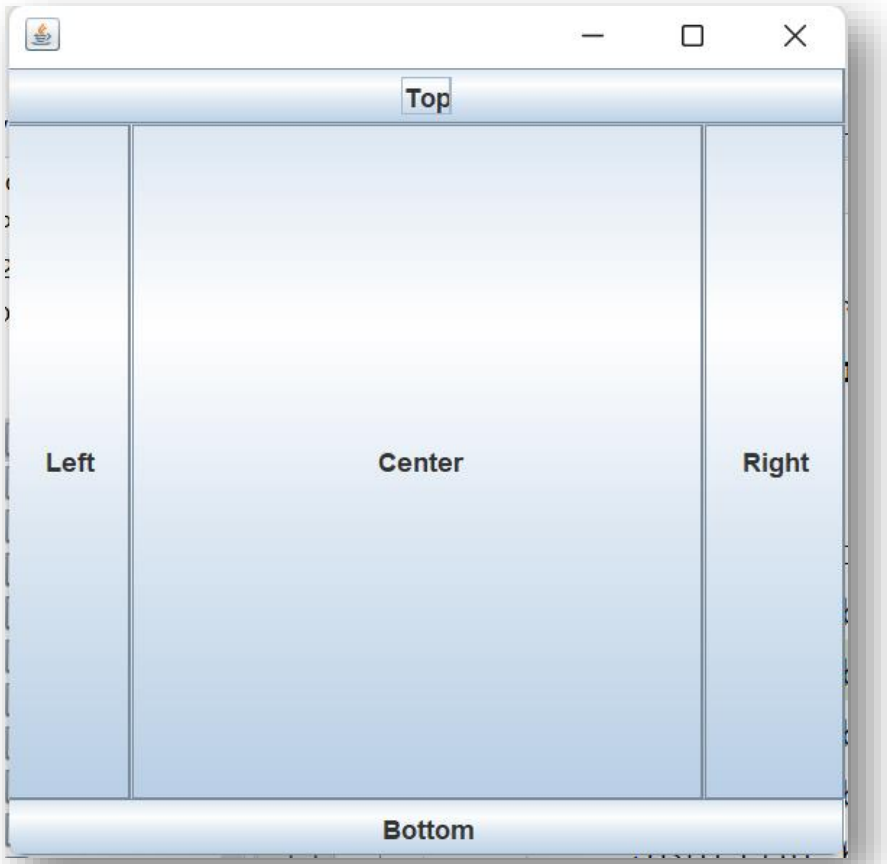
Arranging Items using Border Layout

```
import java.awt.*; import javax.swing.*;
public class BLPro {
        JFrame f;
        BLPro() {
        f=new JFrame();
        JButton b1=new JButton("Top");
        JButton b2=new JButton("Bottom");
        JButton b3=new JButton("Left");
        JButton b4=new JButton("Right");
        JButton b5=new JButton("Center");
        f.add(b1,BorderLayout.NORTH);
        f.add(b2,BorderLayout.SOUTH);
        f.add(b3,BorderLayout.EAST);
        f.add(b4,BorderLayout.WEST);
        f.add(b5,BorderLayout.CENTER);
        f.setSize(400,400);
        f.setVisible(true);
        }
        public static void main(String[] args){
        new BLPro();
        } }
```

In this example, we created five Buttons.

Arranging Items using Border Layout+code and Output

```
2  import java.awt.*; import javax.swing.*;
3  public class BLPro {
4      JFrame f;
5      BLPro(){
6          f=new JFrame();
7          JButton b1=new JButton("Top");
8          JButton b2=new JButton("Bottom");
9          JButton b3=new JButton("Right");
10         JButton b4=new JButton("Left");
11         JButton b5=new JButton("Center");
12         f.add(b1,BorderLayout.NORTH);
13         f.add(b2,BorderLayout.SOUTH);
14         f.add(b3,BorderLayout.EAST);
15         f.add(b4,BorderLayout.WEST);
16         f.add(b5,BorderLayout.CENTER);
17         f.setSize(400,400);
18         f.setVisible(true);
19     }
       public static void main(String[] args){
           new BLPro();
22     } }
```



Output

GridLayout Class

is a class or layout manager that lays out a container's components in a rectangular grid. The container is divided into equal-sized rectangles, and one component is placed in each rectangle.

Declaration

public class GridLayout extends Object implements LayoutManager, Serializable.

There are 3 types of constructor in Grid Layout. They are as following:

1. GridLayout()
2. GridLayout(int rows, int columns)
3. GridLayout(int rows, int columns, inthgap, int vgap)

Example:

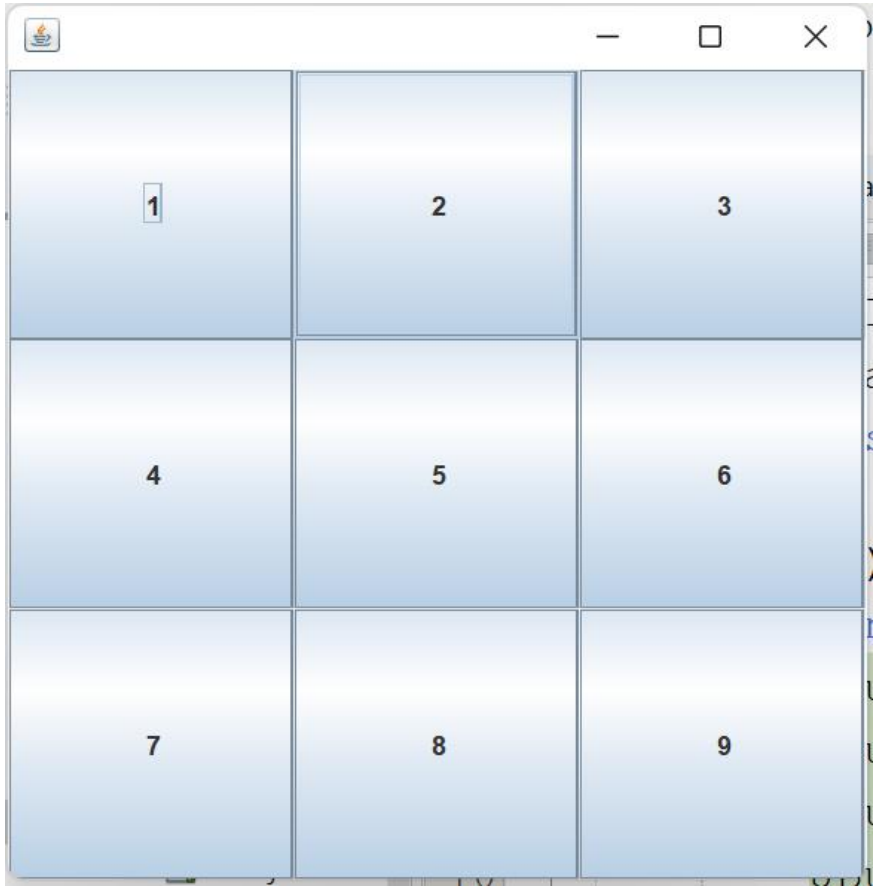
Grid Layout Program

In this example, we created a GridLayout of Three rows and three Columns, hence creating 9 cells

```
import java.awt.*; import javax.swing.*;
public class GLPro extends JFrame{
        JFrame f;
        GLPro() {
        f=new JFrame();
        JButton b1=new JButton("1");
        JButton b2=new JButton("2");
        JButton b3=new JButton("3");
        JButton b4=new JButton("4");
        JButton b5=new JButton("5");
        JButton b6=new JButton("6");
        JButton b7=new JButton("7");
        JButton b8=new JButton("8");
        JButton b9=new JButton("9");
        f.add(b1); f.add(b2); f.add(b3); f.add(b4); f.add(b5);
        f.add(b6); f.add(b7); f.add(b8); f.add(b9);
        f.setLayout(new GridLayout(3,3));
        f.setSize(400,400);
        f.setVisible(true);
        }
        public static void main(String[] args){
        new GLPro();
        } }
```

Grid Layout Program+Code and output

```
1   package GUI;
2   import java.awt.*; import javax.swing.*;
3   public class GLPro extends JFrame{
4       JFrame f;
5       GLPro() {
6           f=new JFrame();
7           JButton b1=new JButton("1"); JButton b2=new JButton("2");
8           JButton b3=new JButton("3"); JButton b4=new JButton("4");
9           JButton b5=new JButton("5"); JButton b6=new JButton("6");
10          JButton b7=new JButton("7"); JButton b8=new JButton("8");
11          JButton b9=new JButton("9");
12          f.add(b1); f.add(b2); f.add(b3); f.add(b4); f.add(b5);
13          f.add(b6); f.add(b7); f.add(b8); f.add(b9);
14          f.setLayout(new GridLayout(3,3));
15          f.setSize(400,400);
16          f.setVisible(true);
17      }
18      public static void main(String[] args){
            new GLPro();
20  } }
```



Output

FlowLayout class

is used, when we want to arrange the components in a sequence, one after another.

Declaration

public class FlowLayout extends Object implements LayoutManager, Serializable

There are 3 types of constructor in the Flow Layout. They are as following:

1. FlowLayout()

2. FlowLayout(int align)

3. FlowLayout(int align, int h, int v)

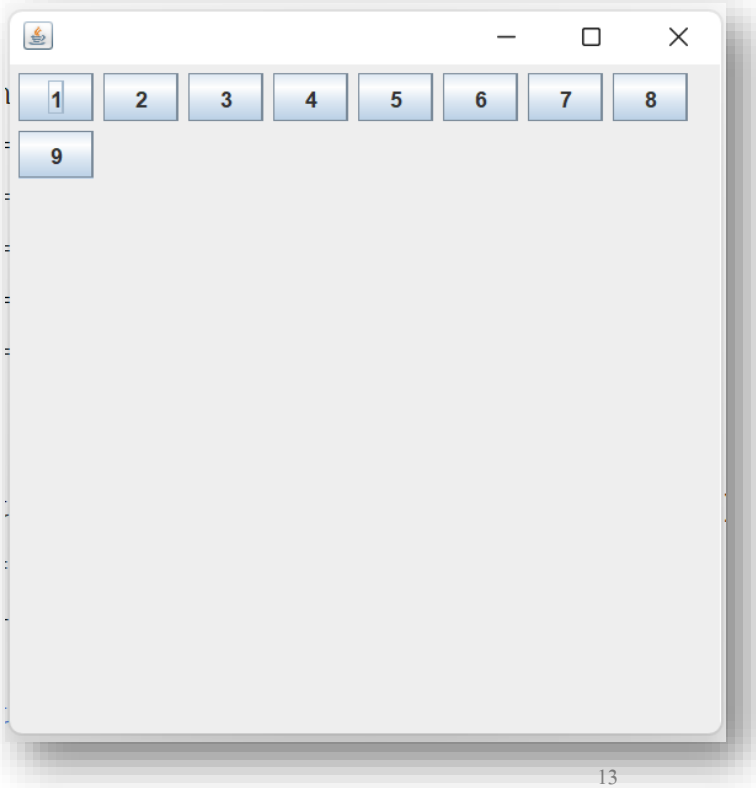
FlowLayout Program

```
import java.awt.*; import javax.swing.*;
public class FLPro{
        JFrame f;
        FLPro() {
        f=new JFrame();
        JButton b1=new JButton("1");
        JButton b2=new JButton("2");
        JButton b3=new JButton("3");
        JButton b4=new JButton("4");
        JButton b5=new JButton("5");
        JButton b6=new JButton("6");
        JButton b7=new JButton("7");
        JButton b8=new JButton("8");
        JButton b9=new JButton("9");
        f.add(b1); f.add(b2); f.add(b3); f.add(b4); f.add(b5);
        f.add(b6); f.add(b7); f.add(b8); f.add(b9);
        f.setLayout(new FlowLayout(FlowLayout.LEFT);
        f.setSize(400,400);
        f.setVisible(true);
        }
        public static void main(String[] args){
        new FLPro();
        } }
```

FlowLayout Class Program Code and Output

```
onets.java
package GUI;
import java.awt.*; import javax.swing.*;
public class FLPro{
        JFrame f;
        FLPro(){
        f=new JFrame();
        JButton b1=new JButton("1"); JButton b2=new JButton("2");
        JButton b3=new JButton("3"); JButton b4=new JButton("4");
        JButton b5=new JButton("5"); JButton b6=new JButton("6");
        JButton b7=new JButton("7"); JButton b8=new JButton("8");
        JButton b9=new JButton("9");
        f.add(b1); f.add(b2); f.add(b3); f.add(b4); f.add(b5);
        f.add(b6); f.add(b7); f.add(b8); f.add(b9);
        f.setLayout(new FlowLayout(FlowLayout.LEFT ));
        f.setSize(400,400);
        f.setVisible(true);
        }
        public static void main(String[] args){
        new FLPro();
        } }
```

Output



BoxLayout Class

Is a layout manager that allows multiple components to be laid out either vertically or horizontally. BoxLayout class is found in the javax.swing package.

Declaration

public class BoxLayout extends Object implements LayoutManager2, Serializable .

BoxLayout has only 1 constructor.

BoxLayout(Container c, int axis) is the only constructor in the Box Layout

Example:

BoxLayout Class Program

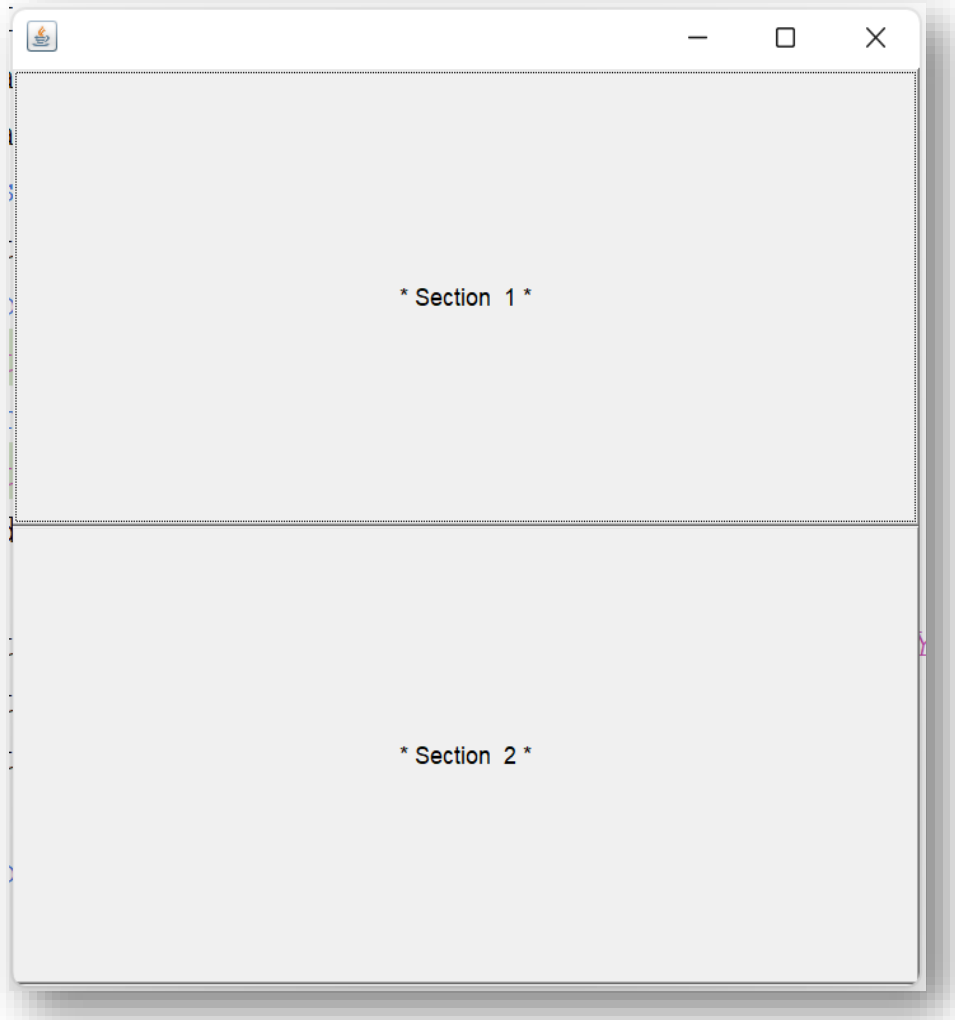
Box Layout is used, when we want to arrange the components vertically or horizontally.

It therefore utilizes two constants that is **Y.ASIS** and **X.AXIS** to determine the Layout of items in the Frame.

```
import java.awt.*;
import javax.swing.*;
public class BLPro1 extends Frame {
        Button buttonBox[];
        public BLPro1() {
        buttonBox = new Button [2];
        for (int i = 0;i<2;i++){
        buttonBox[i] = new Button ("* Section " + (i + 1)+" *");
        add (buttonBox[i]);
        }
        setLayout (new BoxLayout(this, BoxLayout.Y_AXIS));
        setSize(490,490);
        setVisible(true);
        }
        public static void main(String args[]) {
                BLPro1 obj=new BLPro1();
        }
}
```

BoxLayout Program Code and Output

```
1  package GUI;
2  import java.awt.*;
3  import javax.swing.*;
4  public class BLPro1 extends Frame {
5          Button buttonBox[];
6          public BLPro1() {
7          buttonBox = new Button [2];
8          for (int i = 0;i<2;i++){
9          buttonBox[i] = new Button ("* Section  " + (i + 1)+" *");
10         add (buttonBox[i]);
11         }
12         setLayout (new BoxLayout (this, BoxLayout.Y_AXIS));
13         setSize(490,490);
14         setVisible(true);
15         }
16         public static void main(String args[]) {
               BLPro1 obj=new BLPro1();
18         }
19 }
```



CardLayout Class

is a layout manager class for a container that belongs to **java.awt** package. It treats each component in the container as a card. Only one card is visible at a time, and the container acts as a stack of cards. The first component added to a CardLayout object is the visible component when the container is first displayed.

The ordering of cards is determined by the container's own internal ordering of its component objects.

Declaration

public class CardLayout extends Object implements LayoutManager2, Serializable

There are 2 types of constructor in the Card Layout. They are as following:

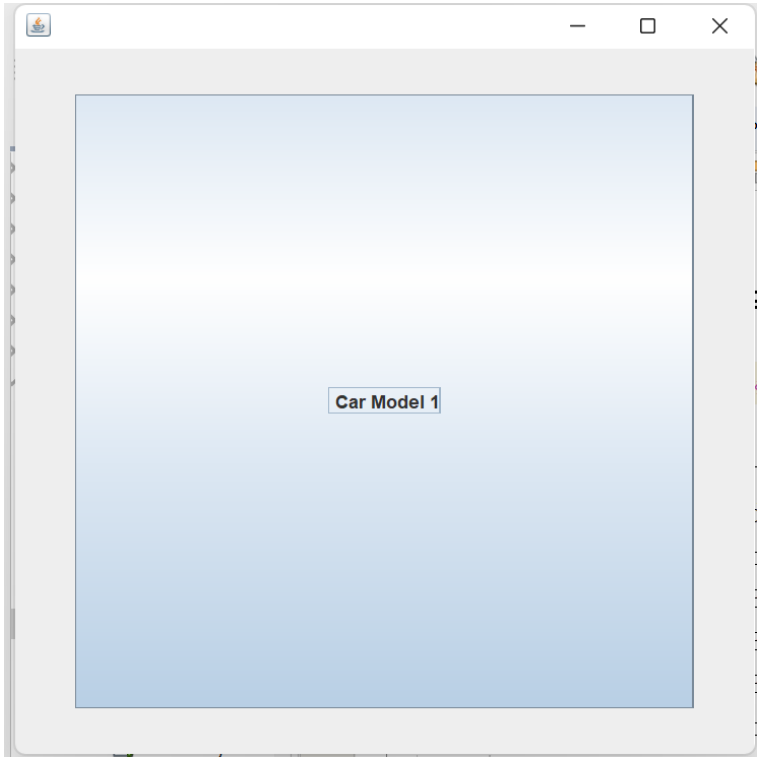
1. CardLayout()
2. CardLayout(inthgap, intvgap)

Card Layout Program

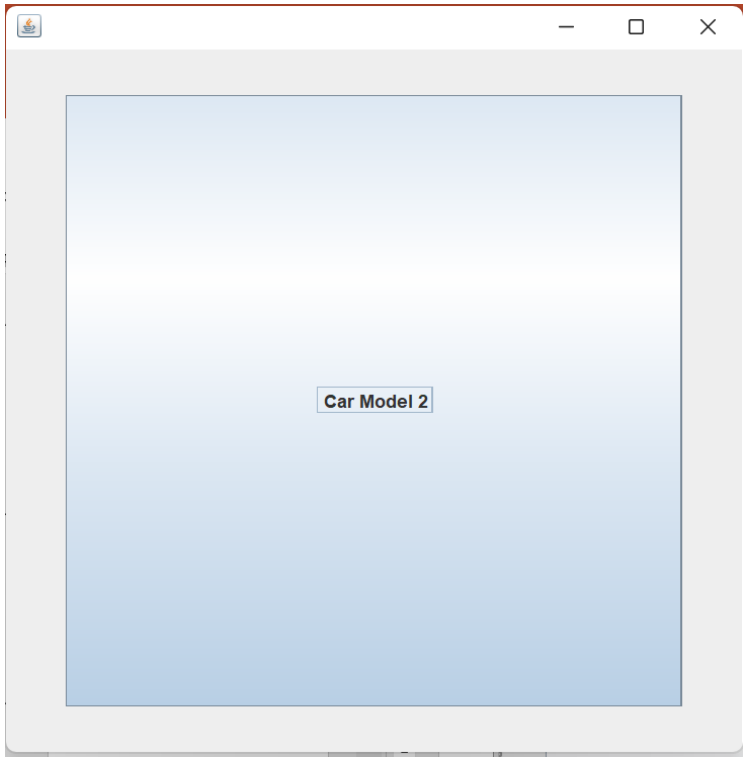
In this example, we used ActionListener interface to to listen to Mouse click to alternate between Car Model 1 to Car Model 2 being housed by the container.

```
import java.awt.*; import java.awt.event.*; import javax.swing.*;
public class CardLPro extends JFrame implements ActionListener {
        CardLayout cl;
        JButton box1,box2,box3; Container d;
        CardLPro() {
        d=getContentPane();
        cl=new CardLayout(40,30);
        d.setLayout(cl);
        box1=new JButton(" Car Model 1");
        box2=new JButton(" Car Model 2 ");
        box3=new JButton(" Car Model 3");
        box1.addActionListener(this);
        box2.addActionListener(this);
        box3.addActionListener(this);
        d.add("P",box1); d.add("Q",box2); d.add("R",box3);
        }
        public void actionPerformed(ActionEvent e) {
        cl.next(d);
        }
        public static void main(String[] args) {
        CardLPro obj =new CardLPro(); obj.setSize(500,500);
        obj.setVisible(true);
        obj.setDefaultCloseOperation(EXIT_ON_CLOSE);
} }
```

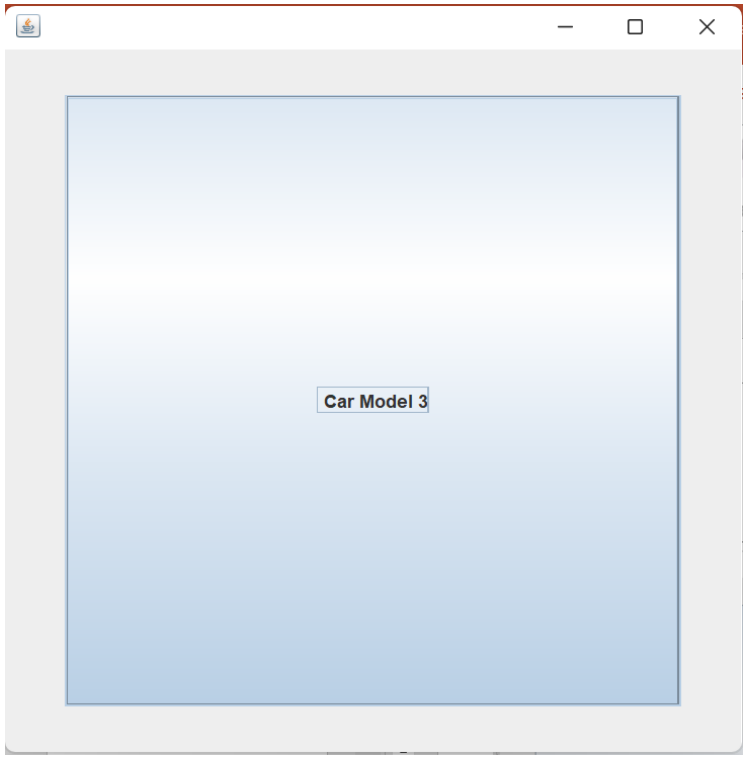
Card Layout Program output



Default Display



Change to Model 2 on click first click



Change to Model 3 on click second click

Introduction to JDBC

Introduction to JDBC

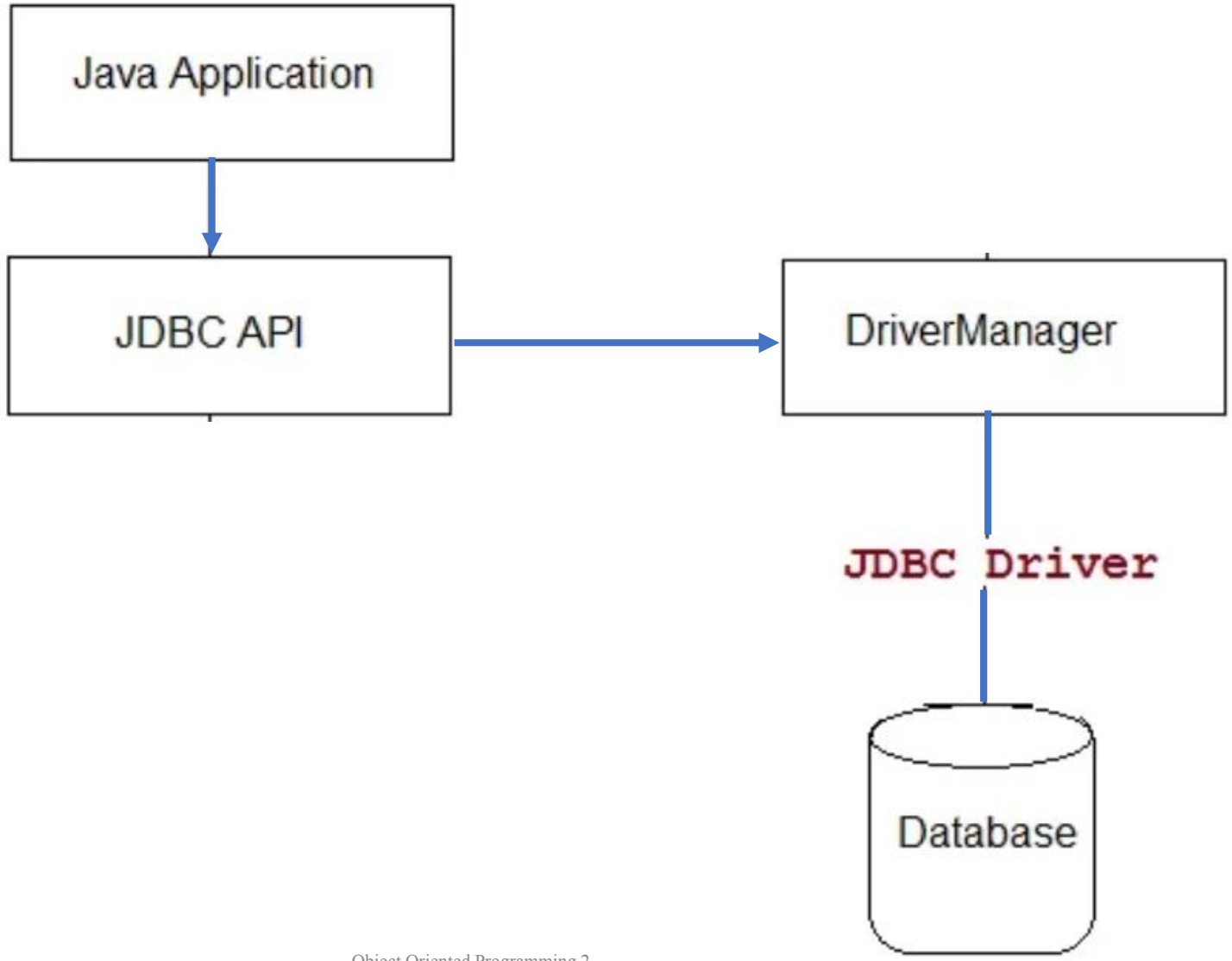
Java Database Connectivity(JDBC) is an **Application Programming Interface(API)** used to connect Java application with Database. JDBC is used to interact with various type of Database such as Oracle, MS Access, MySQL and SQL Server.

JDBC can also be defined as the platform-independent interface between a relational database and Java programming. It allows java program to execute SQL statement and retrieve result from database.

The JDBC API consists of classes and methods that are used to perform various operations like: **connect**, **read**, **write** and **store** data in the database. In this tutorial we will learn the JDBC with examples.

You can get idea of how JDBC connect Java Application to the database by following image.

How Java Application connect to the database



Versions of JDBC

There quite a number of version of Java Database Connectivity APIs. Below are some;

JDBC 4.3

JDBC 4.2

JDBC 4.1

JDBC 4.0

Etc.

Sun Microsystems released JDBC as part of Java Development Kit (JDK) 1.1 on February 19, 1997. Since then it has been part of the Java Platform, Standard Edition (Java SE).

JDBC 4.0

Java introduced **JDBC 4.0**, wtiht advance specification of JDBC. It provides the following advance features.

1. Connection Management
2. Auto loading of Driver Interface.
3. Better exception handling
4. Support for large object
5. Annotation in SQL query.

JDBC 4.1

The JDBC 4.1 version was introduced with Java SE 7 and includes the following features:

1. Allows to use a **try-with-resources** statement to automatically close resources of type **Connection, ResultSet** etc.
2. Introduced the **RowSetFactory** interface and the **RowSetProvider** class to create all types of row sets supported by your JDBC driver.

JDBC 4.2

version is introduced with Java SE 8 and includes the following features:

1. The REF_CURSOR support.- A ref cursor is a variable, defined as a cursor type, which will point to, or reference a cursor result(Ref cursor examples, n.d).
2. Added an interface **java.sql.DriverAction**
3. Added an interface **java.sql.SQLType**
4. Added an Enum **java.sql.JDBCType**
5. Add Support for large update counts
6. Improved the existing interfaces: **Driver, DriverManage, DatabaseMetaData**.
7. Interfaces and classes enhanced for RowSet1.2

JDBC 4.3

JDBC 4.3 is the latest version of JDBC release by Oracle corporation on September 21, 2017 included in Java SE 9.

JDBC 4.3

1. Added support to Statement for enquoting literals and simple identifiers
2. Added Sharding support. Sharding is a type of database partitioning that separates large databases into smaller, faster, more easily managed parts,(Awati, R., & Denman, J. 2022).
3. Enhanced Connection to be able to provide hints to the driver that request an independent unit of work
4. Enhanced DatabaseMetaData to determine if Sharding is supported
5. Added the method drivers to DriverManager to return a Stream of the currently loaded and available JDBC drivers

JDBC Driver

JDBC Driver is required to establish connection between application and database. It also helps to process SQL requests and generating result. The following are the different types of driver available in JDBC which are used by the application based on the scenario and type of application.

1. Type-1 Driver or JDBC-ODBC(Open Database Connectivity) bridge
2. Type-2 Driver or Native API Partly Java Driver
3. Type-3 Driver or Network Protocol Driver
4. Type-4 Driver or Thin Driver

1 Type-1 Driver- JDBC-ODBC bridge

act as a bridge between JDBC and other database connectivity mechanism(ODBC). This driver converts JDBC calls into ODBC calls and redirects the request to the **Open Database Connectivity -**ODBC driver.

Open Database Connectivity (ODBC) interface is a C programming language interface that makes it possible for applications to access data from a variety of database management systems,(David-Engel, n.d)

1 Type-1 Driver- JDBC-ODBC bridge +

Advantage

1. Easy to use
2. Allow easy connectivity to all database supported by the ODBC Driver.

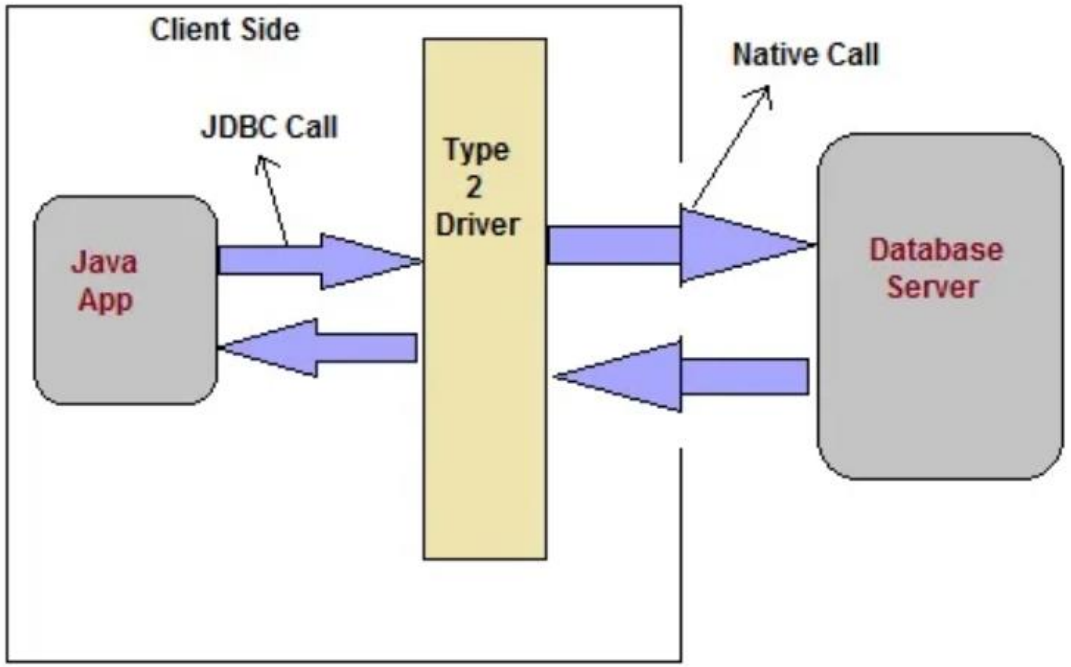
Disadvantage

1. Slow execution time
2. Dependent on ODBC Driver.
3. Uses Java Native Interface(JNI) to make ODBC call.

Note: the JDBC-ODBC Bridge has been removed In Java 8

2. Type-2 Driver Native API Driver

This type of driver make use of Java Native Interface(JNI) call on database specific native client API. These native client API are usually written in C and C++.



(Studytonight.com)

2. Type-2 Driver Native API Driver+

Advantage

1. faster as compared to **Type-1 Driver**
2. Contains additional features.

Disadvantage

1. Requires native library
2. Increased cost of Application

Type-3 Driver Network Protocol Driver

This driver translate the JDBC calls into a database server independent and Middleware server-specific calls. Middleware server further translate JDBC calls into database specific calls.

Advantages

1. Does not require any native library to be installed.
2. Database Independency.
3. Provide facility to switch over from one database to another database.

Disadvantage

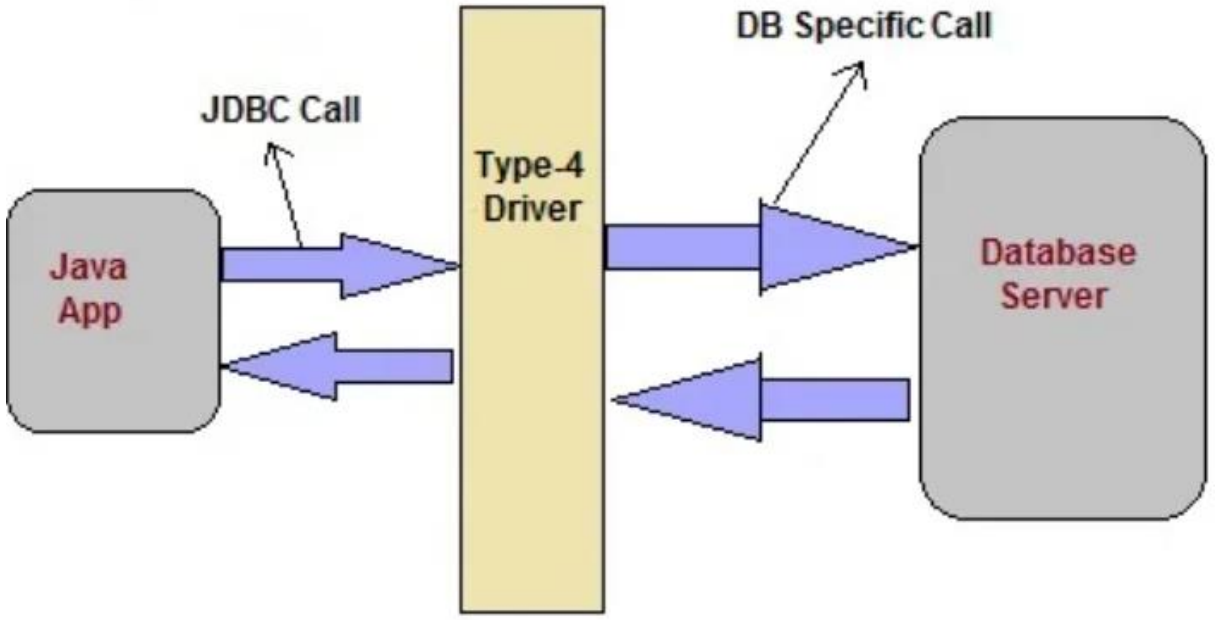
1. Slow due to increase number of network call.

Type-4 Driver Thin Driver

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

Also, it is a platform-independent driver but it is database dependent as it uses a native protocol(Protocol can establish a connection between particular server only).

It does not require any native database library, that is why it is also known as Thin Driver.



(Studytonight.com)

Type-4 Driver Thin Driver+

Advantage

1. Does not require any native library.
2. Does not require any Middleware server.
3. Better Performance than other driver.

Disadvantage

1. Slow due to increase number of network call.

DriverManager class

The DriverManager class is the component of JDBC API and also a member of the java.sql package

In Java, the DriverManager class is an interface between the User and the Driver. This class is used to have a watch on driver which is being used for establishment of the connection between a database and a driver.

The DriverManager class has a list of Driver class which are registered and are called as **DriverManager.registerDriver().**

DriverManager class Methods

S.No.	Method	Description
1	public static void **registerDriver**(Driver driver)	It is used for Registering the Driver with the Driver Manager.
2	public static void **deregisterDriver**(Driver driver)	It is used for Deregistering the Driver with the Driver Manager.
3	public static Connection **getConnection**(String Url)	It is used for establishing a connection with the given URL.
4	public static Connection **getConnection**(String Url, String username, String password)	It is used for establishing the connection with the given URL, username and password.

Connection interface

is used for creating the session between the application and the database. This interface contains Statement, PreparedStatement and DatabaseMetaData. The connection objects are used in Statement and the **DatabaseMetaData.**

commit(), rollback() etc.. are some of the methods of Connection Interface.

Connection interface Methods

S.No.	Method	Description
1	public Statement **createStatement**()	It is used for creating an object of statement for executing the SQL queries.
2	public Statement **createStatement**(int resultSetType, int resultSetConcurrency)	It is used for creating objects for the ResultSet from the given type and concurrency.
3	public void **setAutoCommit**(boolean status)	It is used for setting the commit status. By default, it is always true.
4	public void **commit**()	It is used to save the changes which have been commit or rollback permanent
5	public void **rollback**()	It is used to delete the changes which have been commit or rollback permanent
6	public void **close**()	It is used to delete the changes which have been commit or rollback permanent

Statement interface

is used for executing queries using the database. This interface is a factory of **ResultSet**. It is used to get the Object of **ResultSet**. Methods of this interface is given below.

Statement interface Methods

S.No.	Method	Description
1	public **ResultSetexecuteQuery**(String sql)	It is used for executing the SELECT query
2	public int **executeUpdate**(String sql)	It is used for executing any specified query
3	public boolean **execute**(String sql)	It is used when multiple results are required.
4	public int[] **executeBatch**()	It is used for executing the batch of commands.

ResultSet interface

is used for maintaining the pointer to a row of a table. In starting the pointer is before the first row.

The object can be moved forward as well as backward direction using TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE in **createStatement**(int va,int val). Methods of this interface is given below.

ResultSet interface Methods

S.No.	Method	Description
1	public boolean **next**()	It is used for moving the cursor to the next position from the current position.
2	public boolean **previous**()	It is used for moving the cursor to the previous position from the current position.
3	public boolean **first**()	It is used for moving the cursor to the first position from the current position.
4	public boolean **last**()	It is used for moving the cursor to the Last position from the current position.
5	public boolean **absolute**(int row)	It is used for moving the cursor to the specified position from the current position.
6	public boolean **relative**(int row)	It is used for moving the cursor to the relative row number from the current position.
7	public int **getInt**(int columnIndex)	It is used to get the data from the specified position.

ResultSet interface Methods

S.No.	Method	Description
8	public **int getInt**(String columnName)	It is used to get the data from the specified column name of the current row.
9	public **String getString**(int columnIndex)	It is used to get the data from the specified column name of the current row in form of an integer.
10	public **String getString**(String columnIndex)	It is used to get the data from the specified column name of the current row in form of string.

PreparedStatement interface

In Java, The PreparedStatement interface is a sub interface of Statement. It is mainly used for the parameterized queries.

A question mark (?) is passed for the values. The values to this question marks will be set by the PreparedStatement. Methods of this interface is given below.

PreparedStatement interface Methods

S.No.	Method	Description
1	public void **setInt**(int paramIndex, int value)	It is used for setting the integer value for the given parameter index.
2	public void **setString**(int paramIndex, String value)	It is used for setting the String value for the given parameter index.
3	public void **setFloat**(int paramIndex, float value)	It is used for setting the Float value for the given parameter index.
4	public void **setDouble**(int paramIndex, double value)	It is used for setting the Double value for the given parameter index.
5	public **int executeUpdate**()	It is used for executing a query.
6	public **ResultSetexecuteQuery**()	It is used for executing the select query.

ResultSetMetaData Interface

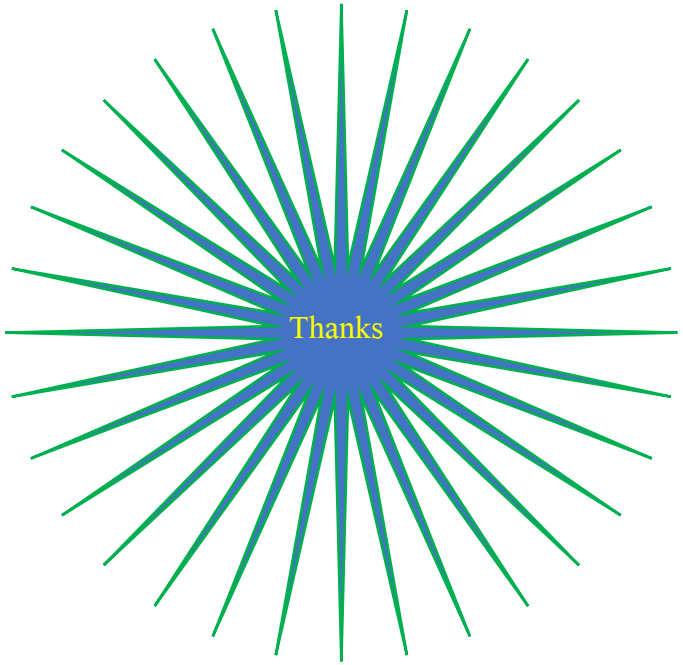
The **ResultSetMetaData** interface is used to get metadata from the **ResultSet** object. Metadata are the data about data. Methods of this interface is given below.

S.No.	Method	Description
1	public int **getColumnCount**()throws SQLException	It is used to get the total number of columns.
2	public String **getColumnName**(int index)throws SQLException	It is used to get the name of the column of a specified column index.
3	public String **getColumnTypeName**(int index)throws SQLException	It is used to get the name of the column of a specified index.
4	public String **getTableName**(int index)throws SQLException	It is used to get the name of a table from the specified column index

Summary

1. GUI - Layout Managers- definitions, Looked at various types of LayoutManager interface classes including; BorderLayout, FlowLayout, GridLayout, CardLayout and BoxLayout all with examples
2. Introduction to Java Database Connectivity-JDBC; saw a number of things including versions of JDBC, how the Java Application connect to the database, looked at DriverManager explaining various drive types DriverManager Classes and interfaces.

Thank you for Listening



References

Awati, R., & Denman, J. (2022, January 12). *What is Sharding?* SearchOracle. Retrieved October 30, 2022, from https://www.techtarget.com/searchoracle/definition/sharding

David-Engel. (n.d.). *Microsoft Open Database Connectivity (ODBC) - open database connectivity (ODBC)*. Open Database Connectivity (ODBC) | Microsoft Learn. Retrieved October 24, 2022, from https://learn.microsoft.com/en-us/sql/odbc/microsoft-open-database-connectivity-odbc?view=sql-server-ver16

Wikimedia Foundation. (2022, September 7). *Java database connectivity*. Wikipedia. Retrieved October 23, 2022, from https://en.wikipedia.org/wiki/Java_Database_Connectivity

Ref cursor examples. (n.d.). Retrieved October 27, 2022, from http://www.dba-oracle.com/t_ref_cursor_examples.htm

David-Engel. (n.d.). *Microsoft Open Database Connectivity (ODBC) - open database connectivity (ODBC)*. Open Database Connectivity (ODBC) | Microsoft Learn. Retrieved October 24, 2022, from https://learn.microsoft.com/en-us/sql/odbc/microsoft-open-database-connectivity-odbc?view=sql-server-ver16