BLOCK 4
GRAPHICAL USER INTERFACE AND JAVA DATABASE CONNECTIVITY

BLOCK 4 INTRODUCTION

To develop any application knowledge of good interface design and its implementation is required. A good user interface not only takes care of user convenience but also makes an application attractive. In Java programming there are APIs such as AWT, Sing, Java FX which are used for UI design.

Handling applications data is one of the key requirements of software development. You can use the JDBC API to access data stored in various relational databases. JDBC API helps you to save, update, delete, and fetch the data from the database. This block is on Graphical User Interface (GUI) in Java and JDBC programming.

This block consists of 4 units and is organised as follows:

Unit 13: This unit explains how to incorporate GUI in Java programming. You will learn AWT, Swing, and JavaFX APIs for GUI. Also, this unit explains use of layouts and adding of HTML content, text and text effects in JavaFX.

Unit 14: This unit explains basics of CSS and use of CSS in JavaFX. How to build UI with FXML is also explained in this unit. Also, event handling in JavaFX along with use of effects, animation and media in JavaFx are explained in this unit.

Unit 15: This unit explain JDBC API one of the most used APIs of Java. Different types of JDBC drives and JDBC database connection steps are explained in this unit. Also, this unit explains use of JDBC driver manager class and JDBC connection interface.

Unit 16: This unit explains use of ResultSet interface in JDBC programming. JDBC transactions, use of commit and rollback, transaction isolation and save points are also explained in this unit. Finally, concept of JDBC batch processing and JDBC RowSet interface are explained.

UNIT 13 INTRODUCTION TO GUI IN JAVA

Structure **Page No.**

13.0 Introduction 343

13.1 Objectives 344

13.2 Introduction to AWT, Swing andJavaFX 344

13.2.1 Abstract Window Toolkit (AWT) 344

13.2.2 Swing 347

13.2.3. JavaFX 349

13.3 Features of JavaFX 353

13.4 User Interface Components of JavaFX 354

13.5 Work with Layouts 357

13.6 Add HTML Content 364

13.7 Add Text and Text Effects in JavaFX 367

13.8 Summary 370

13.9 Solutions/ Answer to Check Your Progress 370

13.10 References/Further Reading 372

13.0 INTRODUCTION

Graphical User Interface (GUI) is a kind of interface through which users can interact with computer applications. Be it web applications, mobile apps, or other commonly used applications like online banking or the online railway reservation system, we look for ease of interaction by using proper buttons, menus, and other controls. These interfaces use proper GUI components and controls. Earlier engineers or programmers used to interact with the computer through the DOS prompt/interface using command line. This interface also used to be called the character user interface (CUI),which was very inconvenient for non-programming users. With the development of the Graphical user interface, the use of computing devices by non-technical users or ordinary users has been more convenient because GUI provides an interface that allows users to interact with electronic devices through graphical objects and controls to convey information and represent action taken by the users.

Javaprovides a very rich set of Graphical User interface (GUI) API for developing GUI programs. In fact, GUI is one of the most helpful features provided by Java Development Kit (JDK). Java provides various essential classes and libraries for the implementation of graphic classes to the programmers for constructing their own Graphical User Interface applications. The Graphic classes developed by the JDK developer team are highly complex and uses many advanced design patterns. However, for the programmers, the reuse of these classes and methods are very easy to use. Some commonly used APIs for Graphic User interface (GUI) programmingare : Applet, AWT, Swing, JavaFX. In this unit, you will learn about the basics of AWT, Swing and JavaFX API and its implementation with examples.

JavaFX is an open-source programming platform for developing next-generation client applications to implement mobile apps, desktop systems and embedded systems built using Java. It is designed using the Model View Controller (MVC) design pattern to keep the code that handles an application's data, separate from the

User Interface Code. To developenterprise applications, generally, MVC design pattern is used, because it does not mix the UI code with the application logic code.The controller is the mediator between UI and Business logic (the data). Working with JavaFX, the model corresponds to an application's data model, the view is FXML. The FXML is XML-based language designed to create the user interface for JavaFX applications.The controller is the code that determines the action when the user interacts with the UI. Mainly the controller handles all events in the application. The First LTS (long term support) version of JavaFX is the JavaFX 11 which was released by Gluon. It is recommended to use the current LTS version for applications development. In this unit JavaFX 11 is used for the demonstration of the examples.

13.1 OBJECTIVES

After going through this unit, you will be able to:

- explain GUI and different types of GUI API available in Java,
- apply the concept of JavaFX in programming,
- differentiate between JavaFX other GUI's APIs,
- describe important features of JavaFX,
- use components & layouts in JavaFX,and
- write a program using JavaFX.

13.2 INTRODUCTION TO AWT, SWING AND JAVAFX

Java provides three main sets of Java APIs for Graphic User interface (GUI) programming languages

- AWT (Abstract Windowing Toolkit)
- Swing
- JavaFX

Let us seethese Java APIs one by one, which are used for Graphical User Interface (GUI) programming in the Java family.

13.2.1 Abstract Window Toolkit (AWT)

AWTis an API that is used for developing window-based applications in Java. AWT was introduced with JDK 1.0, and it is used to create GUI objects liketextboxes, labels, checkbox, buttons, scroll bars, and windows etc. AWT is part of the Java Foundation Classes (JFC) from Sun Microsystems, creator of Java programming language. The JFC containsa set of graphs libraries and classesused to develop the user interface of the Windows-based application program.AWT is one of Java's largest packages (Java.awt). Its hierarchy is logically organized in a top-down fashion. The hierarchy of the Java AWT package/class is shown in figure 13.1.

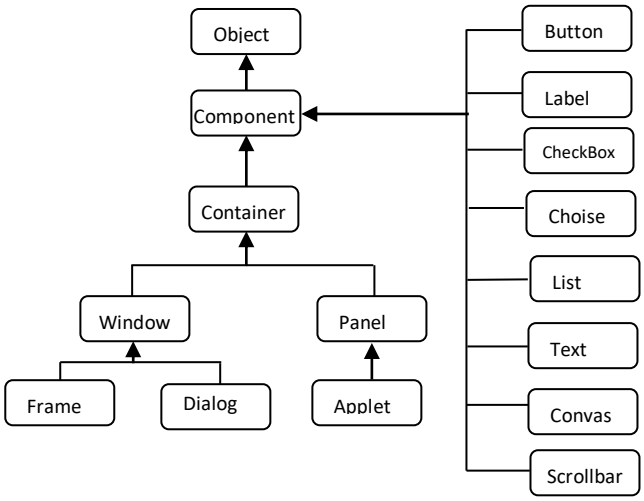

Figure 13.1: Hierarchy of Java AWT classes

Component Class

As we can see in figure 1, the Component class ison the top of the AWT hierarchy. It is an abstract class that encapsulates all of the attributes of a visual component. It is platform-dependent i.e., its components are displayed according to the view of the operating system on which it runs. It is also heavyweight because its components uses the OS's resources during the execution of the program. User interface elements (i.e., textbox, label, list, checkbox, etc.) which are displayed on the screen and used to interact with the user are subclasses (such as Label, TextField, TextArea, RadioButton, CheckBox, List, Choice etc.) of Component class.

Container Class

The container is a subclass of Component class of AWT. It provides theadditional methods that allow other Component objects to call within the container object. Other Container objects can be stored inside a container because they are themselves instances of the Component class. This makes it a multilevelled containment system. A container is responsible for laying out or positioning the component's object using the various layout managers.

Panel Class

The panel class is the subclass of the container class that implements the container class only. It does not provide any special new methods. It provides space to assemble all components, including other panels. Other components can be added within a Panel object by calling its add() method (which is inherited from Container). Once these components are assembled or added within the panel, you can set their position and resize them manually using the setLocation(), setSize(), setPreferredSize(), or setBounds() methods which are defined by the Component class.

Windows Class

The Window is an area that is displayed on the screen when you execute an AWT program. It creates a top-level window that is not contained with any other object;it directly sits on the desktop screen. It provides a multitasking environment for users to interact with the system through the component shown on the screen. It must have a frame, dialog or another window defined by the programmer/owner when it is constructed in the program because window objects cannot be created directly.

Frame Class

The Frame is the subclass of Window. It is the top-level of windows that provides title bar, menu bar, borders and resizing option forwindow.It can also have other

components like button, text field, scrollbar etc.It encapsulates the window and uses BorderLayout as the default layout manager.Frame is one of the most widely used containers for AWT applications.

Now let us see use of different awt components in a program.

Example 1: Write a First programme using AWT in Java

```
package org.ignou.gui;

import Java.awt.*;
import Java.awt.event.WindowAdapter;
import Java.awt.event.WindowEvent;

public class AWTFirstExampleextends WindowAdapter
{
  Frame myFrame;

  AWTFirstExample()
  {
    //Creating a frame
    myFrame= new Frame();

    //Add Windows Lister
    myFrame.addWindowListener(this);
    myFrame.setTitle("AWT Example");

    //Creating a label
    Label myLabel= new Label ("Welcome to First AWT Programme.");

    //adding label to the frame
    myFrame.add(myLabel);

    //setting frame size.
    myFrame.setSize(400, 400);

    //set frame visibility true
    myFrame.setVisible(true);
  }
  //setting Window close operation.
  public void windowClosing(WindowEvente)
  {
    myFrame.dispose();
  }

  public static void main(String args[])
  {
    new AWTFirstExample();
  }
}
```

Output:

Figure 13.2: Output screen of Example 1

13.2.2 Swing

In section 13.2.1 welearned about AWT in Java, the first GUI API provided by Java programming language. Swing is another API for developing Windows based applications in Java.Due to the limitation of AWT (components use native code resources, heavyweight, platform-dependent, etc.), Swing was introduced in the year of 1997 which is a platform-independent "model–view–controller" GUI framework for Java.The components of swing are written in Java which is a part of Java Foundation Classes (JFC). Swing API in Java not onlyprovides platform independence but also are lightweight components. JFC consists of Swing, Java2D, Accessibility, Internationalization and pluggable look and feel support API. The JFC has been integrated into code Java since JDK1.2.

As mentioned above, Swing is developed on top of the AWT, but it has many differences. Some significant differences are given in table 13.1.

Table 13.1: Major Differences between AWT and Swing

S.No.	JavaSwing	JavaAWT
1	It is platform-independent API.	It is platform-dependent API
2	It has lightweight GUI components.	It has heavyweight GUI components.
3	It also supports pluggable look and feel of GUI.	It doesnot support pluggable look and feel.
4	Itprovides more advanced components than AWTe.g.tables, lists, scrollpanes, colorchooser, tabbedpane etc.	Itprovides fewer components than Swing
5	It supports the MVC design pattern	It does not followthe MVC design pattern.
6.	Execution is faster	Execution is slower
7.	The components of Swing do not require much memory space	The components of AWT require more memory space

The hierarchy of the Swing package/class in Javais given in figure 13.3.

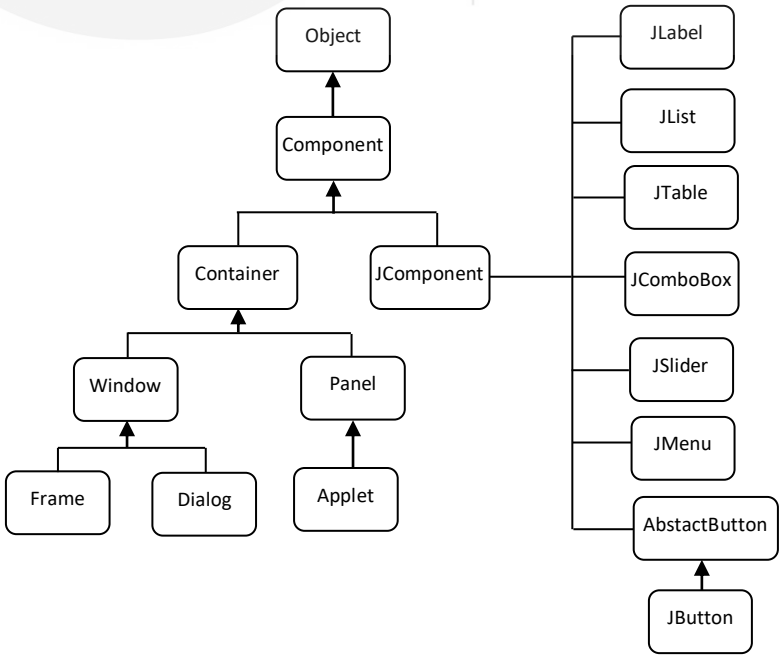

Figure 13.3: Hierarchy of Swing Package/Class

As explained above, Swing is developed on top of the AWT which overcomes the limitations of AWT. Two major features of the Swing Components are:

- Lightweight
- Swing Supports a Pluggable Look and Feel

These features provides an effective and easy-to-use solution to the problems of GUI development in Java. Both the features of Swing are explained below:

Swing Components Are Lightweight: As we know the Swing has been written in Javawhich make it platform-independent and does not map directly to platform-specific peers. Another reason that it is lightweight because its components are rendered using graphics primitives; components can be transparent, which enables non rectangular shapes. That is why the lightweight components are flexible and more efficient. It determines the look and feel of each component; therefore, lightweight components do not translate into native peers. This means that each component of the Swing will work in a consistent manner across all the platforms.

The look and feel of a component iscontrolled by Swing. It supports Pluggable Look and Feel because each of the component is rendered by Java code in place of native peers. It means that it is possible to separate the look and feel of a component from the logic of the component in Swing. By separating the look and feel, it provides a significant advantage**.** It makes it possible to "plug in" a new look and feel for any given component without having any adverse effects on the code that uses it. Hence, it becomes possible to define entire sets of look-and-feels that represent different GUI styles. To usea specific style, programmer need to simply apply "plug-in" in the design.Once this is done, all components are automatically rendered using that style. For example, if you are sure that an application is going to run only in a Windows environment, it is possible to specify the Windows look andfeel. Also, through proper programming, the look and feel can be changed dynamically at run time. But we are not going to discuss that in this course. Now let us see the example program developed usingSwing in Java

Example 2: Use of Swing components

```
package org.ignou.gui;
import Javax.swing.*;
import Java.awt.*;
public class MySwingExampleextends JFrame
{
  public MySwingExample ()
  {
   //setting title of frame as My Window
   setTitle("Swing Example");

   //Creating a label named Welcome to My Second Window
   JLabel jLabel1= new JLabelWelcome to First Swing Programme.");

   //adding label to frame.
   add(jLabel1);

   //setting layout using Flow Layout object.
   setLayout(new FlowLayout());

   //setting Window close operation.
   setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

   //setting size
   setSize(400, 400);
```

```
   //setting frame visibility
   setVisible(true);
  }
  public static void main (String[] args)
  {
    new MySwingExample ();
  }
}
```

Output:

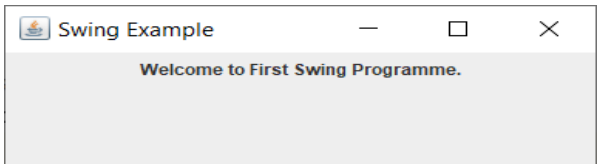

Figure 13.4: Output screen of above example 2

13.2.3 JavaFX

JavaFX is the most popular set of APIs for GUI development using Java, which is used to develop Windows-based applications and Rich Internet Applications (RIA)that can run through a wide variety of devices. JavaFX is the next-generation open-source client application platform for implementing mobile, desktop, and embedded systems, which is built on the Java platform. Firstly, it was introduced into JDK1.8, which intended to replace Swing in Java. An application builton JavaFX can run on multiple platforms, including Web, Mobile and Desktops. In this unit we will deep dive into all essential aspects of JavaFX.

Java provides its GUIAPIs (e.g.,AWT/Swing/JavaFX graphic) with JDK. Also, apart from this, there are some other GUI tools that work with Java easily, such as Google Web Toolkit (GWT), which Google provides, and Standard Widget Toolkit (SWT) provided by Eclipse.

This unit mainly focuses on a practical base session on the latest GUIs APIs called JavaFX. Prerequisite software/tools/libraries to a setup development environment for JavaFX:

Minimum prerequisite software/tools/libraries	Version of software/tools/libraries used during this unit writing(for running JavaFX programs)
JDK 8 and above	JDK 11
Any IDE (Eclipse/NetBeans/IntelliJ Idea, etc.)	IntelliJ Idea
Latest JavaFX Libraries	JavaFX 11 LTS

Now let us create the first Programme using JavaFX

1. Create a new JavaFX Project using IntelliJIDEA
    1. Open IntelliJIDEA which is preinstalled in your PC, if not instlled then install it first.
    2. There are two way to create project, first Click on + New Project from welcome screen as shown in figure 13.5. Second Click to File menu → New →Project…
    3. Popup window of New Project will display where you have to select project type and SDK as shown in figure 13.6.
    4. Enter the Project Name(JavaFXFirstProgramme), Location, Language, select SDK, Change Group and Artifects as per your requirment then click Next button to .

Figure 13.5: Create New Project in IntelliJ IDEA

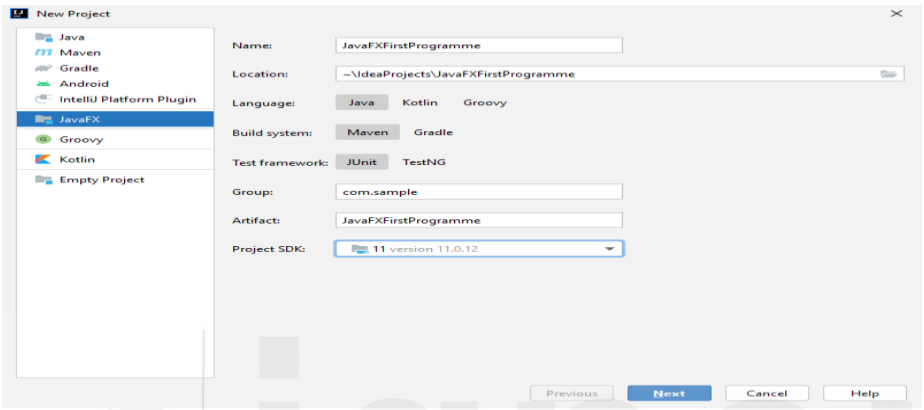

Figure 13.6: Creating JavaFX Application in IntelliJ IDEA

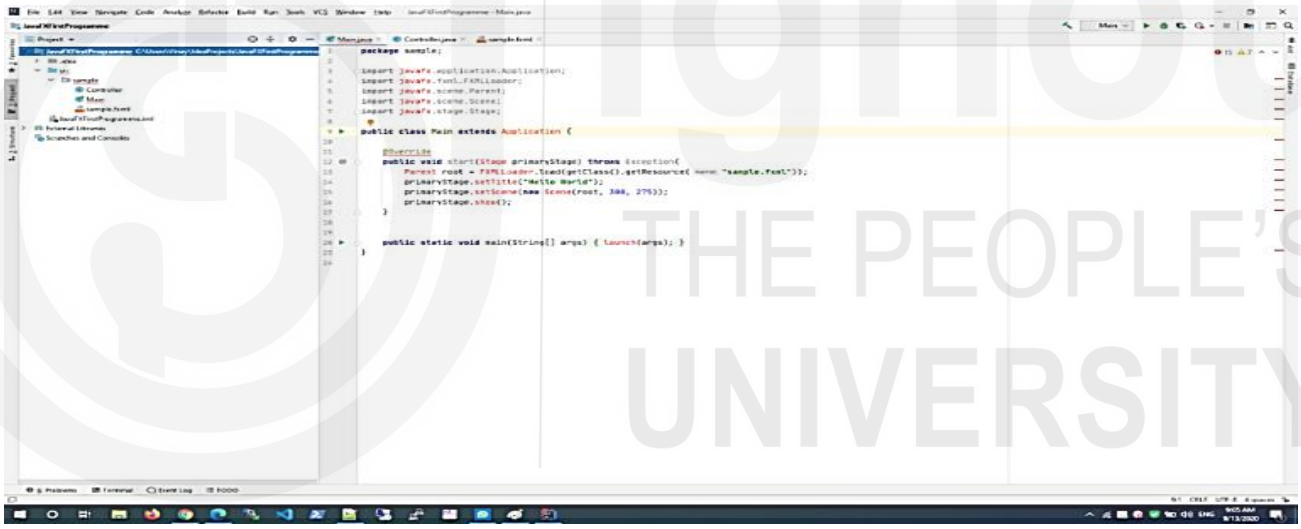

Figure 13.7: First look of default JavaFX Project with Error

5. Default project has been created with error because of running JDK 11 and missing the JavaFX SDK. This error will not generate if you use JDK8, because JDK 8 provides an inbuilt library of JavaFX. Then you have to configure the global library. Then you may create a new project of JavaFX running on JDK 11.

6. For fixing the above error and setting up JavaFX Library in the Project, first we click right-click on the project and select open module settings as shown in figure 13.8 .

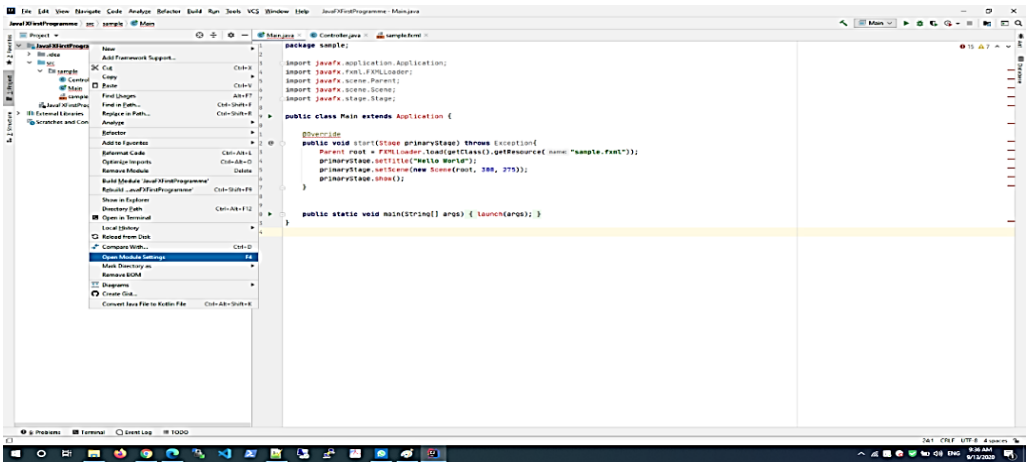

Figure 13.8: Open Project Setting

7. Firstly, check if the project SDK is correctly configured or not. Here the project language level needs to be configured the same as the project SDK selected during the Project creation. In the case of JDK 11, please follow instructions as specified in figure 13.9:

- In project Tab:
  - Project SDK 11
  - Project Language Level: 11-Local Variable Syntax for lambda Parameters
- In Module Tab:
  - Sources Language Level: 11- Local Variable Syntax for lambda Parameters
  - Dependency: Module SDK 11

if you are using JDK-8, you must select ***Project SDK 8*** and ***Project Language Level*: 8 – Lambdas, Type annotations etc**. If in your case, you have to set it as per your JDK version, you are having on your PC/laptop.

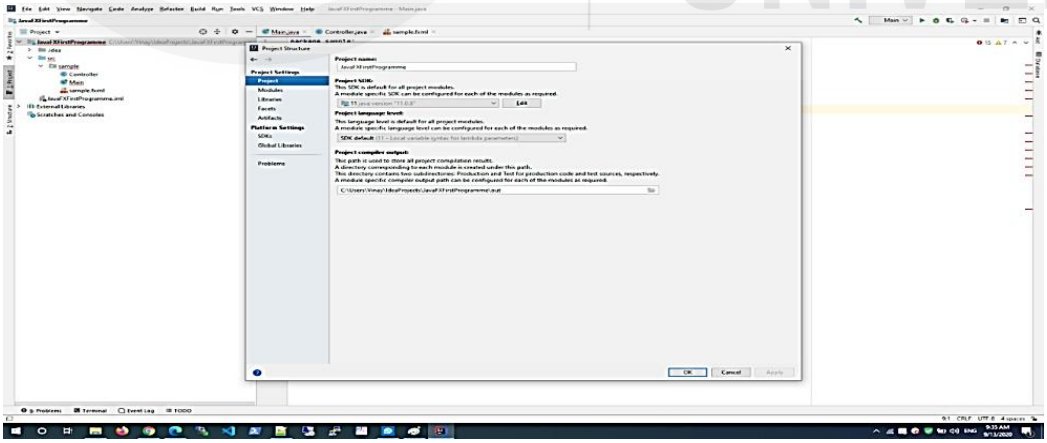

Figure 13.9: Project settings in IntelliJ IDEA

8. Next step is to configure global libraries of JavaFX-SDK-11. For this you have to go to the global libraries in Platform Setting option in Project Structure as shown in figure 13.10. Here we need to click on Plus(+) icon to add new Global Library. Doing this we are configuring IntelliJfor the specific libraries that we need to use in this particular Project.

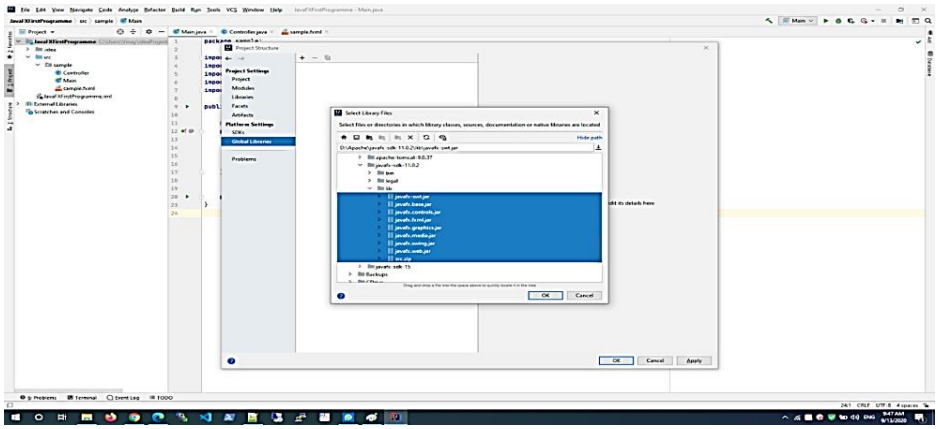

Figure 13.10: Setting Global Libraries in Project Structure

Now you will no longer get any errors as you observed inthe figure 13.7, but still you will find that if you try to run this application,you will get an **Error:** "***JavaFX runtime components are missing, and are required to run this application***". This error is only in case of JDK 11, but for JDK 8 you will not have this issue.

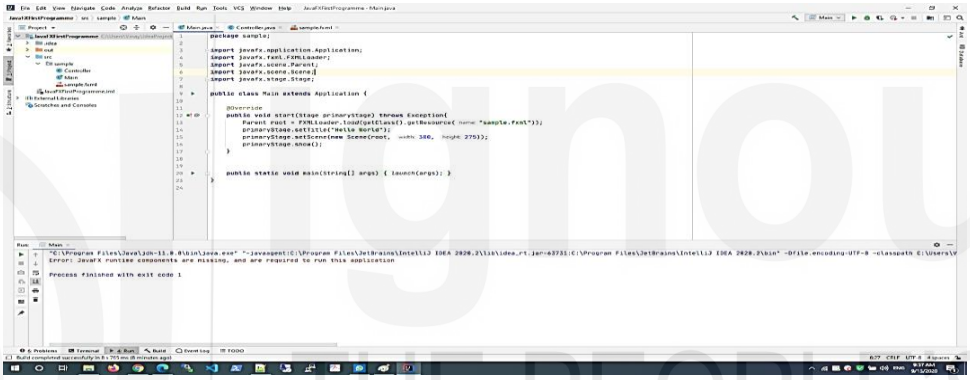

Figure 13.11(a):Default Project without error while useing JDK 11

To fix this error, you need to add a module - info.Java file in the source code directory, which defines the JavaFX control so that the code will work with JDK 11.

Righ click on src folder → New-->Module-Info.Java and add the following line of codes:

```
module JavaFXFirstProgramme
{
    requires Javafx.fxml;
    requires Javafx.controls;
    opens sample;
}
```

Now test run the application; you will get the following output:

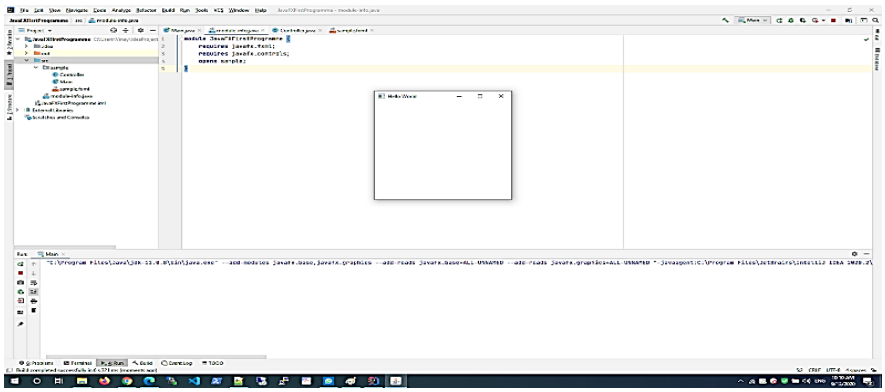

Figure 13.11(b): Output screen of First Run JavaFX Project

Finally, we have a directory structure of the First JavaFX Project as shown in figure 13.12.

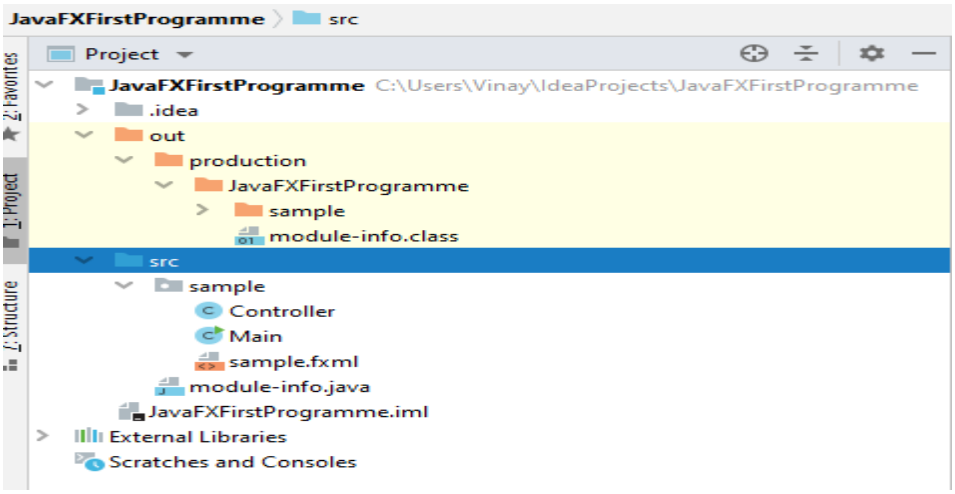

Figure 13.12: JavaFX Project Directory Structure

13.3 FEATURES OF JAVAFX

In this section, we will discuss the features of JavaFX along with its components. As per the official JavaFX documentation, the following features have been incorporated since the release of JavaFX 8 and later versions. The main items which were announced since the release of JavaFX 8 are given below:

- JavaFX is completely written in Java which contains classes and interfaces. Its API is designed for alternative use of Java Virtual Machines (JVM) such as JRuby and Scala.
- JavaFX supports two types of coding styles FXML and Scene builder. FXML is an XML based interface; however, scene builder is a pure Java interface. Scene builder in JavaFX is used for interactively designing the graphical user interface (GUI). Scene builder in JavaFX creates FXML, which can be ported to an IDE where a developer can also add the application's business logic.
- JavaFX supports WebView using WebKitHTML technology to embed web pages. JavaScript running in WebView can be called by Java APIs. Since the release of JavaFX 8, it has also supported HTML5 containing Web Sockets, Web Workers, Web Fonts and printing capabilities features.
- Some JavaFX features can be implemented in existing Swing applications, such as enabled web content and rich graphics media playback. This feature has been incorporated since the release of JavaFX 8. All the key controls are required to develop full-featured application, which is available in JavaFX 8 release, including standard Web Technology such as CSS(e.g., DatePicker and TreeTableViewcontrols are also available in JavaFX including a public API for CSS Styleable class which allows objects to be styled by CSS.)
- The 3D Graphics libraries are introduced in JavaFX 8 release. API for Shape3D (Box, Cylinder, MeshView and Sphere subclasses), SubScene, PickResult, Material, SceneAntialiasing and LightBase (Ambient Light, Point Lightsub classes) are added in 3D Graphics libraries. It also comprises of the Camera API class.

- Canvas API:This APIallows drawing directly within an area of the JavaFX scene that contains one graphical element which is also known as node.
- Printing API: The Javafx.print package isincluded in Java SE 8 release provides the public classes for the JavaFX Printing API.
- Rich Text Support: The JavaFX 8 brings enriched text support to JavaFX comprising bi-directional text and complex text scripts such as Thai and Hindu in controls and multi-line, multi-style text in text nodes.
- Multi-touch Support: The JavaFX provides support for multi-touch operations based on the capabilities of the underlying platform.
- Hi-DPI support: The JavaFX 8 also supports Hi-DPI displays(Hi-DPI displays have increased pixel density).
- Hardware-accelerated graphics pipeline: The JavaFX graphics are based on the graphics rendering pipeline (Prism). JavaFX allows smooth graphics that render quickly through Prism when it is used with a supported graphics card or graphics processing unit (GPU). If a system does not feature one of the recommended GPUs supported by JavaFX,then the Prism defaults that to the software rendering stack.
- High-performance media engine: The media pipeline supports the playback of web multimedia content. It provides a stable, low-latency media framework that is based on the 'GStreamer' multimedia framework.
- Self-contained application deployment model: Self-contained application packages have all of the application resources and a private copy of the Java and JavaFX runtimes. They are distributed as native installable packages and provide the same installation and launch experience as native applications for that operating system.

The above items related to JavaFX are given here to make you aware of the various features and supports provided by JavaFX8. Detailed coverage of the above items/points are beyond scope of this course.

13.4 USER INTERFACE COMPONENTS OF JAVAFX

Your familiarity with the form components available in HTML will help you learn JavaFX components.InJavaFX many components are available for developing GUI. In this section, you will learn components in JavaFX. JavaFX controls are the components that provide control functionalities in application development using JavaFX. In GUI Programming, the UI element is the core graphical element that users see and interact with. JavaFX also provides a wide list of common elements/controls (e.g., label, checkbox ,textbox, menu, a button, radio button, table, tree view, date picker etc.) from basic to advanced level of uses in JavaFX programming. The package "**Javafx.controls**" in JavaFX defines various classes to create the GUI components (controls). The package "**Javafx.scene.chart**" define various types of charts and package. The "**Javafx.scene.Scene**" provides the scene and its components that are available for the JavaFX UI toolkit. JavaFX supports several controls like Table view, Treeview, FileChooser, date picker, button text field etc. Controls are mainly nested inside a layout component, and it manages the layout of controls. The following list of important controls available in JavaFX:

• Accordion • Button • CheckBox • ChoiceBox • ColorPicker • ComboBox • DatePicker	• Label • ListView • Menu • MenuBar • PasswordField • ProgressBar • RadioButton	• Slider • Spinner • SplitMenuBut ton • SplitPane • TableView • TabPane • TextArea	• TextField • TitledPane • ToggleButton • ToolBar • TreeTableView • TreeView

Every component needs to be initialized with a new key before it is used. Let us see some important components one by one:

- **Accordion:** Accordion is the graphical control that looks like a collapsible
- content panel to display textual or graphical information in limited space along with a scrollbar if needed. It is very similar to the accordion component in HTML and bootstrap. It's used as a container that contains multiple controls internally (such as label, textbox, button, image, etc), each of which may have its own contents. The Accordion control is implemented by the class "**Javafx.scene.control.Accordion**".

Example 3: JavaFX Accordion implementation

```
import Javafx.application.Application;
import Javafx.fxml.FXMLLoader;
import Javafx.scene.Parent;
import Javafx.scene.Scene;
import Javafx.scene.control.Accordion;
import Javafx.scene.control.Label;
import Javafx.scene.control.TitledPane;
import Javafx.scene.layout.VBox;
import Javafx.stage.Stage;
public class MyAccordionControlExampleextends Application
{
@Override
public void start(Stage primaryStage) throws Exception
   {
Parent root = FXMLLoader.load(getClass().getResource("mySample.fxml"));
primaryStage.setTitle("Accordion Control Example");
primaryStage.setScene(new Scene(root, 500, 500));

Accordion accordion1 = new Accordion();

TitledPane pane1 = new TitledPane("Student" , new Label("Show all menu available in
Student"));
TitledPane pane2 = new TitledPane("Faculty"  , new Label("Show all menu available in
Faculty"));
TitledPane pane3 = new TitledPane("Books", new Label("Show all menu available in
Student Books"));
accordion1.getPanes().add(pane1);
accordion1.getPanes().add(pane2);
accordion1.getPanes().add(pane3);
VBoxvBox= new VBox(accordion);
Scene scene= new Scene(vBox);
primaryStage.setScene(scene);
primaryStage.show();
   }
public static void main(String[] args)
   {
launch(args);
  }
```

}

Output:

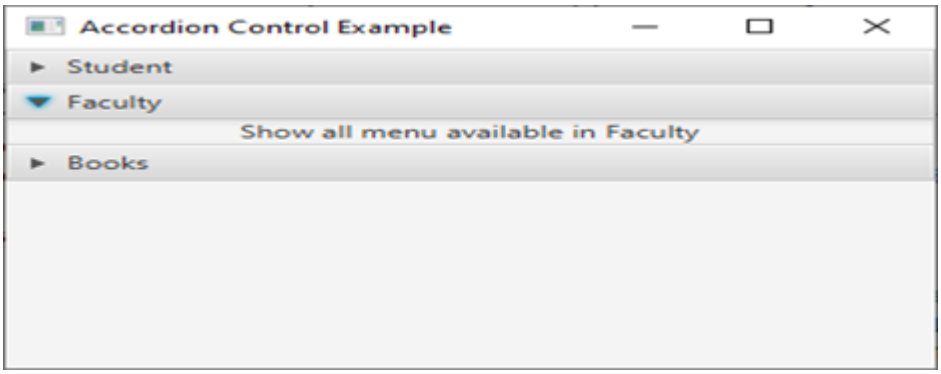

Figure 13.13: Output screen of Example no 3

- **Label:** It is aJavaFX component that is used todisplay textual or graphical content in GUI. It's also called non-editable text control. It can be created using the class **"Javafx.scene.control.Label".**
- **Button:** The button control is used for enabling some action executed when the application user clicks the button. The Button control is created using the class **"Javafx.scene.control.Button".**
- **ColorPicker:** It is a component used to choose or manipulate color in a dialog box. The ColorPicker control is implemented by the class **"Javafx.scene.control.ColorPicker".**
- **CheckBox:** This component enables users to check/tick a component that can be either on (true) or off (false). The CheckBox control is implemented by the class **"Javafx.scene.control.CheckBox".**
- **RadioButton:** This component is used in a situation where one needs to have either an ON (true) or OFF (false) state in a group. This control is implemented by the class **"Javafx.scene.control.RadioButton".**
- **ListView:** A ListView component of JavaFX is used to present the user with a scrolling list of text items. This control is represented by the class **"Javafx.scene.control.ListView".**
- **TextField:** A TextField component in JavaFX allows users to edit a single line text. It is implemented by the call **"Javafx.scene.control.TextField".**
- **PasswordField:** A PasswordField is a special text component that allows the user to enter a secure or sensitive text in a text field. It differs from TextField because the entered text does not show after typing. It is implemented by the class **"Javafx.scene.control.PasswordField".**
- **Scrollbar:** A Scrollbar component is used toallow the user to select from a range of values.
- **FileChooser:** A FileChooser control provides a dialog window from which the user can select a file.
- **ProgressBar:** Using this component, as the task progresses towards completion, the task's percentage of completion can be shown.
- **Slider:** A Slider lets the user graphically select a value by sliding a knob within a bounded interval.

13.5 WORK WITH LAYOUTS

Layout in GUI programming is used for organizing the UI elements or components on the screen and providing a final look and feel to the GUI (Graphical User Interface). This section will discuss the Layoutsand what JavaFX Layout allows us to do. The JavaFX provide various predefined layouts such as **Grid Pane, Anchor Pane, Stack Pane, H Box, V Box, Flow Pane, Tile Pane, Border Pane, Text Flow**, etc. There is no need to memorize all the layouts because each layout is denoted by a class and all these classes belong to the package **Javafx.layout**. The base class of all layouts in JavaFX is **Pane** class. The default layout size in JavaFX is 300x275.

Before you get into the layouts, let us discuss what is meant by the preferred size for JavaFX controls because preferred sizes are anessential concept. Every control computes its preferred size based on its contents, so what is the need of talking about preferred size? What is meant by the preferred width and height of the control? When it is displayed?

JavaFX has a button control. You are familiar with buttons, especially ones for ok and cancel, which you have seen in various applications. You might have observed that by default the button control will size itself. In other words, if we are working with an ok button, the button control will size itself so that its border fits around that text ok. So, it wouldnot stretch itself across the width of the entire window. For example, it will only be just wide enough to accept that text, so layouts often use a preferred size of the controls they are laying out to determine how much space it controls.When the controller is placed into a layout it becomes a child of that layout, so some layouts will ensure that their children display at the preferred widths or heights and sometimes it would depend on where controllers are placed within the layout.

Layout Pane Classes

Let us discuss Layout Panes Classes;JavaFX contains several container classes. Figure 13.14 shows the Hierarchy of Layout Classes provided by JavaFX. Layout Panes is the subclass of the package Javafx.scene.layout. Layout can be created using Java files as well as FXML files.

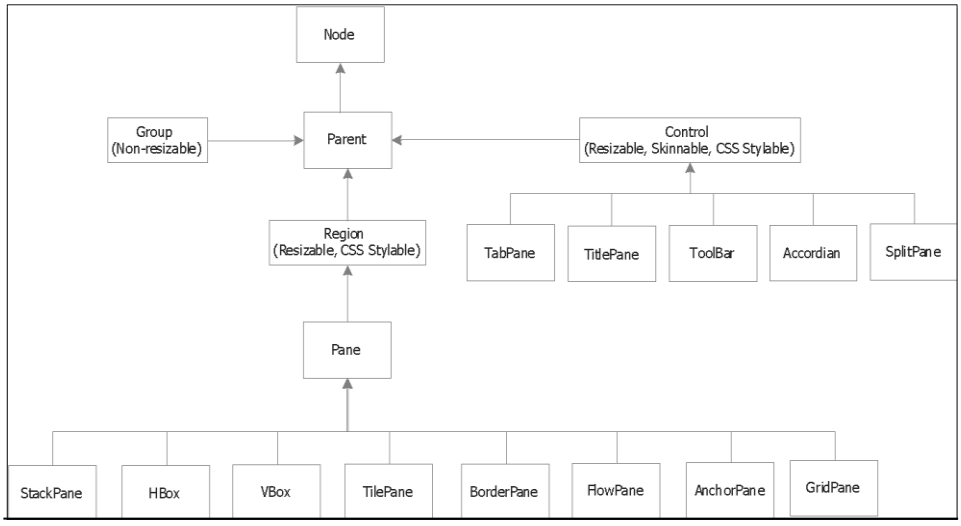

Figure 13.14: JavaFX 2.0 Layout Hierarchy

GridPane is the default layout of JavaFX Project setup as shown in figure 13.15.

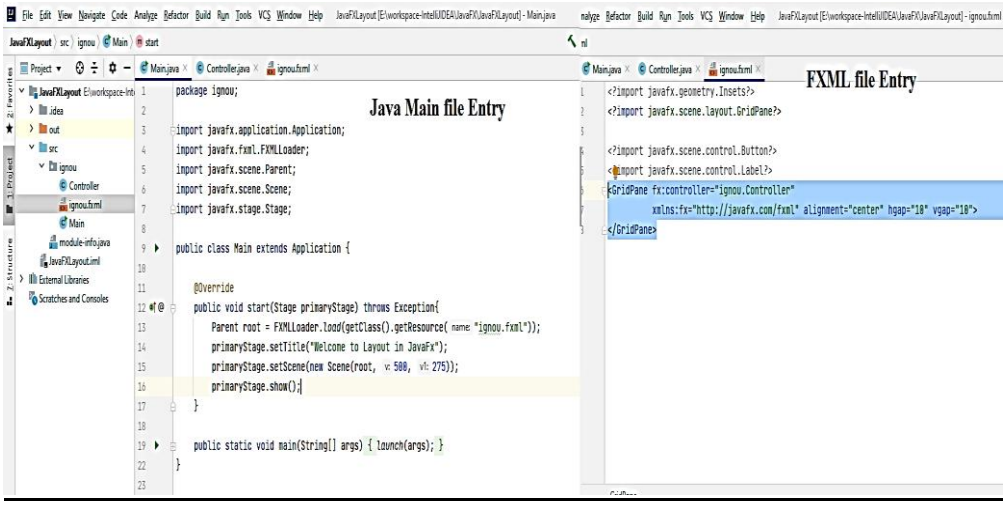

Figure 13.15: Default Pane coding inJavaFX Project

- **HBox:** The HBox layout organizes all the nodes in an application in a single horizontal row. The class named HBox of the package Javafx.scene.layout signifies the text horizontal box layout.

  Syntax in FXML:

  ```
  <?import Javafx.scene.layout.HBox?>
  <HBoxfx:controller="ignou.Controller"
  xmlns:fx=http://Javafx.com/fxmlalignment="center">
  <Label text="Welcome to HBox Layout using FXML"/>
  <Button text="Button 1"/>
  <Button text="Button 2"/>
  </HBox>
  ```

  Syntax in Java:

  ```
  Label label1 =new Label("Welcome to HBox Layout with Pure Java code");
  Button button1 = new Button("Button Number 1");
  Button button2 = new Button("Button Number 2");
  HBoxhbox = new HBox(label1, button1, button2);
  primaryStage.setScene(new Scene(hbox, 500, 275));
  ```

- **VBox:** The VBox layout organizes all the nodes in our application in a single vertical column. The class named VBox of the package Javafx.scene.layout denotes the text Vertical box layout.

  Syntax in FXML:

  ```
  <?import Javafx.scene.layout.VBox?>
  <VBoxfx:controller="ignou.Controller" xmlns:fx="http://Javafx.com/fxml"
  alignment="center">
  <padding>
  <Insets bottom="10" right="10"/>
  </padding>
  <Label text="Welcome to VBox Layout"/>
  <Button text="Button 1"/>
  <Button text="Button 2"/>
  </VBox>
  ```

  Syntax in Java:

  ```
  Label label1 = new Label("Welcome to VBox Layout using Java Code");
  Button button1 = new Button("Button 1");
  Button button2 = new Button("Button 2");
  ```

```
VBoxvbox = new VBox(label1, button1, button2);
primaryStage.setScene(new Scene(vbox, 500, 275));
```

- **BorderPane:** The Border Pane layout organizes the nodes in our applications in top, left, right, bottom and center positions. The class BorderPane of the package Javafx.scene.layout is used for the border pane layout.

  Syntax in FXML:

```
<?import Javafx.scene.layout.BorderPane?>
<BorderPanefx:controller="ignou.Controller"
xmlns:fx="http://Javafx.com/fxml">
<bottom>
<HBoxxmlns:fx="http://Javafx.com/fxml">
<padding>
<Insets bottom="10" right="10"/>
</padding>
<Label text="Welcome to BorderPaneLayout"/>
<Button text="Button 1" prefWidth="90"/>
<Button text="Button 2" prefWidth="90"/>
</HBox>
</bottom>
</BorderPane>
```

  Syntax in Java:

```
Label label1 = new Label("Welcome to BorderPane Layout using Java Code");
Button button1 = new Button("Button 1");
Button button2 = new Button("Button 2");
BorderPaneborderPane = new BorderPane();
borderPane.setTop(label1);
borderPane.setRight(button1);
borderPane.setLeft(button2);
primaryStage.setScene(new Scene(borderPane, 500, 275));
```

- **StackPane:**The stack pane layout organizes the nodes in an application on top of another component like in a stack. In this pane, the node added first is placed at the bottom of the stack and the next node added is placed on top of it. The class StackPane is in the package Javafx.scene.layout denotes the stack pane layout. In the following example Button 1 is top of the Label, and Button 2 is top of the Button 1.

  Syntax in FXML:

```
<?import Javafx.scene.layout.StackPane?>
<StackPanefx:controller="ignou.Controller"
xmlns:fx="http://Javafx.com/fxml">
<padding>
<Insets bottom="10" right="10"/>
</padding>
<Label text="Welcome to StackPane Layout"/>
<Button text="Button 1" />
<Button text="Button 2"/>
</StackPane>
```

Syntax in Java:

```
Label label1 = new Label("Welcome to StackPane Layout using Java Code");
Button button1 = new Button("Button 1");
Button button2 = new Button("Button 2");
StackPanestackPane = new StackPane();
stackPane.setMargin(label1, new Insets(50, 50, 50, 50));
ObservableListobservableList = stackPane.getChildren();
observableList.addAll(label1, button1, button2);
primaryStage.setScene(new Scene(stackPane, 700, 275));
```

- **TextFlow:**The Text Flow layout organizes multiple text nodes in a single flow. The class TextFlowin the package Javafx.scene.layout denotes the text flow layout.

Syntax in FXML:

```
<?import Javafx.scene.text.TextFlow?>
<TextFlowfx:controller="ignou.Controller"
xmlns:fx="http://Javafx.com/fxml" >
<padding>
<Insets bottom="10" right="10"/>
</padding>
<Label text="Welcome to TextFlow Layout"/>
<Button text="Button 1"/>
<Button text="Button 2"/>
</TextFlow>
```

Syntax in Java:

```
Text text1 = new Text("Welcome to TextFlow Layout");
text1.setFont(new Font(15));
text1.setFill(Color.DARKSLATEBLUE);

Button btn1 = new Button("Button 1");
Btn1.setFont(Font.font("Times new Roman", FontWeight.BOLD, 12));

TextFlowtextFlowPane = new TextFlow();

textFlowPane.setTextAlignment(TextAlignment.JUSTIFY);
textFlowPane.setPrefSize(10, 10);
textFlowPane.setLineSpacing(5.0);
ObservableList list1 = textFlowPane.getChildren();
list1.addAll(text1, btn1);
Scene scene = new Scene(textFlowPane);
stage.setTitle("text Flow Pane Example");
stage.setScene(scene);

stage.show();
```

- **AnchorPane:** The Anchor pane layoutis used to anchor the nodes in an applications at a particular distance from the pane. The class AnchorPane in the package Javafx.scene.layout denotes the Anchor Pane layout.

Syntax in FXML:

```
<?import Javafx.scene.layout.AnchorPane?>
<AnchorPanefx:controller="ignou.Controller"
xmlns:fx="http://Javafx.com/fxml">
<padding>
```

```
<Insets bottom="10" right="10"/>
</padding>
<Label text="Welcome to AnchorPane Layout"/>
<Button text="Button 1" />
<Button text="Button 2"/>
</AnchorPane>
```

Syntax in Java:

```
Text text1 = new Text("Welcome to AnchorPane Layout");
text1.setFont(new Font(15));
text1.setFill(Color.DARKSLATEBLUE);
Button btn1 = new Button("Button 1");
Btn1.setFont(Font.font("Times new Roman", FontWeight.BOLD, 12));

AnchorPane anchorPane1 = new AnchorPane();
anchorPane1.setTopAnchor(text1, 50.0);
anchorPane1.setTopAnchor(btn1, 50.0);
VBoxvBox=new VBox(anchorPane1);
```

- **TilePane:** Using Tile Pane layout the nodes in an application are added in the form of uniformly sized tiles. The class Tile Paneis in the package Javafx.scene.layout denotes the TilePane layout.

  Syntax in FXML:

```
<?import Javafx.scene.layout.TilePane?>
<TilePanefx:controller="ignou.Controller"
xmlns:fx="http://Javafx.com/fxml">
<padding>
<Insets bottom="10" right="10"/>
</padding>
<Label text="Welcome to TilePane Layout"/>
<Button text="Button 1" />
<Button text="Button 2"/>
</TilePane>
```

  Syntax in Java:

```
Text text1 = new Text("Welcome to TilePaneLayout");
text1.setFont(new Font(15));
text1.setFill(Color.DARKSLATEBLUE);
Button btn1 = new Button("Button 1");
Btn1.setFont(Font.font("Times new Roman", FontWeight.BOLD, 12));
TilePane tilePane1 = new TilePane();

tilePane1.getChildren().add(text1);
tilePane1.getChildren().add(btn1);
Scene scene = new Scene(tilePane1, 10, 10);
```

- **GridPane:** The Grid Pane layout is used to organizethe nodes in an applicationas a grid of rows and columns. This layout comes handy while creating forms using JavaFX. The class GridPane in the package Javafx.scene.layout denotes the GridPane layout.

  Syntax in FXML:

```
<?import Javafx.scene.layout.GridPane?>
<GridPanefx:controller="ignou.Controller"
xmlns:fx="http://Javafx.com/fxml" alignment="center" hgap="10"
vgap="10">
```

```
<Label text="Welcome to Gridpane Layout" GridPane.rowIndex="0"
GridPane.columnIndex="1"/>
<Button text="Button 1" GridPane.columnIndex="0"
GridPane.rowIndex="1"/>
<Button text="Button 2" GridPane.columnIndex="0"
GridPane.rowIndex="2"/>
</GridPane>
```

Syntax in Java:

```
Text text1 = new Text("Welcome to GridpaneLayout");
text1.setFont(new Font(15));
text1.setFill(Color.DARKSLATEBLUE);
Button btn1 = new Button("Button 1");
Btn1.setFont(Font.font("Times new Roman", FontWeight.BOLD, 12));

GridPanegridPane1= new GridPane ();
gridPane1.add(text1, 0, 0, 1, 1);
gridPane1.add(btn1, 1, 0, 1, 1);
Scene scene = new Scene(gridPane1, 40, 20);
```

- **FlowPane:** The flow pane layout wraps all the nodes in a flow. A horizontal flow pane wraps the elements of the pane at its height, while a vertical flow pane wraps the elements at its width. The class FlowPane in the package Javafx.scene.layout denotes the Flow Pane layout.

Syntax in FXML:

```
<?import Javafx.scene.layout.FlowPane?>
<FlowPanefx:controller="ignou.Controller"
xmlns:fx="http://Javafx.com/fxml" orientation="HORIZONTAL">
<padding>
<Insets bottom="10" right="10"/>
</padding>
<Label text="Welcome to FlowPane Layout"/>
<Button text="Button 1" />
<Button text="Button 2"/>
</FlowPane>
```

Syntax in Java:

```
Text text1 = new Text("Welcome to FlowPane Layout");
text1.setFont(new Font(15));
text1.setFill(Color.DARKSLATEBLUE);
Button btn1 = new Button("Button 1");
Btn1.setFont(Font.font("Times new Roman", FontWeight.BOLD, 12));

FlowPane flowpane1 = new FlowPane();

flowpane1.getChildren().add(text1);
flowpane1.getChildren().add(btn1);

Scene scene = new Scene(flowpane1, 40, 20);
```

Steps to create layout in JavaFX :

- **Create node:** First of all, create the required nodes of the JavaFX application by instantiating their respective classes.
- **Instantiate the respective class of the required layout:** After creating the nodes (and completing all the operations on them), instantiate the class of

the required layout.

- **Set the properties of the layout:** After instantiating the class, you need to set the layout's properties using their respective setter methods.
- **Add all the created nodes to the layout:** Finally, you need to add the object of the shape to the group by passing it as a parameter of the constructor.

Example 4: Creating a Layout using GridPane in pure Java Code

```
package org.ignou.gui;
import Javafx.application.Application;
import Javafx.geometry.Insets;
import Javafx.geometry.Pos;
import Javafx.scene.Scene;
import Javafx.scene.control.Button;
import Javafx.scene.control.Label;
import Javafx.scene.control.PasswordField;
import Javafx.scene.control.TextField;
import Javafx.scene.layout.GridPane;
import Javafx.scene.layout.HBox;
import Javafx.scene.text.Font;
import Javafx.scene.text.FontWeight;
import Javafx.scene.text.Text;
import Javafx.stage.Stage;
public class MyGridPaneExampleextends Application
{
@Override
public void start(Stage primaryStage) throws Exception
{
primaryStage.setTitle("Welcome to GridPane Layout in JavaFx
using Pure Java code");

GridPane grid1= new GridPane();
grid1.setAlignment(Pos.CENTER);
grid1.setHgap(10);
grid1.setVgap(10);
grid1.setPadding(new Insets(25, 25, 25, 25));

Text sceneTitle= new Text("Welcome to First Layout in JavaFx
using Pure Java code");
sceneTitle.setFont(Font.font("Times new Roman",
FontWeight.NORMAL, 20));
grid1.add(sceneTitle, 0, 0, 2, 1);

Label userName= new Label("User Name:");
grid1.add(userName, 0, 1);

TextFielduserTextField= new TextField();
grid1.add(userTextField, 1, 1);

Label pw = new Label("Password:");
grid1.add(pw, 0, 2);

PasswordFieldpwBox= new PasswordField();
grid1.add(pwBox, 1, 2);

Button btn= new Button("Login");
btn.setFont(Font.font("Times new Roman", FontWeight.BOLD,
20));
grid1.add(btn, 1, 4);

primaryStage.setScene(new Scene(grid1, 700, 275));
primaryStage.show();
}
public static void main(String[] args)
```

```
{
launch(args);
}
}
```

Output:

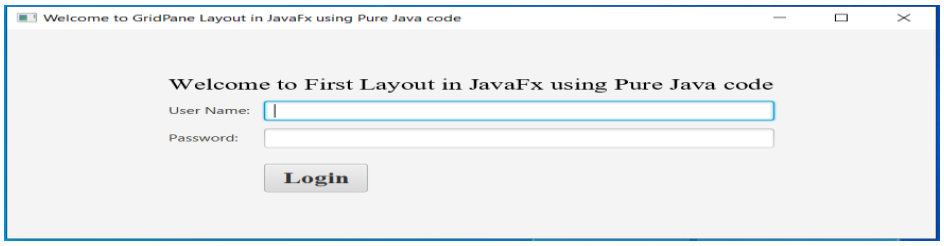

Figure 13.16: Output Screen of Example no 4

☞Check Your Progress -1

1) What is the difference between AWT and Swing?

……

……

……

2) Define user interface component in JavaFX.

……

……

……

3) Explain the Layout Hierarchy in JavaFX.

……

……

……

13.6 ADD HTML CONTENT

Before going to HTML content in JavaFX, let us briefly describe CSS in HTML. What is a Cascading Style Sheet? A cascading style sheet (CSS) is a language used to describe the presentation (the look or the style) of UI elements in a GUI application. CSS was primarily developed for use in web pages for styling HTML elements. It allows for the separation of the presentation from the content and behaviour. In a typical web page, the content and presentation are defined using HTML and CSS, respectively. JavaFX allows us to define the look and feel (or the style) of JavaFX applications using CSS. It can define UI elements using JavaFX class libraries or FXML and use CSS to define their look and feel. The CSS provides the syntax to write rules to set the visual properties. The following are the steps to write the CSS classes:

- A rule consists of a selector and a set of property-value pairs.
- A selector is a string that identifies the UI elements to which the rules will be applied.
- A property-value pair will have a property name and its corresponding value separated by a colon (:).
- Two property-value pairs are separated by a semicolon (;).

- A set of property-value pairs is enclosed within curly braces ({ }) preceded by the selector.

An example of a rule in CSS is given below:

```
.button { -fx-background-color: red;
          -fx-text-fill: white;}
```

What are Styles, Skins, and Themes?

- Styles, skins, and themes are three related concepts. A CSS rule is also known as a style. A collection of CSS rules is known as a style sheet. Styles provide a mechanism to separate the presentation and content of UI elements. They also facilitate grouping of visual properties and their values, so that they can be shared by multiple UI elements. JavaFX provides styles using JavaFX CSS.
- Skins are collections of application-specific styles which define the appearance of an application. Skinning is the process of changing the appearance of an application (or the skin) on the fly. JavaFX does not provide a specific mechanism for skinning.
- Themes are the visual characteristics of an operating system that are reflected in the appearance of UI elements of all applications. For example, changing the theme on the Windows operating system changes the appearance of UI elements in all applications that are running. The skins are application specific, whereas themes are operating system specific.

To implement CSS Style in JavaFX through style file the following steps need to be followed:

- Create a resources/css folder in the project folder
- Add CSS file(Style Sheet) in css folder
- Write rules in Stylesheet as explained above
- Set scene property using getStylesheets() method
  getStylesheets().add("file:resources/css/style.css");
- Set style/CSS ID inJavaFX's component using getStyleClass() method
  getStyleClass().add("button1"); (where button1 is the same ID ruled in CSS.)

Example 5: CSS Style implementation in Java FX

1. MyHtmlCSSStyleExample.Java

```
package org.ignou.gui;

import Javafx.application.Application;
import Javafx.scene.Scene;
import Javafx.scene.control.Button;
import Javafx.scene.layout.HBox;
import Javafx.scene.text.Text;
import Javafx.stage.Stage;

public class MyHtmlCSSStyleExampleextends Application
{
 public static void main(String[] args)
 {
 Application.launch(args);
 }

 @Override
 public void start(Stage stage)
 {
  Text text1 = new Text();
  text1.setText("Welcome to use of CSS in JavaFX");
```

```
        /* Call Style ID from CSS file */
        text1.getStyleClass().add("text1");

        Button yesBtn= new Button("Yes");

        /* Call Style ID from CSS file */
        yesBtn.getStyleClass().add("button");

        Button noBtn= new Button("No");

        /* Call Style ID from CSS file */
        noBtn.getStyleClass().add("button1");

        Button cancelBtn= new Button("Cancel");

        /* Call Style ID from CSS file */
        cancelBtn.getStyleClass().add("button2");


        HBox root = new HBox();
        root.getChildren().addAll(text1, yesBtn, noBtn, cancelBtn);
        Scene scene1 = new Scene(root, 700, 70);

        /* Add a stylesheet to the scene*/
        scene1.getStylesheets().add("file:resources/css/style.css");
        stage.setScene(scene1);
        stage.setTitle("HTML CSS Styling Example");
        stage.show();
    }
}
```

2. Create style sheet file in the location: resources/css/style.css

```
.button
{
-fx-background-color: blue;
-fx-text-fill: white;
-fx-font-size: 10px;
-fx-font: Arial;
-fx-alignment: left;
}

.button1
{
-fx-background-color: red;
-fx-text-fill: white;
-fx-font-size: 15px;
-fx-font: Times new roman;
-fx-alignment: center;
}
.button2
{
-fx-background-color: green;
-fx-text-fill: white;
-fx-font-size: 20px;
-fx-font: Arial;
-fx-alignment: right;
}

.text1
{
-fx-font: Arial;
-fx-font-size: 20px;
}
```

Output:

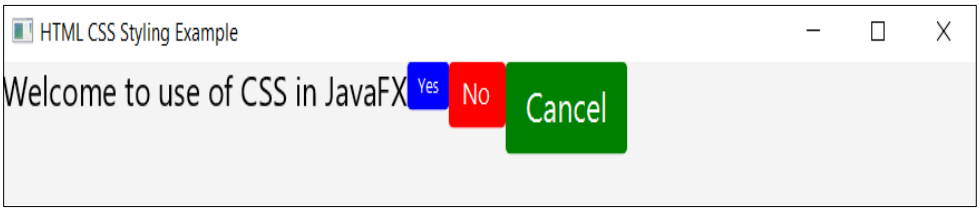

Figure 13.17: Output screen of Example no 5

13.7 ADD TEXT AND TEXT EFFECTS IN JAVAFX

As explained above, a single element in the scene is called a node which handles many types of content, which also include text. Text elements can be added as a textbox, label and using other components. The package of Text node is ***"Javafx.scene.text"*** and represented by the class named Text which is inherited from the Node class in JavaFX that is used to display text. As per requirement, it can apply effects, transformations and animation to text nodes. All node types permit us to provide cultured text contents that fulfil the demands of modern rich Internet applications using all such features.

To add a text object to an application, first, instantiate a class as follow:

```
Text text1=new Text ();
```

The data type of a class Text is string type which means it will store String text; to do that, we have to set the value using setText() method after instantiating a Text node:

```
String textString= "Welcome to Text Node";
text1.setText(textString);
```

Setting Text Position

A position of the text can be defined or set by specifying the value of the properties x and y using their respective setter methods namely setX() and setY():

```
text1.setX(50);
text1.setY(50);
```

Setting Text Font and Color

Text Font and color can be defined or set using its properties. You can use an instance of the "*Javafx.scene.text.Font*" class. The Font.font() method allows to specify the font family name and size. setFill() method can be used for setting a text color. Syntax of setting Text font and color as follow:

```
text1.setText("Setting Text Font and color");
text1.setFont(Font.font ("Times New Roman", 20));
text1.setFill(Color.RED);
```

Alternatively, we can also use a system font, which is platform dependent, means text font varies as per platform in which the application is running. To use this functionality, you have to call the Font.getDefault() method.

```
text1.setFont(Font.getDefault ());
```

Alternatively, we can also use custom fonts in JavaFX. For using a custom font, you can embrace a TrueType font (.ttf) or an OpenType (.otf) in the application. To comprise font as a custom font, follow the following procedure:

- Create a resources/fonts folder in the project folder.
- Copy your font files to the fonts subfolder in the project created.
- In the source code, load the custom font as follows

text1.setFont(Font.loadFont("file:resources/fonts/AlexBrush-Regular.ttf", 120));

Setting Text Bold or Italic

Use the FontWeight constant of the setFont() method to make Bold Text using follow syntax:

text1.setFont(Font.font ("Times New Roman", FontWeight.BOLD, 20));

The property named FontWeight accepts following 9 values:

- FontWeight.BLACK
- FontWeight.BOLD
- FontWeight.EXTRA_BOLD
- FontWeight.EXTRA_LIGHT
- FontWeight.LIGHT
- FontWeight.MEDIUM
- FontWeight.NORMAL
- FontWeight.SEMI_BOLD
- FontWeight.THIN

Use the FontPosture constant of the setFont() method to make italic Text using follow syntax:

text1.setFont(Font.font ("Times New Roman", FontPosture.ITALIC, 20));

The property named FontPosture accepts 2 values FontPosture.REGULAR and FontPosture.ITALIC.

Example 6: Setting a Font, Color and Font Size

```
package org.ignou.gui;

import Javafx.application.Application;
import Javafx.geometry.Pos;
import Javafx.scene.Group;
import Javafx.scene.Scene;
import Javafx.scene.layout.GridPane;
import Javafx.scene.paint.Color;
import Javafx.scene.text.Font;
import Javafx.scene.text.FontPosture;
import Javafx.scene.text.FontWeight;
import Javafx.scene.text.Text;
import Javafx.stage.Stage;

public class MyTextMainextends Application
{
@Override
public void start(Stage primaryStage)
{
primaryStage.setTitle("Welcome to Add Text and Text Effects in
JavaFX");
GridPane grid1 = new GridPane();
grid1.setAlignment(Pos.CENTER);
grid1.setHgap(10);
grid1.setVgap(10);

/* Initialize Text and add string to display along with positioning
in layout. */
Text text1 = new Text();
String textString= "Welcome to Text 1 Node";
text1.setText(textString);
text1.setX(10);
text1.setY(10);
```

```
/* Setting of Font with font size and font color. */
Text text2 = new Text();
text2.setText("Setting Text 2 Font and color");
text2.setFont(Font.font("Times New Roman", 20));
text2.setFill(Color.RED);
text2.setX(10);
text2.setY(35);

/* Make Font Weight Bold */
Text text3 = new Text();
text3.setText("Setting Text 3 Font Bold");
text3.setFont(Font.font("Arial", FontWeight.BOLD, 20));
text3.setX(10);
text3.setY(65);

/* Set Font Posture Italic */
Text text4 = new Text();
text4.setText("Setting Text 4 Italic Font");
text4.setFont(Font.font("Times New Roman", FontPosture.ITALIC,
20));
text4.setX(10);
text4.setY(90);

/* Use of Custom Font Example */
Text text5 = new Text();
text5.setText("Use of Custom font in JavaFX");
text5.setFont(Font.loadFont("file:resources/fonts/AlexBrush-
Regular.ttf", 40));
text5.setX(10);
text5.setY(130);

/* Grouping all text variables together */
Group group1 = new Group(text1, text2, text3, text4, text5);
grid1.add(group1, 0, 0);

/* putting text group on the scene */
primaryStage.setScene(new Scene(grid1, 700, 275));
primaryStage.show();
}
public static void main(String[] args)
{
launch(args);
}
}
```

Output:

Figure 13.18: Output screen of Example no 6

☞ Check Your Progress-2

1) Define the terms Styles, Skins and Themes.

.....

.....

2) Write a programme in JavaFX using the VBox pane.

……

……

……

3) Explain the setFont() method and its property with example.

……

……

……

13.8 SUMMARY

This unit gives an overview of the graphical user interface(GUI) in Java. In this unit AWT/Swing/JavaFX and User Interface Components of JavaFX are explained. Also GUI programming examples are givenusing coding style in AWT, Swing and JavaFX.Also, it is explained how to setup a project for JavaFX using intellij Idea and create the First Hello World program. The unit describedhow to work with layout in JavaFX along with coding style using FXML and pure Java code. This unit also explained concept of HTML Contents uses in JavaFX, text and various text effects in JavaFX with the help of programming examples.

13.9 SOLUTION/ANSWERS TO CHECK YOUR PROGRESS

☞ **Check Your Progress-1**

1) The difference between AWT and Swing are as follows:

Java Swing	Java AWT
It is platform-independent API.	It is platform-dependent API
It has lightweight GUI components.	It has heavyweight GUI components.
It supports pluggable look and feel.	It doesn't support pluggable look and feel.
It provides more advanced components then AWT like tables, lists, scrollpanes, colorchooser, tabbedpane etc.	it provides less components than Swing
It supports MVC design pattern	It does not follow the MVC design pattern.
Execution is faster	Execution is slower
The components of swing do not require much memory space	The components of AWT require more memory space

2) JavaFX controls are the components that provide control functionalities in JavaFX Applications. In GUI Programming, the user interface element are the core graphical elements,with that users see and interact with. JavaFX gives a wide list of common elements/controls such as label, textbox, checkbox, button, radio button, table, tree view, date picker, menu. The package "Javafx.controls" defines various classes to create the GUI components, charts, and skins available for the JavaFX UI toolkit.

3) A layout in GUI is required to organize the User Interface elements on the screen and provide a final look and feel to the Graphical User Interface(GUI). In JavaFX, Layout defines the way in which the components are to be seen on the stage. It basically organizes the scene-graph nodes. We have several built-in layout panes in JavaFX:HBox, VBox, StackPane, FlowBox, AnchorPane, etc. Each Built-in layout is denoted by a separate class that needs to be instantiated to implement that particular layout pane. All these classes are a member of Javafx.scene.layout package. The Javafx.scene.layout.Pane class is the base class for all the built-in layout classes in JavaFX.

☞ Check Your Progress-2

1) Styles provide a mechanism to separate the presentation and content of UI elements. They also facilitate the grouping of visual properties and their values to be shared by multiple UI elements. JavaFX provides styles using JavaFX CSS.

Skins are collections of application-specific styles which define the appearance of an application. Skinning is the process of changing the appearance of an application (or the skin) on the fly. JavaFX does not provide a specific mechanism for skinning. However, the JavaFX CSS and JavaFX API, available for the Scene class and other UI-related classes, can easily provide skinning for JavaFX applications.

Themes are visual characteristics of an operating system that are reflected in the appearance of UI elements of all applications. For example, changing the theme on the Windows operating system changes the appearance of UI elements in all applications that are running.

2)

```
package org.ignou.gui;
import Javafx.application.Application;
import Javafx.scene.Scene;
import Javafx.scene.control.Button;
import Javafx.scene.layout.VBox;
import Javafx.stage.Stage;
public class MyVBoxExample extends Application
{
    public void start (Stage stage) throws Exception
    {
        stage.setTitle("VBox Example");

        Button b1 = new Button ("Play ");
        Button b2 = new Button("Stop");

        VBox vbox1 = new VBox(b1, b2);
        Scene scene = new Scene (vbox1, 200, 100);
        stage.setScene(scene);
        stage.show();
    }
    public static void main (String[] args)
    {
       Application.launch(args);
    }
}
```

3) The setFont() method is used to change the font of the text. This method takes an object of the Font class. To set the font, you can use an instance of the Javafx.scene.text.Font class. The Font.font() method permits you to specify the font family name and size.

The setFont() method takes four parameters: family, weight, posture, and size. The **family** parameter signifies the family of the font that you want to apply to the text. The **weight** property denotes the weight of the font, and it comprises values such as BOLD, EXTRA_BOLD, EXTRA_LIGHT, LIGHT, MEDIUM, NORMAL. The **posture** property represents the font posture, such as regular or italic. The **size** property represents the size of the font, for example:

```
Text t = new Text ("Graphical User Interface");
t.setFont(Font.font ("Arial", 18));
t.setFont(Font.font("Tahoma", FontWeight.BOLD,FontPosture.ITALIC,
18));
```

13.10 REFERENCE/FURTHER READING

- Carl Dea, Gerrit Grunwald, José Pereda, Sean Phillips and Mark Heckler "JavaFX 9 by Example", Apress,2017.
- Sharan, Kishori, " BeginningJava 8 APIs, Extensions and Libraries: Swing, JavaFX, JavaScript, JDBC and Network Programming APIs",Apress, 2014.
- Package and classes in AWT (https://docs.oracle.com/Javase/7/docs/api/Java/awt/package-summary.html)
- JavaFXGetting Started with JavaFX Release 8(https://docs.oracle.com/Javase/8/Javafx/get-started-tutorial/index.html)
- https://www.oracle.com/Java/technologies/Javase/Javafx-docs.html
- https://docs.oracle.com/Javase/7/docs/api/Java/awt/package-summary.html
- https://openjfx.io/openjfx-docs/
- https://docs.oracle.com/Javase/8/Javafx/get-started-tutorial/jfx-overview.htm#BABEDDGH
- https://gluonhq.com/products/Javafx/
- https://openjfx.io/Javadoc/15/Javafx.controls/Javafx/scene/control/package-summary.html

UNIT 14 WORKING WITH UI CONTROLS

Structure **Page No.**

14.0 Introduction 373

14.1 Objective 374

14.2 Skin Applications with CSS 374

14.3 Build UI with FXML 380

14.4 Event Handling in JavaFX 383

14.5 Effects, Animation and Media 385

14.6 Summary 392

14.7 Solutions/Answers to Check Your Progress 392

14.8 References/Further Readings 394

14.0 INTRODUCTION

The previous unit of this block gave you an introduction to Graphical User Interface (GUI) in Java and also introduced UI controls. The JavaFX provides a wide list of UI control elements such as label, textbox, checkbox, button and radio button. The UI (User Interface) elements are those elements that are actually shown to the user for information exchange. The package javafx.scene.control have all the necessary classes required for the UI elements. This unit explains you how to JavaFX UI controls work with CSS, FXML and event handling.

Cascading Style Sheets (CSS) are used for adding styles such as text alignments, font-size, colors, including background(s) images and page settings in HTML documents. Basically, CSS is used to control the presentation of one or more web pages in your application. This unit designates the usage of cascading style sheets with JavaFX application. You are already aware of the XML; one similar language such as FXML will be covered in this unit. You will learn how to use FXML with JavaFX application. Events handling play very important role in Java Programming. For example, whenever you click on the button, select an item from drop-down menu, an event generates. For handling these events in JavaFx application, you will learn four stages such as Target selection, Route construction, Event capturing and Event bubbling in this unit. In event handling mechanism, a source generates an event and sends to one or more listeners. Once the listeners received this event, processes the event and then returns.

This unit continues the study of effects, animations and media in JavaFX. This Unit explains to you how to apply effects on UI elements such as Blend, DropShadow and Glow. You have seen many animated movies or images. In this unit you will use animation with user interface controls. You will also include media like audio or video in your application. For incorporating media in an application, JavaFX API provides necessary classes such as Media, MediaPlayer and MediaView. The main use of this package is media playback. This unit describes concepts with examples. The previous unit explained to you how to run JavaFX application with IDE. So, this unit is not describing this running procedure again. You can run examples of this unit whether with IDE (Eclipse/NetBeans/IntelliJ Idea, etc.) or through the command prompt.

14.1 OBJECTIVES

At the end of this unit, you will be able to:

- describe how to create CSS and how to apply this upon UI control elements in JavaFX application,
- build User Interface with FXML,
- handle events in JavaFX application,
- apply effects, animations on the UI controls, and
- incorporate audio or video media in JavaFX application.

14.2 SKIN APPLICATIONS WITH CSS

You are already aware of the Cascading Style Sheets (CSS), which are used for adding style such as fonts, colors, backgrounds and spacing between the paragraphs in HTML documents. It is used to control the presentation of one or more web pages. This section defines how to use cascading style sheets with JavaFX applications. You can use cascading style sheets to create a custom look and feel for your JavaFX application.

Before going into detail, you may recall the JavaFX application structure,which contains three keys components, namely Stage, Scene and Nodes as shown in thefollowing figure-1.A **stage (**or a window)embraces all the objects of a JavaFX application and it is represented by the Stage class of the package **javafx.stage**. The show() method can be used to display the contents of a stage. A **scene** graph is a data structure that is a collection of nodes, and is represented by package javafx.scene in a JavaFX application. The scene class holds all the contents of a scene graph. A **node** is a basic graphical object of a JavaFX application such as UI control objects (button, radio checkbox, etc.), 2D and 3D geometric objects (circle, rectangle, etc.),

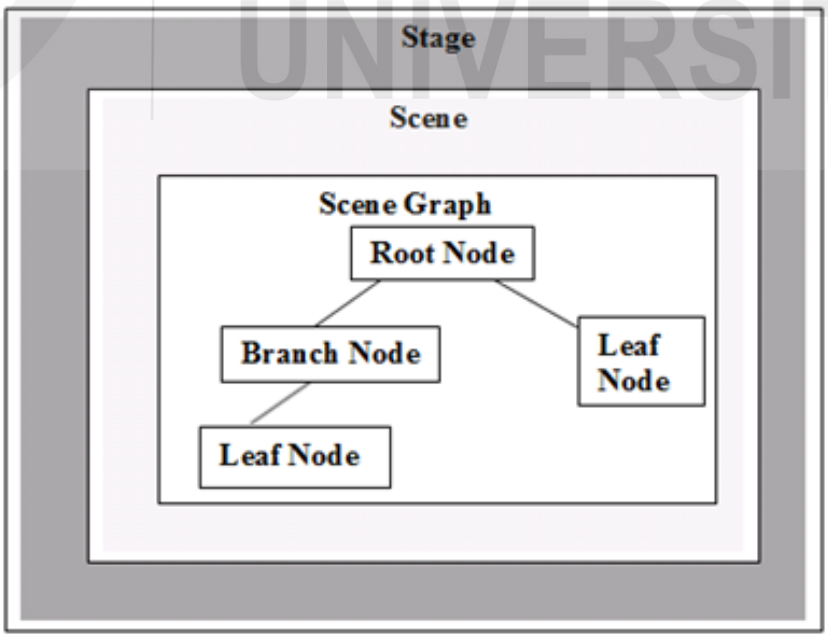

Figure 14. 1: JavaFX application structure

Container/layout objects (Border Pane, Grid Pane, etc.), media element objects (audio, video etc.).

You can use CSS in JavaFX applications similar to use CSS in HTML web documents. JavaFX CSS are created on basis of W3C CSS specification(accessible at https://www.w3.org/Style/CSS/Overview.en.html)with some additional JavaFX features. You can refer JavaFX CSS reference guide at the link of Oracle web page (https://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html).

JavaFX CSS

JavaFX provides a package javafx.css which contains the classes that are used to apply CSS in JavaFX applications.You can use CSS to enrich the look of the JavaFX application. A style sheet comprises rules that specify how formatting should be applied to the particular elements in your application. Each style rule contains two parts which are selector and declaration. The selector indicates which element(s) the declaration applies, and the declaration specifies the formatting properties of the element.

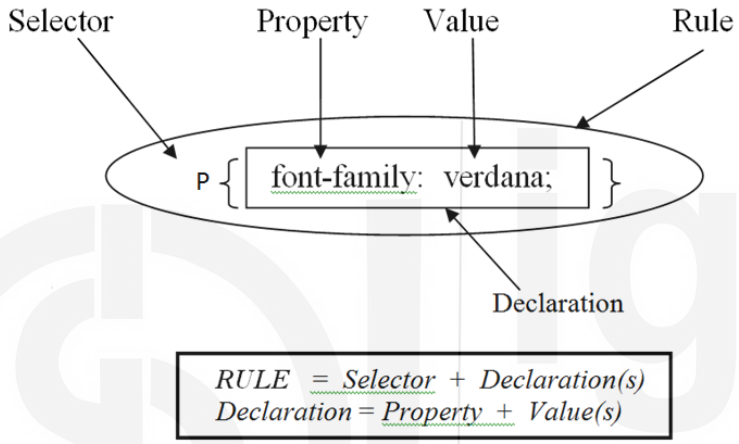

Figure 14.2: Format of CSS rules

JavaFX contains the new generation UI library, which is used to configure the theme of the application.

The JavaFX 8 application holds caspian.css as the default style sheet found in the JavaFX runtime JAR file (jfxrt.jar). The latest version of JavaFX application uses Medona.css as a style sheet that defines styles for the root node and the UI controls.There are several ways to apply a CSS style in a JavaFX UI controls such as JavaFX default CSS style sheet, Scene specific, Parent specific and Element specific style property. If you do not create your own style sheet, then JavaFX provides a default CSS style sheet that may be applicable to all JavaFX elements.

Creating Style Sheets

You can generate your own cascading style sheets which overrides the styles in the default style sheet. You can create style sheet(s) with .css extension, and it is placed in the same directory of the main class for your JavaFX application.You can create your external Style Sheet or inline Style Sheet for the JavaFX applications.

When you design either your external Style Sheet or inline Style Sheet, you may follow the rules for the selection of property name as :

- You must define "-fx-" as a prefix with all JavaFX property names.
- If more than one word exists in the property name, then use the hyphen (-) to separate them.

- Property name and their value are separated by colon (:)
- If more rules are defined for the Property name, then they are separated by a semicolon (;).

CSS class Selector

You can write a dot (.) character followed by the Style class selector name. It is used to identify more than one element in an application. In the following example, the font property name is preceded by prefix -fx- and Property name and their value are separated by colon, and rules are separated by semicolon for the JavaFX label element.

```
.label
{  -fx-font : bold 12pt "arial";   -fx-padding:10     }
```

For accessing the above-mentioned style class which are appropriate to label node, you can use the getStyleClass().add() method.

ID Styles

JavaFX provides us facility to create the style for the individual node. The style name can be given as the ID name preceded by the hash(#) symbolto select one unique element.

```
#submit
{
   -fx-font : bold 14pt "arial";
   -fx-background-color : #87ceeb;
}
```

You can use the above-mentioned style for the individual Submit button like the following way:

```
Button Submit=new Button ("Submit");   //submit button
Submit.setId("submit");    //set ID for the submit button
```

Create your own Style Sheet

You can add your own external style sheet to a scene in JavaFX application like the following:

```
Scene sc = new Scene(root, 400, 200);
sc.getStylesheets().add("path/name_of_ownstylesheet.css");
```

If you want to set a CSS style sheet on a parent layout element then style sheet will be applied to all elements inside that layout element or child element. It is similar to setting stylesheet on a Scene object. You can add your own external style sheet to a Parent element in JavaFX application like the following:

```
Button b1 = new Button("Submit");
Button b2 = new Button("Reset");
VBox vbox = new VBox(b1, b2);
vbox.getStylesheets().add("button_specfic_styles.css");
```

The following example illustrates to you how to create an own CSS for JavaFX application. This example displays Grade Card page which consists of controls such as label, text field and submit button. This example contains two files:

OwnCSSExample.java and Examplestyle.css. The source code for both files is listed below:

Example-1: Source code for OwnCSSExample.java

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.text.*;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;
public class OwnCSSExample extends Application
{
  public void start(Stage stage) throws Exception
  {
    Label enroll=new Label("Enroll No.");    //label creation for Enroll
    Label prg=new Label("Programme");  //label creation for programme

    //Creating Text Field for Enroll and programme
    TextField txtf1=new TextField();
    TextField txtf2=new TextField();

    Button Submit=new Button ("Submit");   //submit button
    Submit.setId("submit");    //set ID for the submit button

    GridPane gpane=new GridPane();    //creating grid pane

    //setting horizontal and vertical gaps between the rows
    gpane.setHgap(5);
    gpane.setVgap(5);

    Scene scene = new Scene(gpane,400,200);
    //adding the the nodes to the GridPane's rows
    gpane.addRow(0, enroll,txtf1);
    gpane.addRow(1, prg,txtf2);
    gpane.addRow(2, Submit);

    gpane.getStylesheets().add("Examplestyle.css");//add CSS file to the grid
    pane
    stage.setTitle("Own CSS Example");  //set title to the Stage
    stage.setScene(scene);//set scene to the stage
    stage.show();  //showing contents of the stage
  }
  public static void main(String[] args)
  {
    launch(args);
  }
}
```

Source code for Examplestyle.css

```
.label
{
-fx-font : bold 12pt "arial";
-fx-padding:10
}

#submit
{
```

```
-fx-font : bold 14pt "arial";
-fx-background-color : #87ceeb;
-fx-text-fill:red;
-fx-padding: 8px 8px
}
```

Now, you can compile and execute the OwnCSSExample.java file from the command prompt by using the following commands:

```
javac OwnCSSExample.java
java OwnCSSExample
```

The output of the above example is shown in figure 14.3 like the following:

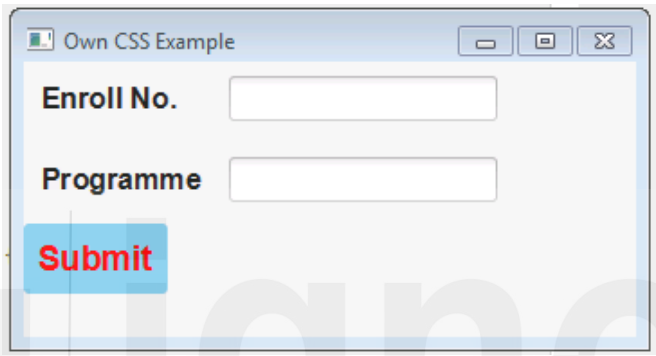

Figure 14.3: Output Screen for CSS creation example

Create Inline Style Sheets

JavaFX allows us to describe the style rules in the JavaFX application code itself. You can insert inline styles in JavaFX applications by using setStyle() method. The inline style provides the finest level of control. These styles are applicable only for those nodes on which they are set. The following example shows you how to create an inline style sheet to a button.

```
Button b1 = new Button("Submit");
b1.setStyle("-fx-background-color:yellow; -fx-text-fill: white;");
```

The following example illustrates inline style sheet rules in the source code file for some UI controls in JavaFX such as text field, password field and button. This example contains only one java file namely CssInStyleExample.java. The source code is listed below:

Example-2: Source code for CssInStyleExample.java

```
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.*;
import javafx.scene.text.*;
import javafx.scene.*;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.stage.Stage;

public class CssInStyleExample extends Application
{
 public static void main(String args[])
 {
   launch(args);
 }
```

```
public void start(Stage stage)
{
 Text txt1 = new Text("Name");     // label for Student Name
 Text txt2 = new Text("Password");     // label for Password
 Text txt3 = new Text("Programme");// label for Programme

 // Text Field for Student
 TextField txtField1 = new TextField();

 // Text Field for password
 PasswordField txtField2 = new PasswordField();

 // Text Field for programme
 TextField txtField3 = new TextField();

 //Creating Buttons
 Button bt1 = new Button("Submit");
 Button bt2 = new Button("Reset");

 //Creating a Grid Pane
 GridPane gPane = new GridPane();

 //Set size for the pane
 gPane.setMinSize(400, 200);

 //Set the padding
 gPane.setPadding(new Insets(10, 10, 10, 10));

 //Set the vertical and horizontal gaps between the columns
 gPane.setVgap(5);
 gPane.setHgap(5);

 //Set the Grid alignment
 gPane.setAlignment(Pos.CENTER);

 //arrange all the nodes in the grid
 gPane.add(txt1, 0, 0);
 gPane.add(txtField1, 1, 0);
 gPane.add(txt2, 0, 1);
 gPane.add(txtField2, 1, 1);
 gPane.add(txt3, 0, 2);
 gPane.add(txtField3, 1, 2);
 gPane.add(bt1, 0, 3);
 gPane.add(bt2, 1, 3);

 //inline style for nodes
 bt1.setStyle("-fx-background-color:blue; -fx-text-fill: white;");
 bt2.setStyle("-fx-background-color: blue; -fx-text-fill: white;");
 txt1.setStyle("-fx-font: normal 15px 'arial' ");
 txt2.setStyle("-fx-font: normal 15px 'arial' ");
 txt3.setStyle("-fx-font: normal 15px 'arial' ");
 gPane.setStyle("-fx-background-color: lightblue;");

 // creating a scene object
 Scene scene = new Scene(gPane);

 // Set Stage title
 stage.setTitle("CSS InStyle Example");
```

```
        // add scene to the stage
        stage.setScene(scene);

        //showing the contents of the stage
        stage.show();
    }
}
```

Now, you can compile and execute the CssInStyleExample.java file from the command prompt by using the following commands.

javac CssInStyleExample.java

java CssInStyleExample

The output of the above example is shown in following figure 14.4:

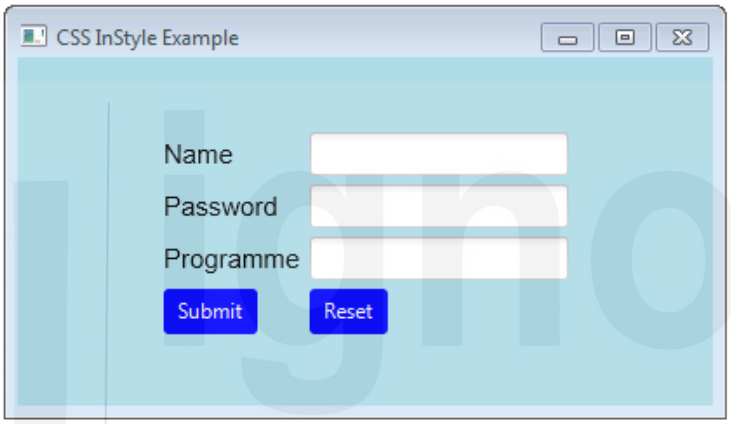

Figure 14.4: Output Screen for inline CSS creation example

14.3 BUILD UI WITH FXML

In the previous section, you have learned about the creation of cascading style sheets (CSS) and linking with JavaFX UI controls. This section describes to you how to create JavaFX user interfacesusing FXML.

FXML is a XML-based user interface markup language for defining the user interface of a JavaFX application. It is created by Oracle Corporation and is another form of designing user interfaces by means of procedural code. Using FXML, you can write the UI controls code separate from the application logic.

JavaFX FXML Example

You can write a very simpleway to JavaFX FXML. Here is a FXML example that comprises a simple JavaFX Graphical User Interface.

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.control.Label?>

<VBox>
<children>
<Label text="Welcome Students in the FXML world "/>
</children>
</VBox>
```

The first line in the FXML document is similar to standard first line of XML documents. The next two lines are import statements which you want to import the classes to use in FXML. After the import statements you shall write the actual structure of the GUI. This example defines a VBox JavaFX layout component which contains a single Label as child element. The Label is used todisplay a text in the Graphical User Interface.

For the understanding of the concept of FXML through the below example which comprises three files:

- ApplicationFXML.java- It is the main JavaFX Application class which loads the FXML document using FXMLLoader.
- ControllerFXML.java – This file contains event handler code.
- FXMLfile.fxml - This FXML Document contains user interface for the application

Example - 3: Source code for ApplicationFXML.java

```
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class ApplicationFXML extends Application
{
   public void start(Stage stage) throws Exception
   {
      Parent root = FXMLLoader.load(getClass().getResource("FXMLfile.fxml"));
      Scene sc = new Scene(root);
      stage.setScene(sc);

      // Set the Stage Title
      stage.setTitle("FXML Example");
      stage.show();
   }
   public static void main(String[] args)
   {
      launch(args);
   }
}
```

Source code for FXMLfile.fxml

```
<?xml version="1.0" encoding="UTF-8"?>
<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane id="AnchorPane" prefHeight="200" prefWidth="320"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="ControllerFXML">

<children>
<Button layoutX="126" layoutY="90" text="Click here!"
                onAction="#ButtonEvent" fx:id="button" />
```

```
<Label layoutX="126" layoutY="120" minHeight="16" minWidth="69"
            fx:id="label" />
</children>
</AnchorPane>
```

Source code for ControllerFXML.java

```
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Label;

public class ControllerFXML
{
  @FXML
   private Label label;
   public void initialize() {   }
  @FXML
   private void ButtonEvent(ActionEvent event)
   {
label.setText("Welcome in SOCIS!");
   }
}
```

Now, you can compile both java files. When you run this project, it shall show a small window with a "Click here" button. After clicking on the button, it will show a welcome message.

When your main application class named as ApplicationFXML will run, the FXMLLoader will load FXMLfile.fxml. During loading FXMLfile.fxml, loader will find the name of the controller class which is specified by fx:controller="ControllerFXML" in the FXML file. Then it will call the controller's initialize method, and the code that is registered as an action handler with the button will be executed.

☞Check Your Progress-1

1) What is JavaFX CSS? What are different ways to apply styles to a javaFX application? Explain with examples.

   ……..

   ……..

   ……..

2) Explain the differences between CSS class Selector and ID styles.

   ……..

   ……..

   ……..

3) In JavaFX application, a key concept is the scene graph. Explain it with diagram.

   ……..

   ……..

   ……..

4) What is FXML? How to use FXML in JavaFX application, explain it with example.

……..

……..

……..

14.4 EVENT HANDLING IN JAVAFX

In the previous section 14.2, you have used UI control elements with CSS in JavaFX applications. This section describes events and the handling of events forthese UI elements.

JavaFX API facilitates us to develop GUI based applications. Whenever you interact with such applications (nodes), an event is generated. For example, when a user clicks on a button, selects an item from a menu, then an event generates. You can say thatevents are notifications. In JavaFX, events are principally used to notify the application about the actions taken by the user.When you write GUI applications, then the program requires event handling mechanism for controlling the user's interaction with the GUI.

JavaFX provides a javafx.event package, which makes available for the basic framework of FX events. The Event class works as the base class for JavaFX events. Each event is associated with source, event target or listener and event type. In event handling mechanism, a source generates an event and sends it to one or more listeners. Once the listeners received this event, they process the event and then returns.

Event source is an object (e.g., a Button object) that generatesan event (e.g., MouseEvent). Event is generated when the changes occurred in the internal state of that object. The source may create various types of events. The origin of the event implements the EventTarget interface.

Listener or target object that listens for a particular type of event which is generated by the source. For the communication between listener and source, the listener must have been connected with the source and must implement methods to receive and process an interface like EventHandler.

There are various **eventstypes** in JavaFX such as ActionEvent (e.g. button is pressed), MouseEvent (e.g. mouse activity), KeyEvent (e.g. keystroke has occurred), ScrollEvent (e.g. scrolling by mouse wheel) etc. You can define your own event by inheriting the class javafx.event.Event. These events are mainly categorised into two groups as Foreground Events and Background Events. **Foreground Events**are based on a user's direct interaction (e.g. button is pressed), while **Background Events** require end-user interaction such as hardware or software failure, timer expiry, etc. For more details, you can refer to the https://docs.oracle.com/javase/8/javafx/events-tutorial/processing.htm

Stages of Event Handling

When a source generates an event, the JavaFX application goes through the four stages (Target selection, Route construction, Event capturing, and Event bubbling) for event handling. Before knowing the stages of event handling, you should know two important terms; Event Filters and Event Handlers.

Event Filter

Event filter covers application logic to process an event. It can be used to process the events such as mouse actions, keyboard actions, scroll actions and other user interactions which are generated by your application. Event filters allow you to handle an event during the event capturing stage of event handling process. During the event capturing stage, a node must be registered with an event filter to process an event.

Event Handler

An event handler is a function or method executed in response to an event, such as keystrokes and button clicks. It is used during the bubbling stage of event handling process. Node which is participated in the dispatch chain path can register to more than one filter/handler. You can define a common filter/handler to the parent, which is treated as a default for all the child nodes.

The following are stages of event handling:

Target selection

When any user interacts with UI controls, then an action occurs. At that moment,the system decides which node gets focus. For example, you click somewhere on the screenfor themouse event, the target node is found at the location of the cursor. If more than one node is found at the cursor point, thenthe uppermost node will be considered as the target.

Route construction

Whenever an event is generated, anevent dispatch route is created in order to handle the events.

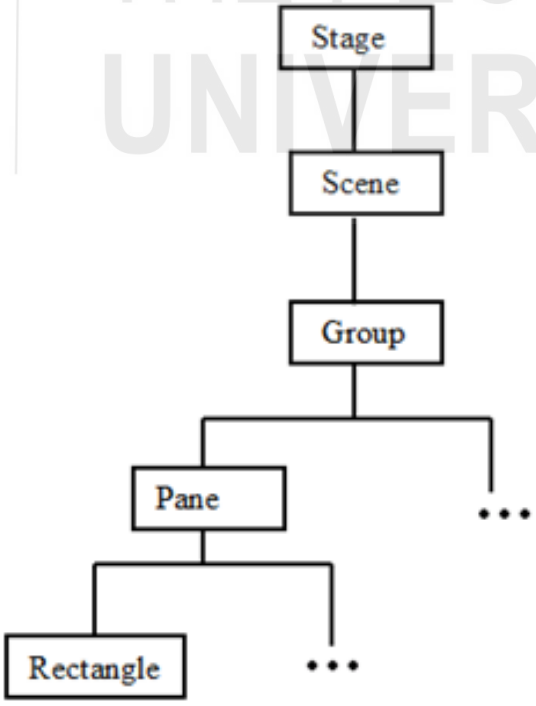

Figure 14.5: Event Dispatch Route

This dispatch route comprises the path from the stage to the node on which the event is generated.The Event route is regulated by the event dispatch chain which was

formed in the operation of the buildEventDispatchChain() method of the selected event target.

Event capturing

Once the event dispatch route is created, the event is transmitted from the source node. The generated event is traversed through all the nodes in the route from top to bottom. If the event filter is registered with any of these nodes which are in the dispatch route, then event will be executed otherwise, it will be transferred to the target node. The target node processes the event in that case. You can see in event dispatch route, which are shown in Figure 14.5, the event moves from the Stage node to the Rectangle node during the event capturing phase.

Event bubbling

When the event is captured and processed by the target node or registered filter, the event starts navigation of all the nodes again from the bottom to the root node. If any of the node in the despatch chain has handler which is registered for the type of event happened then that handler is called,and it will get executed otherwise the process is returned to the next node up in the route and the process is repeated.If a handler does not consume the event then the root node ultimately receives such event, and processing is finished.

You will find the example on event handling with animation feature in the next section.

14.5 EFFECTS, ANIMATION AND MEDIA

In the previous section, you have known about the event handling in javaFX application. This section describes how to give effects and animations on the UI controls as well as about the integration of media in your application.

Effects in JavaFX

You may be aware of the creation of the effects on graphics which are any action or changes that heightens the presence of the graphics in an application. The meaning of effects is functionally similar to the JavaFX application, where in it is an algorithm that applies on nodes to visually enrich their appearance. The JavaFX API provides effect property of the **Node** class to specify the effect which is used to set various effects such as **Blend, Bloom, BoxBlur, ColorAdjust, ColorInput, DropShadow, GaussianBlur,Glow, ImageInput, InnerShadow, Lighting, MotionBlur, PerspectiveTransform, Reflection, SepiaTone, and Shadow** to a node. Each of these effects isdenoted by a class, and all these classes are accessibleby a package named **javafx.scene.effect.Effect**.This section describes the effects of using two DropShadow and Reflection with examples. Here is not possible to define each effect with examples.

Apply Effect to JavaFX Node

You can apply effects on any JavaFX UI control using setEffect() method, which needs to be called through a node object. The following example is showing DropShadow effect on Rectangle shape. Here, color code (FFFF00) representing the yellow color. JavaFX provides javafx.scene.effect. DropShadow class is used for providing dropshadow effect on UI controls. The parameters of DropShadow class are Radius, X Offset, Y Offset and Color like the following. These parameters are optional.

Rectangle.setEffect(new DropShadow(1, 20, 30, Color.web("#FFFF00")));

DropShadow Effects

The following example-4 demonstrates how to apply DropShadow effects to a JavaFX UI control. For this you can create an application which has a Rectangle shape and put DropShadow effecton the shape. The source code is listed below:

Example- 4 : Source code for EffectsJavaFX.java

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.effect.DropShadow;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
public class EffectsJavaFX extends Application
{
  public static void main(String[] args)
  {
    launch(args);
  }
  public void start(Stage stage)
  {
    Rectangle r = new Rectangle(100,100, Color.RED);
    r.setEffect(new DropShadow(1, 20, 30, Color.web("#FFFF00")));
    Scene scene = new Scene(new Pane(r), 300, 250);
    stage.setScene(scene);
    stage.setTitle("Effects Example");
    stage.centerOnScreen();
    stage.show();
  }
}
```

Now, you can compile and execute the above program from the command prompt. The output of the program is display as shown in figure 14.6:

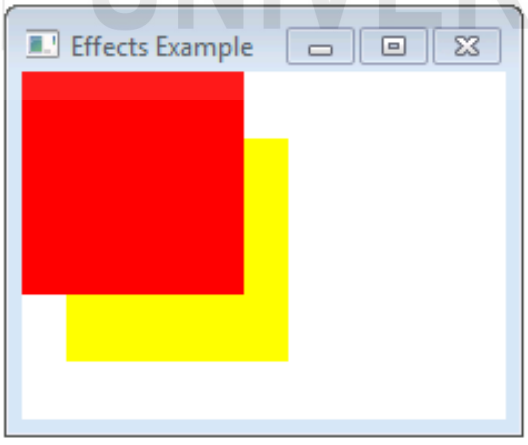

Figure 14.6: Reflection Effects

The JavaFX provides mirror-like reflection effect using javafx.scene.effect. Reflection class. It takes the parameters such as topOffset, topOpacity, bottomOpacity, fraction,which affect resulting reflection to a JavaFX Node.

The following example-5 demonstrates how to apply Reflection effects to a JavaFX UI control.

Example-5:Source code for ReflectionEffects.java

```
import javafx.application.Application;
import javafx.geometry.VPos;
import javafx.scene.Scene;
import javafx.scene.effect.Reflection;
import javafx.scene.layout.Pane;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;
public class ReflectionEffects extends Application
{
  public static void main(String[] args)
  {
    launch(args);
  }
  public void start(Stage stage)
  {
    Text txt = new Text("Working with UI Controls");
    txt.setLayoutX(30);
    txt.setLayoutY(20);
    txt.setTextOrigin(VPos.TOP);
    txt.setFont(Font.font("Arial", FontWeight.BOLD, 40));
    Reflection ref = new Reflection();
    ref.setTopOffset(0);
    ref.setTopOpacity(0.75);
    ref.setBottomOpacity(0.10);
    ref.setFraction(0.7);
    txt.setEffect(ref);
    Scene scene = new Scene(new Pane(txt), 425, 175);
    stage.setScene(scene);
    stage.setTitle("Reflection Effects Example");
    stage.show();
  }
}
```

Now, you can compile and execute the above program from the command prompt. The output of the program is display as shown in figure 14.7:

Figure 14.7: Example for Reflection Effects

Animation in JavaFX

You have seen many animated graphics which have shownthe illusion of its motion using the rapid display.You can create an animated nodein JavaFX application by changing its property over time. You can apply animations or transitions such as **Fade Transition**, **Fill Transition**, **Rotate Transition**, **Scale Transition**, **Stroke Transition**, **Translate Transition**,**Pause Transition**, **Parallel Transition, Path Transition**, **Sequential Transition** etc. All these transitions are represented by individual classes in the package **javafx.animation**.

Translate transition is used to create movement/transition from one point to another point within a defined duration. It is done by keep changing the translateX and translateY properties of the node at the regular interval. In JavaFX, this animation is represented by the class javafx.animation.TranslateTransition. The example-6 is showing Translate transition feature of animation. Using **Rotate transition** feature**,** you can rotate an object. You can set rotation using 'toAngle'and 'byAngle' method. The 'toAngle' indicates what angle the node should rotate, and 'byAngle' indicates how much it should rotate from the current angle of rotation. For more details and example, you can refer to the Oracle JavaFX document at the link (https://docs.oracle.com/javafx/2/animations/basics.htm).

To incorporate animations in JavaFX application, you may follow the below steps

- Create a node using their class.
- Instantiate the animationclass that is to be used
- Set the properties of the animation
- To complete this application, play the animation using the play() method of the Animation class.

The following example illustrates how to apply '**Translate Transition**' animation on a Rectangle node with event handling. For this you can create an application which has a Rectangle shape and two Play and Pause buttons for change the running position of the shape. The source code is listed below:

Example-6:Source code for AnimationNEventHandler.java

```
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import javafx.event.EventHandler;
import javafx.scene.*;
import javafx.scene.control.Button;
import javafx.scene.input.MouseEvent;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.util.Duration;
public class AnimationNEventHandler extends Application
{
  public static void main(String[] args)
  {
    launch(args);
  }
  public void start(Stage stage) throws Exception
  {

    Rectangle r = new Rectangle(100,100);  //Create Rectangle
    r.setFill(Color.BLUE);  //set the color of  Rectangle

    //create play button and set coordinates
    Button btn1 = new Button("Play");
    btn1.setTranslateX(100);
    btn1.setTranslateY(150);

    // creating pause button and set coordinate
    Button btn2 = new Button("Pause");
    btn2.setTranslateX(145);
    btn2.setTranslateY(150);

    //Instantiating TranslateTransition class to create animation
    TranslateTransition trans = new TranslateTransition();
```

```
    //set attributes for the TranslateTransition
    trans.setAutoReverse(true);
    trans.setByX(200);
    trans.setCycleCount(100);
    trans.setDuration(Duration.millis(600));
    trans.setNode(r);

    //Create EventHandler
    EventHandler<MouseEvent> handler = new EventHandler<MouseEvent>()
    {
      public void handle(MouseEvent event)
      {
        if(event.getSource()==btn1)
        {
          trans.play(); //animation will play when click on the play button
        }
        if(event.getSource()==btn2)
        {
          trans.pause(); //animation will pause when click on the pause button
        }
        event.consume();
      }

    };

    //Add Handler for the play and pause button
    btn1.setOnMouseClicked(handler);
    btn2.setOnMouseClicked(handler);

    //Create Group and scene
    Group root = new Group();
    root.getChildren().addAll(r,btn1,btn2);
    Scene scene = new Scene(root,200,100,Color.TRANSPARENT);
    stage.setScene(scene);
    stage.setTitle("Animation with Event Handling");
    stage.centerOnScreen();
    stage.show();
  }
}
```

Now, you can compile and execute the above program from the command prompt. The output of the program is display as shown in figure 14.8.

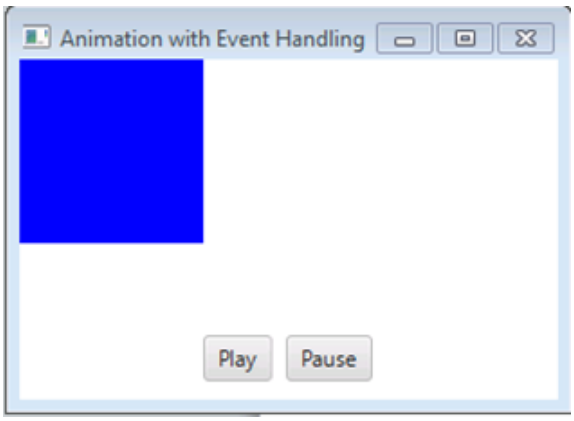

Figure 14.8: Output screen for the example of event handling with animation

Media in JavaFX

You have seen many rich internet applications (RIA) on the internet or even used such applications like GoogleMaps and Google Docs. JavaFX provides media API that enables you to integrate audio and video into rich internet applications. The package **javafx.scene.media** covers all the essential classes for making an attractive, interactive JavaFX application.It consists of three prime classes such as **Media, MediaPlayer** and **MediaView**. The package is used for media playback.

The Media class denotes a media resource which could be an audio or a video, whereas MediaPlayer class is used for playing media. The MediaPlayer class is used in association with the Media and MediaView classes to display and control media playback. MediaPlayer provides media-controlling methods such as pause(), play(), stop() and seek()which apply to all types of media. MediaView arrangesa view of Media being played by a MediaPlayer.

It is very easy to incorporate video or audio in JavaFX application. For integrating media in JavaFX, you can use the following steps:

- You can instantiate **javafx.scene.media.Media** class by passing the path of the video or audio file in its constructor like the following:

```
Media media = new Media("source of video/audio");
```

- Now, you can pass the media class object to the new instance of javafx.scene.media.MediaPlayer object like the following:

```
Mediaplayer player = new MediaPlayer(media);
```

- Invoke the MediaPlayer object's by play() method

```
player.play();
```

- You can instantiate MediaView class and pass player object into its constructor like the following:

```
MediaView mediaView = new MediaView (mediaPlayer)
```

- Add the MediaView object and configure Scene.

```
Scene scene = new Scene(new Pane(mView), 500, 400);
stage.setScene(scene);
stage.setTitle("Video Example");
stage.show();
```

The following example shows you how to run video file in a javaFX application. Demonstrated Video in this example which is available on the YouTube (https://www.youtube.com/watch?v=CtwJ1Mgf_ic) or you can use your own video for running this example.

Example-7: Source Code for ExampleMedia.java

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;
import javafx.scene.media.MediaView;
```

```
import javafx.stage.Stage;
import java.io.File;
public class ExampleMedia extends Application
{
  public static void main(String[] args)
  {
    launch(args);
  }
  public void start(Stage stage) throws Exception
  {
    File mFile = new File("Computer.MP4");

    //Instantiating Media class
    Media media = new Media(mFile.toURI().toString());

    //Instantiating MediaPlayer class
    MediaPlayer player = new MediaPlayer(media);

    //Instantiating MediaView class
    MediaView mView = new MediaView(player);
    player.play();

     //configure scene
    Scene scene = new Scene(new Pane(mView), 500, 400);
    stage.setScene(scene);  // Setting the scene to stage
    stage.setTitle("Video Example");
    stage.show();
  }
}
```

Now, you can compile and execute the above program from the command prompt. The output of the program is display as shown in figure 14.9.

Figure 14.9: Example for video incorporating in javaFX application

The above example is for video including in JavaFX application. In a similar way, you can include your audio also.

☞Check Your Progress-2

1) Explain the term event. What steps are needed to be followed in order to handle the events?

……

……

……

2) What is the difference between foreground and background events?

……

……

……

3) Give names of animation in JavaFX. Explain any two of them with example.

……

……

……

14.6 SUMMARY

This unit explained about the JavaFX UI Controls, which are used in different perspectives such as creating and linking with CSS, building UI with FXML, event handling, effects, animation and media incorporating with UI. Also, in this unit it is explained how to create and apply CSS on UI control elements with examples. You have also learnt how to build javaFX UI with FXML. The FXML is a XML based language, and it is used for constructing Java object graphs. FXML offers an appropriate alternative to constructing such graphs in procedural code. You also got knowledge about events handling, and it is used when users interact with UI controls.

This unit explained about giving effects and animation to javaFX UI controls elements. At the end of this unit, you have learnt about the media integration in JavaFX application using API that enables you to integrate audio and video into the applications. The package javafx.scene.media covers all the essential classes for making JavaFX applications. This package contains three main classes such as Media, MediaPlayerandMediaView. The package is used for media playback. Now you are able to work with UI controls.

14.7 SOLUTIONS/ANSWERSTO CHECK YOUR PROGRESS

☞ Check Your Progress - 1

1) CSS stands for Cascading Style Sheets which are used for adding style such as fonts, colors, backgrounds and spacing between the paragraphs in HTML documents. It is used to control the presentation of one or more web pages. JavaFX CSS is based on the W3C CSS with some additional features. The objective of JavaFX CSS is to allow developers (who are already at ease with CSS for HTML) to use CSS for customizing JavaFX controls. There are two ways to add CSS to JavaFX application by your own Style Sheet and/or inline Style Sheet.

   You can add your own external style sheet to a scene in JavaFX application like the following:

   Scene sc = new Scene(root, 400, 200);

   sc.getStylesheets().add("path/name_of_stylesheet.css");

Using inline style, you can set CSS styles for a specific element by setting the CSS properties directly on the element by using the setStyle() method, for example, inline style sheet for a button.

```
Button button = new Button("Reset");
button.setStyle("-fx-background-color: #FFbbbb");
```

For more details, you can refer section 14.2 in this Unit.

2) The difference between an ID and a class selector is that an ID is only used to identify a single element in javaFX application whereas class selector is used to identify more than one element. The IDs are only used when one element in the web page should have a particular style applied to it. The name of class selector is preceded by dot(.) character, while the name of ID is preceded by the hash(#) symbol to select the element.

3) A scene graph is a data structure that denotes the contents of a window. The scene graph consists of nodes that show graphical components in the window, such as buttons, radio buttons and checkbox. A scene graph is represented by package javafx.scene in a JavaFX application. The scene class holds all the contents of a scene graph. Some of the nodes behave as containers that contains other nodes. The GUI in a window is created by building a scene graph. For more details, you may refer section 14.2 of this unit.

4) FXML is an XML-based user interface markup language for defining the user interface of a JavaFX application. It is created by Oracle Corporation. It is another form of designing user interfaces using procedural code. You may refer to section 14.3 of this unit for viewing the example.

☞ Check Your Progress - 2

1) An event is an object which describes a state change in a source. It generates whenever a user interacts with UI control elements. It is generated by clicking a mouse, pressing a button, entering a character using the keyboard. It can also generate without human interaction, such as timer expires, a software or hardware failure etc.

The event handling consists of four stages: target selection, route construction, Event capturing, and bubbling. In the target selection phase, when any user interacts with UI controls then an action occurs, at that time the system decides which node gets focus; this happens in target selection phase. Whenever an event is generated, an event dispatch route is created to handle the events in the stage of route construction. This dispatch route comprises the path from the stage to the node on which the event is generated. The event route is regulated by the event dispatch chain which wasformed in the operation of the buildEventDispatchChain() method of the selected event target.

In Event capturing stage, the event is transmitted from the source node. The generated event traversed all the nodes in the route from top to bottom. If the event filter is registered with any of these nodes which are in the dispatch route then event will be executed, otherwise it will be transferred to the target node. The target node processes the event in that case. When the event is captured and processed by the target node or registered filter, the event starts navigationing all the nodes again from the bottom to the root node. If any of the node in the dispatch chain has handler which is registered for the type of event happened then that handler is called, and it will get executed otherwise the process is returned to the next node up in the route and the process is repeated. If a

handler does not consume the event, then the root node ultimately receives such event and processing is finished. These happen in the event bubbling phase.

2) Foreground Events are based on the direct interaction of a user (e.g. button is pressed), while Background Events involve the interaction of end-user such as hardware or software failure, timer expiry. Foreground Events are created by a person directly interacting with the graphical components in a GUI. For example, double-clicking on a button, moving the mouse, input character through a keyboard, selecting an item from the menu, scrolling the page, etc.The hardware or software failure, operation completion interruptions are the example of background events.

3) The names of animations are Fade Transition, Fill Transition, Rotate Transition, Scale Transition, Stroke Transition, Translate Transition, Path Transition, Sequential Transition, Pause Transition, Parallel Transition, etc. All these transitions are represented by individual classes in the package javafx. animation.

Translate transition is used to create movement/transition from one point to another point within a defined duration. It is done by keep changing the translateX and translateY properties of the node at the regular interval. In JavaFX, this animation is represented by the class javafx.animation. TranslateTransition. Using **Rotate transition** feature**,** you can rotate an object. You can set rotation by using 'toAngle' and 'byAngle' method. The 'toAngle' indicates what angle the node should rotate, and 'byAngle' indicates how much it should rotate from the current angle of rotation.

14.8 FURTHER READINGS

- Carl Dea, Gerrit Grunwald, José Pereda, Sean Phillips and Mark Heckler "JavaFX 9 by Example", Apress,2017.
- Sharan, Kishori. Beginning Java 8 APIs, Extensions and Libraries: Swing, JavaFX, JavaScript, JDBC and Network Programming APIs. Apress, 2014.
- https://docs.oracle.com/javase/8/javafx/get-started-tutorial/index.htmlhttps://www.w3.org/Style/CSS/Overview.en.html
- https://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html
- https://docs.oracle.com/javafx/2/css_tutorial/jfxpub-css_tutorial.htm
- https://docs.jboss.org/richfaces/latest_4_5_X/Developer_Guide/en-US/html/chap-Developer_Guide-Skinning_and_theming.html
- https://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm
- https://docs.oracle.com/javafx/2/api/javafx/fxml/doc-files/introduction_to_fxml.html
- https://docs.oracle.com/javafx/2/events/processing.htm
- https://docs.oracle.com/javafx/2/events/filters.htm
- https://docs.oracle.com/javase/8/javafx/api/javafx/scene/effect/Effect.html
- https://docs.oracle.com/javase/8/javafx/get-started-tutorial/animation.htm
- https://docs.oracle.com/javafx/2/api/javafx/scene/media/package-summary.html

UNIT 15 JDBC PART-1

Structure **Page No.**

15.0 Introduction 395
15.1 Objectives 396
15.2 JDBC Introduction 396
15.2.1 Core JDBC components 398
15.3 JDBC Driver 398
15.3.1 Type 1/JDBC-ODBC Bridge 399
15.3.2 Type 2/Java to Native API 399
15.3.3 Type 3/ Java to Network Protocol 399
15.3.4 Type 4/ Java to Database Protocol 394
15.4 JDBC Database Connection Steps 401
15.5 JDBC SQLEXCEPTION 405
15.6 JDBC Driver Manager Class 406
15.7 JDBC Connection Interface 408
15.7.1 Statement Interface 409
15.7.2 PreparedStatement Interface 411
15.7.3 CallableStatement Interface 414
15.8 Summary 418
15.9 Solutions/ Answer to Check Your Progress 418
15.10 References/Further Reading 419

15.0 INTRODUCTION

In this unit, we will discuss the Database Connectivity concept for Java applications. We know, that we can save our data in the file system, but it doesn't offer any capability to querying on data efficiently. At the same time, we know that the various databases like Oracle, MySQL, Sybase etc., provide file-processing capabilities and facilitates efficient query processing. We will explore the Java Database Connectivity (JDBC) Application Program Interface (API) and the interaction with a database using the JDBC feature of Java.

We can write Java applications using JDBC to manage different programming activities, starting from establishing a database connection and then sending a query and retrieving result updates from the database. In general, using JDBC API, we can create the table, update and insert values, fetch records, create prepared statements, perform transactions and catch exceptions.

We can visualize the working of the JDBC API where we are writing a typical Java program and using standard library routines. First, a database connection has been established; subsequently, we use JDBC to query the database. After processing, the result is returned to the application, and finally, we close the connection.

In this unit, we will also explore various JDBC components like Driver, Driver Manager, Connection, Statement etc., which are used for database interaction. A JDBC Driver is used to open the database connection for query execution and receiving results with the Java application. The Driver Manager ensures the transparent access of each database using the correct Driver, which is specific to that database. The Connection interface creates the session between the application and

the database and, finally, the SQL query execution and subsequent fetching of results through some pre-defined libraries stored as Statement object, Prepared Statement object and CallableStatement object.

15.1 OBJECTIVES

After going through this unit, you will be able to:

- write the steps for database connectivity within the Java application using JDBC,
- write Java application to execute SQL query and fetch the results using JDBC API,
- explain various JDBC components required for database interactions,
- use drivers to open the connection for the execution of database commands ,
- use Driver manager, which loads all the system drivers and selects the most appropriate Driver, and
- use Connection interface, which offers different methods for database communication

15.2 JDBC INTRODUCTION

JDBC API provides database connectivity between Java applications and various databases. It defines how to access Relational Database (RDBMS) through standard interfaces and classes. Precisely JDBC API helps us to write Java applications that can access tabular data.

JDBC was created by Sun Microsystems and is SQL-based API to access databases, but now it is own by Oracle. JDBC has been developed as an alternative to the Open Database Connectivity (ODBC), a C-based API created by Microsoft and is used to access databases via SQL.

JDBC API allows the Java application to access various types of databases like Oracle, MS Access, Sybase, MySQL and SQL Server etc. It offers the platform-independent interface between the Java application and the RDBMS and allows the application to execute SQL statements. Various tasks are associated with database usage, such as establishing a connection, query creations, query executions, and viewing and modifying the resulting records. Primarily, JDBC allows mobile access to an underlying database, and for seamless performance, it provides a complete set of interfaces.

To better understand the concept, let us first discuss the Java database connectivity architecture. The JDBC Architecture provisions both 2-tier and 3-tier models but in general, it consists of two layers where through JDBC API, the first layer manages the application-to-JDBC Manager connection and the other layer JDBC Manager-to-Driver Connection.

The two-tier architecture provides direct communication of Java applications with the database where the JDBC driver facilitates communication with a specific database. The Driver receives a request from the application and transforms it into a vendor-specific database call, which is passed to the database. This configuration is termed the client-server architecture, where the client is the user machine, and the database acts as a server. This architecture facilitates the user to send the query to the database, and results are retrieved by the user.

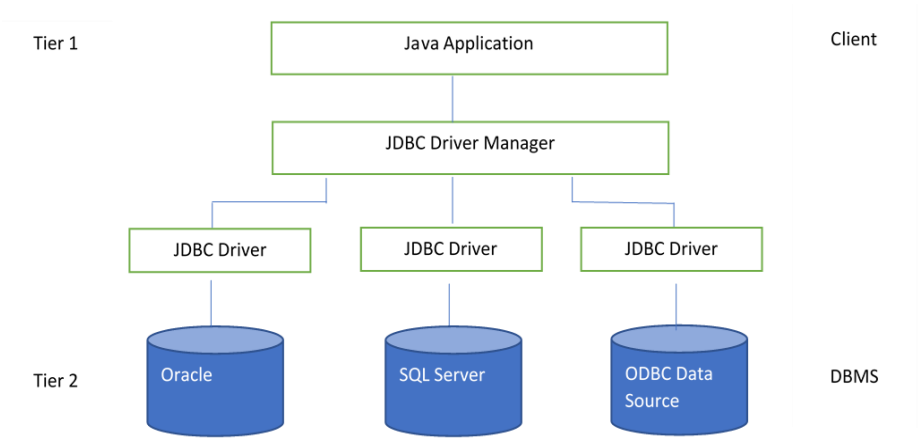

Figure 15.1 Two-tier architecture

In the three-tier architecture, the middle tier is used to take care of the business logic. In this architecture, the user machine sends a command to the middle-tier, which directs it for processing to the database. After processing the commands,database sends back the results to the middle tier, and subsequently, middle-tier sends it to the user machine. The JDBC API is used in the Application business layer of a three-tier architecture.

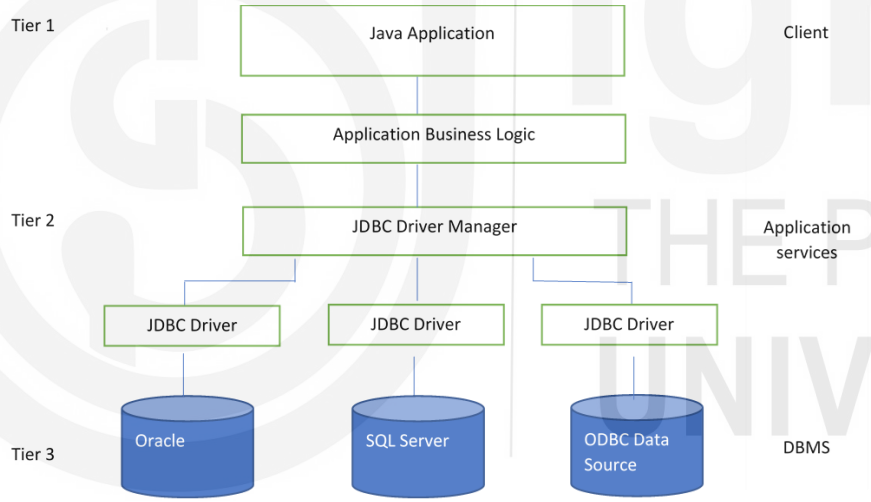

Figure 15.2 Three-tier architecture

The Java application communicates with a database, and the JDBC API facilitates the Java application to execute the SQL queries. The application then retrieves the processed results in a consistent and database independent manner. It uses a driver manager and database-specific Driver and ensures that the driver manager uses the correct Driver to access each database. The JDBC API provides various classes and interfaces that represent objects for this purpose. It includes Connection interface, Statements, PreparedStatement, CallableStatements Interface, ResultSet, Metadata Interface, BLOBs, CLOBs, Database drivers, DriverManager etc.

The JDBC API uses a Driver Manager class to ensure the transparent access of each database. The transparent access to a database by facilitating the correct Driver specific to that database. The JDBC drivers enable the Java applications to make use of different operating systems and hardware platforms.

The API technology provides the standards for Java applications to use the database independently due to its WORA capabilities (“Write Once, Run Anywhere”). Database independence is achieved by using various methods and interfaces that set

up an accessible communication with the database. The application can thus execute queries, retrieve results and update data. JDBC API is part of the Java Standard Edition (SE) and Java Enterprise Edition (EE) platform and provides Java.sql.* and Javax.sql.* packages.

15.2.1 Core JDBC Components

JDBC consists of the following core components through which it can interact with a database:

1. JDBC Driver Manager
2. JDBC Driver
3. Connection Interface
4. Statement Interface
5. ResultSet Interface

JDBC Driver manager: It is the main class that defines an object which is used to connect the application to a JDBC driver. The Driver Manager manages various types of drivers running on an application. The primary responsibility is to correctly load all the system drivers and select the suitable Driver for opening a database connection. There are Standard Extension packages Javax.naming and Javax.sql which are used to create the database connection. We can make use of any package for the connection, but it is recommended to use a Data Source object registered with a Java Naming and Directory Interface™ (JNDI) naming service.

Driver: A JDBC driver opens the database connection for the execution of database commands and receiving results with the Java application. A JDBC driver establishes a connection with a database and subsequently sends an update and retrieves queries. The JDBC API requires drivers for each database and ensures database independence.

Connection Interface: The connection interface offers different methods for database communication,and all the database communication is possible only through the connection object.

Statement Interface: The statement interface object submits a SQL query to the database. It provides two sub-interfaces PreparedStatement and CallableStatement. These interfaces execute queries and fetch the processed result of the executed SQL queries.

ResultSet Interface: The ResultSet interface provides the outcome of a query. The ResultSet object maintains a cursor that points at the current row in the result set data. The result set refers to the row and column data possessed by a ResultSet object of the underlying database.

15.3 JDBC DRIVER

A JDBC driver is a software component that translates standard JDBC calls into a database library API call and enables interaction of the Java application with the database. The JDBC drivers open the database connection to execute database commands and receive results with the Java application. The advantage of using these drivers is to ensure database independence, which implies that we can easily change the backend database by only changing the database driver and a few lines of code.

There are four different types of JDBC drivers having diverse implementations. Let us discuss these drivers:

15.3.1 Type 1/JDBC-ODBC Bridge

The JDBC-ODBC bridge driver acts as a bridge where ODBC drivers are used for accessing JDBC. This bridge driver connects to the database by converting JDBC method calls into ODBC function calls. First, the Java statement transformed into a JDBC statement which subsequently calls the ODBC. This transformation is done by using the Type-I Driver and, finally, the query execution. Oracle does not support this Driver, and this JDBC-ODBC bridge driver is removed from Java 8 onwards.

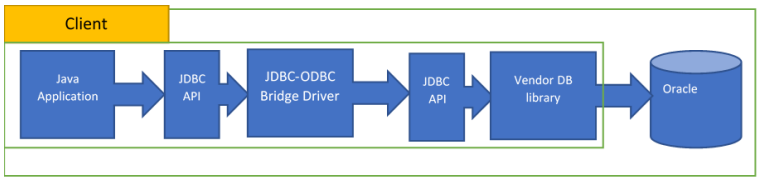

Figure 15.3 JDBC-ODBC bridge driver

Advantage: The JDBC-ODBC Bridge driver was easy to use and offered easy connectivity with any database.

Disadvantage: The Driver was required to install on the client machine.

15.3.2 Type 2/Java to Native API

The Java to Native API driver uses the Java Native Interface (JNI) to make calls to a local database library API. The Type 2 driver connects with the database by converting the JDBC method calls into a native call of the database API. It requires some native code to communicate directly with the database server, but not entirely written in Java.

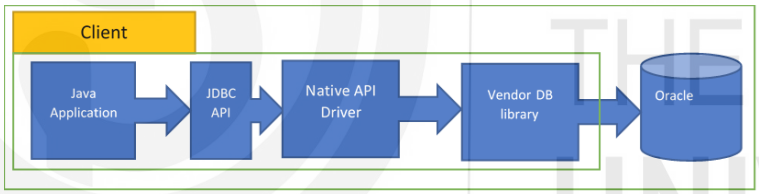

Figure 15.4 Java to Native API driver

Advantage: The Type 2 driver offers better performance than the JDBC-ODBC Driver.

Disadvantage: The native Driver and the vendor library need to install on every client machine. Also, we need to change the native API if we change the database.

15.3.3 Type 3/ Java to Network Protocol

The Network Protocol driver connects with the middleware using a three-tier approach. The Driver connects with the database by converting the JDBC method calls into the vendor-specific database protocol. It is also called an All-Java Driver as entirely written in Java (pure Java drivers).

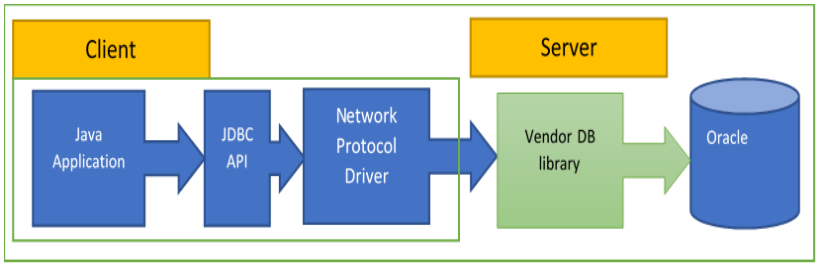

Figure 15.5 Java to Network Protocol driver

Advantage: The Type 3 driver offers a flexible JDBC solution as the native database libraries are not required to install on the client-side. Another advantage of this Driver is that a single driver can be used to access multiple databases.

Disadvantage: The maintenance of the Type 3 driver is costly as the coding at the middleware is database-specific. The server performs various tasks like load balancing, logging etc., but it will require to have network support on the client machine.

15.3.4 Type 4/ Java to Database Protocol

The Thin Driver is the fastest pure Java driver and communicates directly through socket connections without requiring any middleware or native library. The Driver connects with the database by converting the JDBC method calls into the vendor-specific database-protocol directly. This Driver converts the Java statements to SQL statements directly without the need of any intermediate library, and so it is also called a thin driver.

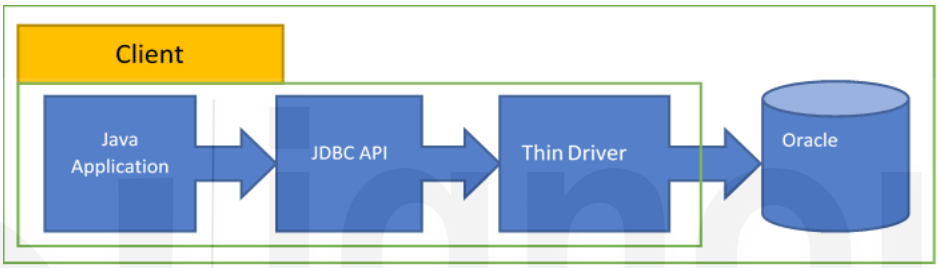

Figure 15.6 Thin deriver

Advantage: These drivers are deployed online without requiring any software installation on the client or server.

Disadvantage: The Type 4 driver is database dependent which means if our backend database changes, we need to deploy a new driver compatible with the database.

Generally, the choice of the Driver depends on many factors like the requirements of the project, cost, flexibility, interoperability, performance etc. Type 1 and Type 2 driver are available mostly free of charge, but they require software installation and software configuration on each client. Type 3 and Type 4 are not open-source. Still, the Type 3 driver offers access to different types of databases, i.e. it is suitable for the application which requires access to multiple databases at the same time. In contrast, Type 4 drivers usually provide better performance.

☞Check Your Progress – 1

1) Briefly explain the necessary steps to create a JDBC application.

……

……

……

……

2) Which JDBC driver is the fastest Driver, and why?

……

……

……

3) How many packages are available in JDBC API?

……

……

……

……

15.4 JDBC DATABASE CONNECTION STEPS

Let us first discuss and explore the various components used in the JDBC Database Connection steps. The JDBC API provides a set of interfaces and classes to establish a connection with the database, create and execute SQL queries, to fetch the results and processing that ResultSet.

JDBC Steps for connecting a Java application with the database can be summarized as follows:

- Load the Driver
- Create a connection
- Create Statement
- Query execution
- ResultSet processing
- Close connection

The JDBC provides driver interface, a Connection interface, Statement interface, and ResultSet interface for the execution.

Step 1: Loading the Driver: The JDBC drivers are acting as a gateway to a database. The first thing we need to do is to initialize a driver before its usage in the program. We need to load the Driver or register the Driver once to open a communication channel with the database. For MYSQL JDBC Driver, we can download MySQL-connector-Java-8.0.21-bin.jar (current), extract the .jar file and add its path into $CLASSPATH.

There are two ways through which we can register the Driver.

1. **DriverManager.registerDriver():** This is an inbuild Java class with register as its static member and will call the driver class constructor at the compile time.

To register the MYSQL Driver, we can use the following syntax

DriverManager.registerDriver(new com.mysql.jdbc.Driver())

2. **Class.forName():** This method will dynamically load the Driver's class into memory at the runtime.

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

Generally, the JVM automatically loads the classes used in the program. But, when the driver class is not used explicitly, we need to explicitly inform the JVM to load the driver class to be used in the program. Since Java version 6 onwards, the JDBC driver is not required to explicitly load; instead, we just need to have the appropriate jar in the classpath. For this reason, that driver loading step is very much non-compulsory from Java 6 onwards. However, it is highly recommended that you check whether automatic loading is available with the Driver used in the application.

Step 2: Create the connections: After we load the Driver, it is required to open a connection which is done by creating a connection object through the static getConnection() method. The connection object establishes a physical connection with the database of the DriverManager class.

Connection jdbcconn = DriverManager.getConnection(URL, user, password)

jdbcconn is a reference to Connection interface, "user" is the username used to access the SQL command prompt, "password" is the password used to access the SQL command prompt, and URL is Uniform Resource Locator. We can define it as follows:

URL = "jdbc:mysql://localhost:3306/emp", user = "root" and password = "admin"

We can connect with the database using the credentials as user "root" with password "admin", and the schema is emp through the port number 3306 of host localhost (or we may use the address of database server).

Step 3: Create a statement: After establishing a connection to interact with the database, the createStatement() method is used to send commands to the database. The Statement object subsequently used to execute the query.

The JDBC Statement used is as follows:

Statement jdbcstmt = jdbcconn.createStatement();

jdbcconn is a reference to the Connection interface

Step 4: Query Execution: The query execution to retrieve the result from the database, we can use the executeQuery() method of the Statement class. There are multiple types of queries in the database like an update, insert or retrieval of data. This executeQuery() method returns the ResultSet object, which is used to fetch the records of a table.

The executeUpdate(SQL query) method of Statement class is used to execute update or insert query. It returns an integer that represents the number of affected rows by the SQL statement.

The JDBC Statement to create a table is as follows:

jdbcstmt.executeUpdate("CREATE TABLE EMP (id INTEGER not NULL, first VARCHAR(30)," last VARCHAR(30), age INTEGER, city VARCHAR(30), salary INTEGER, PRIMARY KEY (id))");

The JDBC Statement to get the data is as follows:

ResultSet jdbcresultset = jdbcstmt.executeQuery("Select * from Emp");

Step 5: ResultSet Processing: After query execution, we have the ResultSet, and the data access through the ResultSet object over a cursor. The cursor is a pointer and initially placed before the first row. Now, using the next() method, it can advance to the subsequent rows of the ResultSet. There are different getter methods of the ResultSet that are available to access the values of the current row, depending on datatypes.

For example, to access the records of all the Employees the ResultSet can be iterated like:

while(jdbcresultset.next())

{

System.out.println("id : " + jdbcresultset.getInt("id") + " First : " + jdbcresultset.getString("first") + " Last : " + jdbcresultset.getString("last") + " Age : " + jdbcresultset.getInt("age") + " City : " + jdbcresultset.getString("city") + " Salary : " + jdbcresultset.getInt("salary"));

}

Step 6: Close the connections: After fetching the records, we will close the connection. The Connection interface close() method used to close the connection and the Statement and ResultSet objects will be closed automatically.

Syntax :

jdbcconn.close();

Let us now take an example, in which we create a JDBC application which establishes a database connection to consolidate our understanding of the concept.

Example: 15.5.1:

```
import Java.sql.*;
import Java.sql.DriverManager;
public class connection
{
 public static void main(String[] args)
 {
  Connection jdbcconn = null;
  try
  {
   //STEP 1: The JDBC driver is registered
   Class.forName("com.mysql.cj.jdbc.Driver");
   // Alternatively,
   DriverManager.registerDriver(newcom.mysql.cj.jdbc.Driver());

   //STEP 2: Open connection
   System.out.println("Connecting to database...");
   jdbcconn =
   DriverManager.getConnection("jdbc:mysql://localhost:3306/emp",
   "root", "admin");

   //STEP 3: Create the Statement object
   Statement jdbcstmt = jdbcconn.createStatement();

   //STEP 4: Execute query to show table
   System.out.println("Executing statement...");
   ResultSet jdbcresultset = jdbcstmt.executeQuery("Select * from
   Emp");

   //STEP 5: The ResultSet is processed
   while(jdbcresultset.next())
   {
    //Retrieve data
    System.out.println("id : " + jdbcresultset.getInt("id") + "
    First : " + jdbcresultset.getString("first")  + " Last : " +
    jdbcresultset.getString("last") +  " Age : " +
    jdbcresultset.getInt("age") + " City : " +
    jdbcresultset.getString("city") + " Salary : " +
    jdbcresultset.getInt("salary"));
   }
   //STEP 6: Clean-up environment
   jdbcconn.close();
```

```
  }
  catch(SQLException sqlex)
  {
   //This will take care of JDBC errors
   sqlex.printStackTrace();
  }
  catch (Exception ex)
  {
   // This will take care of Class.forName errors
   ex.printStackTrace();
  }
  finally
  {
   // This will take care of closing all the resources
   Try
   {
    if (jdbcconn !=null)
    jdbcconn.close();
   }
   catch (SQLException sqlex)
   {
    sqlex.printStackTrace();
   }
   //  end finally
  }
  // end try
  System.out.println("Executed!");
 }
 //end main
}
// end Example
```

Output:

```
Connecting to database...
Executing statement...
id : 501 First : Aman Last : Sharma Age : 40 City : Delhi Salary : 8000
id : 502 First : Ajay Last : Tandon Age : 50 City : Mumbai Salary : 9000
id : 505 First : Suraj Last : Gupta Age : 45 City : Pune Salary : 8000
id : 521 First : Aman Last : Sharma Age : 40 City : Delhi Salary : 8000
id : 522 First : Ajay Last : Tandon Age : 50 City : Mumbai Salary : 9000
id : 525 First : Suraj Last : Gupta Age : 45 City : Pune Salary : 8000

Executed!
```

In this example: first, the import statements are used to import required classes in the code.

In step 1, We registered the JDBC driver (MySQL's Connector/J driver /Type 4 driver), and the Java virtual machine has loaded the desired driver implementation.

In step 2, we have used the getconnection() method. It opens a connection using database URL, username and password.

In step 3, we have created the statement object.

In step 4, we executed the query, where we have accessed an existing database through the connection object, and then data is inserted and accessed through the JDBC driver.

In step 5, we have extracted the data through the result set object and

In step 6, we have closed all the database connections to end the sessions.

15.5 JDBC SQLEXCEPTION

Exception handling is advantageous as it separates the Error-Handling Code from the main logic of the program. In the programming constructs, an exception occurs whenever we come across an abnormal condition, which interrupts the normal flow of the main logic code. The exceptions are handled in a controlled manner, and this stops the normal flow of execution and the control redirects to the possible catch clause. In JDBC, the exception instance generally thrown is SQLException. An SQLException means when a connection object is unable to find an appropriate driver to connect with the database, an exception is thrown. An SQLException is thrown when we encounter an error during database interaction which may be an inappropriate Driver or URL.

The following are the commonly used methods for JDBC SQLException:

getMessage( ): This method is used to get the error message. For database error, it will provide Oracle error number and message, and for the driver error, it will provide JDBC driver error message.

getErrorCode( ): This method is used to get the exception error code, and generally, it is a vendor-specific exception.

setNextException(SQLException sqlex): This method is used to add another SQL exception in the chain.

getNextException( ): This method gets the next exception object from the exception chain.

printStackTrace( ): This method is used to print the current exception, throwable, and backtrace to a standard error stream.

printStackTrace(PrintStream s): This method is used to print the throwable and its backtrace to the specified print stream.

printStackTrace(PrintWriter w): This method is used to print the throwable and backtrace to the specified print writer.

The try-catch block utilized the information from the exception object, and it will catch the exception to continue the program execution.

```
try
{
  // Error prone code
}
catch(Exception ex)
{
  // exception handling code
}
finally
{
  // must be executed code for e.g. jdbcconn.close();
}
```

In the try-catch block, we write the code which has potential chances of failure. If the code fails, the try block should throw an exception object, and the catch block will take care of those exceptions. The exception in the catch block handles the

exception object and may print a statement or perform a pre-defined action to handle the issue. We may use multiple catch blocks, which allows taking more than one exception. The finally-block execute all other necessary components of the program despite the exception. JDBC Exceptions are an efficient way of handling exceptions and should always be used.

Now to consolidate our understanding of Exception handling, let us take the previous example 15.5.1 and pass the incorrect credentials.

In the first case, we pass the wrong driver, and the result will be class not found exception at exceptionhandling.main(exceptionhandling.Java:8)

Output:

15.6 JDBC DRIVERMANAGER CLASS

In the second case, we pass the incorrect database name, and the program will show the unknown database error at exceptionhandling.main(exceptionhandling.Java:13). The outcome of the program as follows:

Output:

```
Connecting to database...
Java.sql.SQLSyntaxErrorException: Unknown database 'emp1'
        at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.Java:120)
        at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.Java:97)
        at
com.mysql.cj.jdbc.exceptions.SQLExceptionsMapping.translateException(SQLExceptionsMappin
g.Java:122)
        at com.mysql.cj.jdbc.ConnectionImpl.createNewIO(ConnectionImpl.Java:836)
        at com.mysql.cj.jdbc.ConnectionImpl.<init>(ConnectionImpl.Java:456)
        at com.mysql.cj.jdbc.ConnectionImpl.getInstance(ConnectionImpl.Java:246)
        at com.mysql.cj.jdbc.NonRegisteringDriver.connect(NonRegisteringDriver.Java:197)
        at Java.sql/Java.sql.DriverManager.getConnection(DriverManager.Java:677)
        at Java.sql/Java.sql.DriverManager.getConnection(DriverManager.Java:228)
        at exceptionhandling.main(exceptionhandling.Java:13)
Executed!
```

The JDBC DriverManager is a concrete Java class that provides an interface between the user and drivers. The other JDBC components are Java interfaces implemented by the various driver packages. The JDBC driver can create the database connections, but generally, the DriverManager is used to get the connection. The JDBC DriverManager provides elementary services to maintain JDBC drivers and defines an object that connects the application to the Driver.

```
Java.lang.ClassNotFoundException: com1.mysql.cj.jdbc.Driver
        at
Java.base/jdk.internal.loader.BuiltinClassLoader.loadClass(BuiltinClassLoader.Java:602)
        at
Java.base/jdk.internal.loader.ClassLoaders$AppClassLoader.loadClass(ClassLoaders.Java:178)
        at Java.base/Java.lang.ClassLoader.loadClass(ClassLoader.Java:522)
        at Java.base/Java.lang.Class.forName0(Native Method)
        at Java.base/Java.lang.Class.forName(Class.Java:340)
        at exceptionhandling.main(exceptionhandling.Java:8)
Executed!
```

The DriverManager manages the various drivers running on an application. It tracks the available drivers and appropriately chooses the one which connects the application with the database.

It also tracks the login time limits of the driver and recording logs and tracing messages. The DriverManager.getConnection() method is the most commonly used method of this class to establish the database connection.

It maintains driver classes registered through the DriverManager.registerDriver() method and a list of all the available drivers preserved with the DriverManager class. The Class.forName() method is used to load the Driver. The DriverManager is being informed by the JDBC drivers whenever their implementation class is loaded.

Syntax:

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

There are many associated methods with this class. Let us summarize a few of the important methods:

registerDriver(Driver driver): This method is used to register the Driver with the DriverManager class.

deregisterDriver(Driver driver): This method is used to deregister or drop from the list of registered drivers in the DriverManager class.

getConnection(String URL): This method is used to establish a connection with the given database URL.

getConnection(String URL, String username, String password): This method is used to establish the connection with the given URL, username and password.

getConnection(String URL, Properties info): This method is used to establish a connection with the given URL.

getDrivers(String URL): This method attempts to locate the Driver by the given string.

getDrivers(): This method retrieves the information of the registered drivers of the DriverManager class.

☞Check Your Progress – 2

1) What are two approaches for registering the Driver?

……

……

……

2) Write down the importance of the JDBC DriverManager class.

……

……

……

3) getConnection()method belongs to which class?

……

……

……

4) Why do we get the following exception Java.sql.SQLException: No suitable driver found?

……

……

5) Can we use multiple catch and finally blocks with a single try statement?

……

……

……

15.7 JDBC CONNECTION INTERFACE

A Connection is an interface used for creating the session between the application and the database. A Java application may require to connect with a single database or sometimes, depending on the application, may require many connections with the different databases. The Connection interface of the Java.sql package characterizes a session with the connected database.

After a connection establishment, the execution of the SQL statements takes place. This process of query execution and result fetching is within the context of the connection. The implementation of these SQL queries requires the usage of some pre-defined libraries stored as Statement object, PreparedStatement object and CallableStatement object. Based on the application requirements, we can use any one of them.

The Connection interface contains Statement interface, PreparedStatement interface and CallableStatement interface. The getConnection() method of the class DriverManager is used to get a Connection object which is subsequently used to get the Statement object of Statement, PreparedStatement and CallableStatement. The connection interface also supports various methods for transaction management, another critical concept while using databases.

The DriverManager.getConnection() method is used to establish a connection and the DriverManager class searches and locates the most appropriate driver from the list of all registered Driver classes that can connect with the database specified in the URL.

syntax:

```
Connection jdbcconn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/emp", "root", "admin");
```

The commonly used methods for the Connection Interface are as follows:

createStatement(): This method creates a statement object for executing the SQL queries.

createStatement(int resultSetType, int resultSetConcurrency): This method creates a statement object for the ResultSet

PreparedStatement(String SQL): This method creates a PreparedStatement object for executing parameterized queries.

getMetaData(): This method returns a Database MetaData object containing the connected database information.

setAutoCommit(boolean status): This method is used to set the commit status, which is by default always true.

commit(): This method saves the changes which have been committed or are permanent rollback.

rollback(): This method drops all changes when commit or permanent rollback

close(): This method closes the connection and immediately releases the JDBC resource.

15.7.1 Statement Interface

The Statement interface is used for the general-purpose access of the database. Itis generally used when working with static SQL queries at runtime. We connect the JDBC to interact with the database, the Java.sql package, the two interfaces Java.sql.Statement and Java.sql.PreparedStatement. which are used for efficiently executing the SQL queries.

For query execution, we may use any interface, but they have a different implementation. The Statement interface is used to execute the static SQL statement, and the PreparedStatement interface is used to execute pre-compiled queries.

The Statement interface provides various methods for query execution. It executes SQL queries using the basic methods like Statement.executeQuery(String) or Statement.executeUpdate(String) for DDL operations. This interface provides the Object of ResultSet. The Connection.createStatement() method is used to get a Statement object.

syntax:

Statement jdbcstmt = jdbcconn.createStatement();

The PreparedStatement and CallableStatement are the two sub-interfaces offered by the Statement interface.

The PreparedStatement object stores the pre-compiled query, which allows multiple runs of the same SQL query with different parameters efficiently. On the other hand, the CallableStatements are used to execute the SQL stored procedure. We will discuss these sub-interfaces in the subsequent section.

The following are the commonly used methods for Statement interface.

ResultSet executeQuery(String SQL): The executeQuery method is used to execute the SELECT statement, and will return the ResultSet object.

int executeUpdate(String SQL): The executeUpdate method is used to execute a specific query like insert, update, delete etc. and will return the number of affected rows.

boolean execute(String SQL): The execute method is used to execute queries that returns a boolean value and will return true if a ResultSet object fetched the result else, it would return false.

executeBatch(): The executeBatch method executes batch statements.

Let us write a simple example of Statement interface to perform various basic operations like insert, update and delete.

Example 15.7.1:

```
import Java.sql.*;
import Java.sql.DriverManager;
class statementInterface
{
  public static void main(String args[])
  {
   Connection jdbcconn = null;
```

```
try
{
 Class.forName("com.mysql.cj.jdbc.Driver");
 jdbcconn =
 DriverManager.getConnection("jdbc:mysql://localhost:3306/emp",
 "root", "admin");
 Statement jdbcstmt = jdbcconn.createStatement();

 // Update the record using execute method
 boolean status = jdbcstmt.execute("Update Emp set age = 40 where
 id =502");
 if(status == false)
 {
  System.out.println("Number of Updated rows " +
  jdbcstmt.getUpdateCount() );
 }
 //  Insert the record using execute method
 int count1 = jdbcstmt.executeUpdate("Insert into Emp(id, First,
 Last, Age, City, Salary) values(512,'Gaurav', 'Mehta', 44,
 'Delhi', 5000)");
 System.out.println("Number of Rows Inserted " + count1);

 // Update the record using executeUpdate method
 int count2 = jdbcstmt.executeUpdate("Update Emp set age = 35
 where id = 521");
 System.out.println("Number of  Updated rows" + count2);

 // Delete the record using executeUpdate method
 //count = jdbcstmt.executeUpdate("Delete from Emp where id =
 41");
 //System.out.println("Number of Rows Deleted " + count);

 // Executing Query using executeQuery method
 ResultSet jdbcresultset = jdbcstmt.executeQuery("Select * from
 Emp");

 // Retrieve data using ResultSet
 while(jdbcresultset.next())
 {
  System.out.println("id : " + jdbcresultset.getInt("id") + "
  First : " + jdbcresultset.getString("first")  + " Last : " +
  jdbcresultset.getString("last") +  " Age : " +
  jdbcresultset.getInt("age") + " City : " +
  jdbcresultset.getString("city") + " Salary : " +
  jdbcresultset.getInt("salary"));
 }
 jdbcconn.close();
}
catch(Exception e)
{
 //Handle errors for Class.forName
 System.out.println(e);
}
//end try
System.out.println("Executed!");
}
//end main
}
// end Example
```

Output:

```
Number of Updated rows 1
Number of Rows Inserted 1
Number of  Updated rows1
id : 501 First : Aman Last : Sharma Age : 40 City : Delhi Salary : 8000
```

```
id : 502 First : Ajay Last : Tandon Age : 40 City : Mumbai Salary : 9000
id : 505 First : Suraj Last : Gupta Age : 45 City : Pune Salary : 8000
id : 512 First : Gaurav Last : Mehta Age : 44 City : Delhi Salary : 5000
id : 521 First : Aman Last : Sharma Age : 35 City : Delhi Salary : 8000
id : 522 First : Ajay Last : Tandon Age : 50 City : Mumbai Salary : 9000
id : 525 First : Suraj Last : Gupta Age : 45 City : Pune Salary : 8000

Executed!
```

15.7.2 PreparedStatement Interface

As we have discussed that for query execution, we can use any of the interfaces Java.sql.Statement and Java.sql.PreparedStatement. The Java.sql.Statement is generally used for simple DDL queries but for better efficiency where repeating SQL queries or parameterization is required, we use the Java.sql.PreparedStatement. PreparedStatement interface inherits the primary statement interface mentioned earlier and offers parameterization, and also it is much safer against SQL injection attacks.

PreparedStatement interface is generally used when we need to execute the SQL statements multiple times, and the interface accepts input parameters at the runtime. The interface corresponds to pre-compiled statements, offers the better performance of the application and will fetch the result faster as the query is compiled only once. These statements are first compiled into the database and subsequently stored as a PreparedStatement object. This object facilitates the SQL statement execution several times.

The PreparedStatement interface, when used for the parameterized queries a question mark (?) symbol as a placeholder known as parameter-marker, is passed for the values. The question mark values are set by the PreparedStatement and values supplied for every parameter before query execution. The advantage of using PreparedStatement is that we can use the same query multiple times with different parameter values.

For example, String SQL="insert into Emp values(?, ?, ?, ?, ?, ?)"

Here, we pass these (?) parameter used as a placeholder in the query and the values set at the runtime by calling the PreparedStatement setter methods. The various setter methods are used to provide values for these placeholders based on the data-types like setInt(), setFloat(), setString() etc. The setInt(int parameterIndex, value) is the standard form of the setter method where “Int” part in setInt is varying based on the datatype float, string etc. and the parameter index position is defined as parameterIndex in the statement. The parameterIndex starts from 1, unlike Java array indices, which starts at 0.

We can use the PreparedStatement method of the Connection Interface to get the PreparedStatement object.

Syntax

PreparedStatement jdbcprepstmt = connection.prepareStatement(SQL);

The PreparedStatement is more efficient as it directly sent the compiled SQL to the database, thus is more efficient for repeated executions.

The following are the commonly used methods for PreparedStatement interface:

setInt(int parameterIndex, int value): The setInt method used for setting Int value as per the parameter index.

setString(int parameterIndex, String value): The setString method is used to set the String value as per the parameter index.

setFloat(int parameterIndex, float value): The setFloat method is used to set the Float value as per the parameter index.

setDouble(int parameterIndex, double value): The setDouble method is used to set the Double value as per the parameter index.

executeUpdate(): The executeUpdate method used for executing a query like create, drop, insert, update, delete etc.

ResultSetexecuteQuery(): This method is used for executing the select query, and it will return an instance of ResultSet.

Let us write a simple example of the parameterized interface to perform the basic operations of insert, update and delete

Example 15.7.2:

```
import Java.sql.*;
import Java.sql.DriverManager;
public class preparedStmt
{
  public static void main(String[] args)
  {
    Connection jdbcconn = null;
    try
    {
      Class.forName("com.mysql.cj.jdbc.Driver");
      System.out.println("Connecting to database...");
      jdbcconn =
      DriverManager.getConnection("jdbc:mysql://localhost:3306/emp",
      "root", "admin");
      preparedStmt jdbcprepstmt  = new preparedStmt();
      jdbcprepstmt.insertEmp(jdbcconn, 621, "Kushagr", "Mahajan", 31,
      "Pune", 6000);
      jdbcprepstmt.updateEmp(jdbcconn, 521, 33);
      jdbcprepstmt.displayEmp(jdbcconn, 502);
      jdbcprepstmt.deleteEmp(jdbcconn, 522);

      jdbcconn.close();
    }
    catch(Exception e)
    {
      //Handle errors for Class.forName
      System.out.println(e);
    }
    //end try
    System.out.println("Executed!");
  }
  //end main
  /*
   * PreparedStatement to Insert the record
   * @parameter jdbcconn
   */
  privatevoid insertEmp(Connection jdbcconn, intid, String first,
  String last, intage, String city, intsalary)
  throws SQLException
  {
    String insertSQL = "insert into Emp values( ?, ?, ?, ?, ?, ?)";
    PreparedStatement jdbcprepstmt  = null;
    try
```

```
  {
    jdbcprepstmt  = jdbcconn.prepareStatement(insertSQL);
    jdbcprepstmt.setInt(1, id);// here 1 is to specify that is it
    the first parameter
    jdbcprepstmt.setString(2, first);
    jdbcprepstmt.setString(3, last);
    jdbcprepstmt.setInt(4, age);
    jdbcprepstmt.setString(5, city);
    jdbcprepstmt.setInt(6, salary);

    int rowcount = jdbcprepstmt.executeUpdate();
    System.out.println("Count of rows inserted " + rowcount);
  }

  finally
  {
    if (jdbcprepstmt  != null)
    {
      jdbcprepstmt.close();
    }
  }
}

/*
 * PreparedStatement to Update the record
 * @parameter jdbcconn, id, age
 */
private void updateEmp(Connection jdbcconn, intid, intage) throws
SQLException
{
  String updateSQL = "Update emp set age = ? where id = ?";
  PreparedStatement jdbcprepstmt  = null;
  try
  {
    jdbcprepstmt  = jdbcconn.prepareStatement(updateSQL);
    jdbcprepstmt.setInt(1, age);
    jdbcprepstmt.setInt(2, id);
    introwcount = jdbcprepstmt.executeUpdate();
    System.out.println("Count of rows updated " + rowcount);
  }
  finally
  {
    if (jdbcprepstmt  != null)
    {
      jdbcprepstmt.close();
    }
  }
}
/*
 * PreparedStatement to Delete the record
 * @parameter jdbcconn, id
 */
private void deleteEmp(Connection jdbcconn, intid) throws
SQLException
{
  String deleteSQL = "Delete from emp where id = ?";
  PreparedStatement jdbcprepstmt  = null;
  try
  {
    jdbcprepstmt  = jdbcconn.prepareStatement(deleteSQL);
    jdbcprepstmt.setInt(1, id);
    introwcount = jdbcprepstmt.executeUpdate();
    System.out.println("Count of rows deleted " + rowcount);
  }
```

```
    finally
    {
      if (jdbcprepstmt  != null)
      {
        jdbcprepstmt.close();
      }
    }
  }
  /*
   * PreparedStatement to Display the record
   * @parameter jdbcconn, id
  */
  private void displayEmp(Connection jdbcconn, intid) throws
  SQLException
  {
    String selectSQL = "Select * from emp where id = ?";
    PreparedStatement jdbcprepstmt  = null;
    try
    {
      jdbcprepstmt  = jdbcconn.prepareStatement(selectSQL);
      jdbcprepstmt .setInt(1, id);
      ResultSet jdbcresultset = jdbcprepstmt.executeQuery();
      while (jdbcresultset.next())
      {
        System.out.println("id : " + jdbcresultset.getInt("id") + "
        First : " + jdbcresultset.getString("first") + " Last : "+
        jdbcresultset.getString("last") + " Age : " +
        jdbcresultset.getInt("age") + " City : " +
        jdbcresultset.getString("city") + " Salary : " +
        jdbcresultset.getInt("salary"));
      }
    }

    finally
    {
      if (jdbcprepstmt  != null)
      {
        jdbcprepstmt.close();
      }
    }
  }
}
```

Output:

```
Connecting to database...
Count of rows inserted 1
Count of rows updated 1
id : 502 First : Ajay Last : Tandon Age : 40 City : Mumbai Salary : 9000
Count of rows deleted 1
Executed!
```

The query execution in the relational database goes through the steps of parsing, compilation, optimization and finally, implementation of an optimized query. The Statement interface is required to go through all the steps, but with the PreparedStatement the first three steps are executed while creating the preparedStatement itself. This makes this execution quicker than Statement.

15.7.3 CallableStatement Interface

We have already seen that we can execute the simple SQL query using Statement interface and pre-compiled parameterized SQL query using PreparedStatement. The

JDBC API also provides a CallableStatement interface used to implement store procedures and an extension of the PreparedStatement.

The Stored procedure is a set of SQL statements that is stored as a group and resided within the database and can be shared by multiple programs. These stored procedures make the performance better because these are pre-compiled. The pre-compiled procedure is executed with just one call to the database server being in the same database server space, so it reduces the network traffic. Further, we can apply any business logic to the database through the concept of stored procedures and functions. For example, we can evaluate the age of the employee using the date of birth. To get the result, we can create a function that can take input as the date of birth and get the output as the age of the employee.

CallableStatement interface is used to call the stored procedures and functions. We can use the prepareCall() method of the Connection interface to get the CallableStatement object.

Syntax

```
CallableStatement jdbccallstmt = connection.prepareCall("{call
PROCEDURE_NAME( ?, ?, ?)}");
```

The symbol '?' is a placeholder they can use to register IN, OUT, and INOUT parameters.

There are three types of parameters used in the stored procedure. The PreparedStatement object generally uses only the IN parameter, but the CallableStatement object can use all three parameters IN, OUT, and INOUT.

The In parameter is used when the SQL statement is created, and the value is unknown, i.e. the IN parameter is used to pass the values to the stored procedure. The OUT parameter value returned after the execution of the query, i.e. OUT parameter used to hold the result returned by the stored procedure. The INOUT parameter provides both input and output values, i.e. IN OUT acts as both IN and OUT parameter. We use the setter method to bind values to IN parameters, and we use the getter method to retrieve values from the OUT parameters.

In our program, we use CallableStatement method registerOutParamter() to register OUT parameter before calling the stored procedure.

The following are the commonly used methods for CallableStatement interface:

execute(): This method is used to execute a query and return a Boolean value. The value is true if a result is a ResultSet object, and it will be false if there are no results or it's an updated count.

executeUpdate(): This method is for DML statements or DDL statements like Insert, Update Create etc.

executeQuery(): This method is for SQL statement that returns ResultSet.

Let us write a simple example of a callable interface. In this example, we are using a stored procedure "employee_details" that receives "id" as the parameter and provides all other details of the employee.

Example 15.7.3:

```
delimiter $$$
DROP PROCEDURE IF EXISTS emp.employee_details $$$
CREATE PROCEDURE emp.employee_details
```

(IN p_id int,
OUT p_first varchar(30), OUT p_last varchar(30), OUT p_age int, OUT p_city varchar(30), OUT p_salary int
)
BEGIN
SELECT id, first, last, age, city, salary
INTO
p_id, p_first, p_last, p_age, p_city, p_salary
from EMP
where id = p_id;
END
$$$

```
import Java.sql.*;
import Java.sql.DriverManager;
public class callableStmt
{
 public static void main(String[] args)
 {
  Connection jdbcconn = null;
  CallableStatement jdbccallstmt = null;
  try
  {
   Class.forName("com.mysql.cj.jdbc.Driver");
   System.out.println("Connecting to database...");
   jdbcconn =
   DriverManager.getConnection("jdbc:mysql://localhost:3306/emp",
   "root", "admin");
   System.out.println("Creating statement...");
   String SQL = "{call employee_details ( ?,?,?,?,?,?)}";
   jdbccallstmt = jdbcconn.prepareCall(SQL);
   //First Bind the IN parameter and after that bind the OUT
   parameter
   int p_id = 521;
   jdbccallstmt.setInt(1, p_id); // This would set ID as 21
   // We need to register the OUT parameters
   jdbccallstmt.registerOutParameter(2, Java.sql.Types.VARCHAR);
   jdbccallstmt.registerOutParameter(3, Java.sql.Types.VARCHAR);
   jdbccallstmt.registerOutParameter(4, Java.sql.Types.BIGINT);
   jdbccallstmt.registerOutParameter(5, Java.sql.Types.VARCHAR);
   jdbccallstmt.registerOutParameter(6, Java.sql.Types.BIGINT);
   // Stored procedure is run using the execute method
   jdbccallstmt .execute();
   //Fetch employee details with getter method
   String p_first = jdbccallstmt.getString(2);
   String p_last = jdbccallstmt.getString(3);
   Integer p_age = jdbccallstmt.getInt(4);
   String p_city = jdbccallstmt.getString(5);
   Integer p_salary = jdbccallstmt.getInt(6);
   System.out.println("Employee Details with ID:" + p_id + " are " +
   p_first + " " + p_last + " " + p_age + " " +  p_city + " " +
   p_salary);
   jdbcconn.close();
  }
  catch (Exception e)
  {
   System.out.println(e);
  }
 }
}
```

Output:

```
Connecting to database...
Creating statement...
Employee Details with ID:521 are Aman Sharma 33 Delhi 8000
```

The JDBC API offers three different interfaces viz. The statement interface executes routine SQL queries, PreparedStatement to execute dynamic or parameterized SQL queries and CallableStatement to implement the stored procedures. The three interfaces significantly differ in terms of functionality and performance. Let us summarize the differences between these three interfaces.

Table 15.1: the differences betweenStatement, PreparedStatement, and Callable Statement interfaces

Statement	PreparedStatement	CallableStatement
executes Normal SQL queries	executes parameterized or dynamic queries	calls the stored procedure
Preferred to use when query execution only once like DDL statements	preferred to use when a query executed multiple times	preferred to use when the stored procedure executed
we cannot pass the parameters	accepts the parameters at run time	accepts at run time and uses three different types of parameters IN, OUT, IN OUT
offers low performance	offers better performance when multiple queries are executed	offers very high performance

☞Check Your Progress 3

1) Why is Statement object required to perform SQL query execution?

……

……

……

2) What is the fundamental difference between the Statement and Prepared Statement interface?

……

……

……

3) What is the return type of execute(), executeQuery() and executeUpdate() and which method executes any kind of SQL statement?

……

……

……

4) What executeUpdate() method of PreparedStatement interface does?

……

……

……

15.8 SUMMARY

In this unit, we have seen how the JDBC API is used for Database Connectivity for Java applications. JDBC API provides various methods and interfaces for accessible communication with the database, which facilitates an application to execute queries, retrieve results and update data. We have discussed the JDBC drivers, which ensure the database independence and only changing the database driver, and a few lines of code allows us to change the backend database. The JDBC Database Connection Steps to understand the process starting from loading a driver, which opens a communication channel with the database to create a connection through DriverManager class—subsequently creating and executing the statements to fetch the result and finally closing the connection.

The various types of JDBC drivers are available, and there are several factors that may be critical in choosing an appropriate driver for the application. The DriverManager class provides vital services to maintain JDBC drivers and defines an object that connects Java applications to a JDBC driver. Registering and deregistering drivers, locating and storing the list of available drivers, and establishing connections are critical tasks handled by the DriverManager class.

The connection interface is used to represent a session with the connected database and offers various objects for efficient database query execution. The Statement interface generally executes a simple query, the PreparedStatement interface is used to manage repeating queries, and the CallableStatement interface is further improvising on implementing stored procedures.

15.9 SOLUTIONS/ANSWERS TO CHECK YOUR PROGRESS

☞Check Your Progress 1

1) Following are the necessary steps to create a JDBC application:

   a. The first step is to import all the required packages which contain the JDBC classes.

   b. The second step is to register the appropriate JDBC driver, which can perform the task of opening the communications channel with the database.

   c. The third step is to use the Driver Manager class getConnection() method, which can open a connection.

   d. The fourth step is to use the object of Statement Interface to execute a query.

   e. The fifth step is to extract the data using the ResultSet method.

   f. The sixth step is to close all the connections and thus clean up the environment.

2) The Thin Driver is the fastest pure Java driver which communicates directly through socket connections. It doesn't require any middleware or native library. It thus connects with the database by converting the JDBC method calls directly into the vendor database protocol without requiring any intermediate library.

3) It provides two packages Java.sql.* and Javax.sql.*

☞Check Your Progress 2

1) The two approaches for registering the Driver are as follows:

   a. The first approach uses the class.forName() method for dynamically loading the Driver's class into the memory, and JVM registers the Driver automatically.

   b. The second approach uses the DriverManager.registerDriver() method, which statically loads the Driver and is generally used in non-JDK compliant JVM.

2) The JDBC DriverManager is a concrete Java class that provides an interface between the user and drivers. The other JDBC components are Java interfaces and are implemented by the various driver packages. The JDBC driver can create the database connections, but generally, the DriverManager is used to get the connection. The JDBC DriverManager provides elementary services to maintain JDBC drivers and defines an object that connects Java applications to a JDBC driver.

3) The getConnection() method is used to establish a connection with the database and is belongs to DriverManager class.

4) The "exception Java.sql.SQLException: No suitable driver found" exception is due to the unformatted SQL URL string.

5) Yes, we can use multiple catch blocks with a single try statement, and also, the try-catch blocks can be nested similar to if-else statements, but there can only be one finally block with a try-catch statement.

☞Check Your Progress 3

1) The execution of the SQL queries requires some pre-defined library defined in the form of Statement object, PreparedStatement object and CallableStatement object. Based on the application requirement, we need to create either Statement, PreparedStatement or CallableStatement object.

2) The fundamental difference between the Statement interface and the PreparedStatement interface is that the PreparedStatement interface uses pre-compiled statements for execution and thus offers better performance and safeguards from SQL injection attacks.

3) The execute() return type is Boolean, executeQuery() return type is ResultSet object and executeUpdate() return type is int. The boolean execute() can execute any kind of SQL statement.

4) This method is used for executing a query like create, drop, insert, update, delete etc.

15.10 REFERENCES/FURTHER READING

- Herbert Schildt "Java The Complete Reference", McGraw-Hill,2017
- Horstmann, Cay S., and Gary Cornell, " *Core Java: Advanced Features*" Vol. 2. Pearson Education, 2013.
- Prasanalakshmi, "*Advanced Java Programming*", CBS Publishers 2015
- Sagayaraj, Denis, Karthik and Gajalakshmi ,"*Java Programming – for Core and Advanced Users*", Universities Press 2018

- Sharan, Kishori, "Beginning Java 8 APIs, Extensions and Libraries: Swing, JavaFX, JavaScript, JDBC and Network Programming APIs", Apress, 2014.
- Parsian, Mahmoud, "*DBC metadata, MySQL, and Oracle recipes: a problem-solution approach"* Apress, 2006.
- https://docs.oracle.com/Javase/tutorial/jdbc/basics/index.html
- https://www.roseindia.net/jdbc/

ignou
THE PEOPLE'S
UNIVERSITY

UNIT 16 JDBC PART-2

Structure **Page No.**

16.0 Introduction 421
16.1 Objectives 422
16.2 JDBC ResultSet Interface 422
16.2.1 ResultSet Types 423
16.2.2 ResultSet Concurrency 423
16.2.3 ResultSet Holdability 426
16.2.4 ResultSet MetaData 428
16.3 JDBC Transactions 430
16.3.1 rollback() and commit() 430
16.3.2 JDBC Transaction Isolations 434
16.3.3 JDBC Savepoints 436
16.4 JDBC Batch Processing 440
16.5 JDBC RowSet Interface 443
16.6 Introduction to Java Data Object (JDO) 446
16.7 Summary 447
16.8 Solutions/ Answer to Check Your Progress 447
16.9 References/Further Reading 450

16.0 INTRODUCTION

In this unit, we will discuss the concept of the ResultSet interface, which contains tabular data of a database query. The ResultSet objects maintain a cursor and associated navigational methods to manage the data. There are three different ResultSet types that control the cursor movement and allow access to the row data through relative and absolute positions. The ResultSet concurrency provides read-only and updatable-mode, which contains access for the modification of the data. The ResultSet holdability ensure the database update within the same context by either closing the instance on commit or holding up the connection to accommodate changes in the same ResultSet instance.

The JDBC transaction will also be discussed in this unit. The transaction concept is a fundamental concept from the batch processing point of view, where the related queries are executed in a batch, and it is essential to execute either all the queries of the set or none. If even one of the queries fail, it may lead the database to the inconsistence state. To consolidate our understanding of the topic, we will also discuss the transaction isolation and save points within the context of the Transaction. Transaction isolation is a critical concept in transaction management to control and satisfy the ACID properties. Also, the idea of save points or checkpoints has been discussed to ensure that in case of any failure, we may not require complete rollback but merely using the checkpointing system, we can keep our time and efforts.

The concept of JDBC RowSet discussed where the RowSet interface provides a wrapper around a ResultSet object and thus provides all the ResultSet capabilities. In addition to that, it also offers support to the JavaBeans component model. The

RowSet interface is worthy for databases where there is no driver support for specific ResultSet properties. At the end of this unit, the idea of Java Data Objects and the advantages over the JDBC API has been explained in brief.

16.1 OBJECTIVES

After going through this unit, you will be able to:

- explain the concept of ResultSet interface,
- understand the basic concepts of Transaction,
- use of commit() and rollback() methods appropriately for any transaction,
- describe the concept of transaction isolation levels for concurrent execution,
- explain the concept of SavePoints,
- write the program using batch processing in JDBC,
- write the program using the RowSet interface to fetch the data in the tabular form, and
- understand the basics of Java Data Objects.

16.2 JDBC RESULTSET INTERFACE

Resultset Interface is used to store and fetch the executed outcome of any SQL statement. The query execution requires the data to be read from the database and subsequently keep the result after processing. The ResultSet acts like the storage; it stores the data or result of the query after the execution of the SQL statements. The java application may use this result after fetching it from the ResultSet.

We can create a ResultSet as

```
Statement jdbcstmt = jdbcconn.createStatement();
ResultSet jdbcresultset = jdbcstmt.executeQuery (“select * from Emp”);
```

The ResultSet contains the tabular data where each record includes equal columns. The data is accessed through the cursor maintained by the ResultSet object, and initially, the cursor is positioned before the first row, and after calling the next() for the first time, it points at the first record. The cursor position is maintained at the current row by the Resultset object, and to access the subsequent rows, either we use next() method or previous() method.

The data in the ResultSet is accessed using the cursor, and the cursor moves to the subsequent rows in the forward direction where it is taken care of by next() method, and it moves to the previous rows through previous() method of the ResultSet object. These methods return true when the next record is available and return false when no more rows are in the ResultSet object that implies the cursor is pointing after the last record. Also, the next() and previous() methods need not be manually closed as they get automatically closed after the Statement object is committed.

We have seen in the previous unit that to access the data using ResultSet we can iterate through the ResultSet.

```
// Retrieve data using ResultSet
while(jdbcresultset.next())
{
System.out.println("id : " + jdbcresultset.getInt("id") + " First : " +
jdbcresultset.getString("first")  + " Last : " + jdbcresultset.getString("last") +  " Age :
" + jdbcresultset.getInt("age") + " City : " + jdbcresultset.getString("city") " Salary :
```

```
" + jdbcresultset.getInt("salary"));
}
```

We can access the data of the record column-wise. For that, we require the getter methods depending on the column's data type we are accessing like for integer datatype, we can call getInt() method or for string data type, we can use getString() method. We can access the data calling the column name or column index.

The ResultSet has three attributes - Type, Concurrency and Holdability, which we generally set while creating Statement or Prepared Statement.

16.2.1 ResultSet Types

The ResultSet offers different Types based on the cursor movement which provide few important characteristics but not all types may be available with all the databases and drivers which means not all methods works with all ResultSet Types. Few of the Navigation methods are first(), last(), next(), previous(), relative(), absolute(), afterLast(), beforeLast() etc. We can check their availability using DatabaseMetaData.supportsResultSetType(int type) method.

The three ResultSet types are TYPE_FORWARD_ONLY, TYPE_SCROLL_ SENSITIVE and TYPE_SCROLL_INSENSITIVE. The TYPE_FORWARD_ONLY is the default ResultSet type, it confirms the movement, or it allows the user to iterate only in the forward direction like first row, second row 2, third row and so on.

TYPE_SCROLL_SENSITIVE and TYPE_SCROLL_INSENSITIVE allow the user to iterate in both directions and also permits to access relative positions concerning the current position, or even we can access an absolute position.

But, the TYPE_SCROLL_SENSITIVE is sensitive enough to reflect any change in the record. At the same time, it is already open, i.e. it provides a dynamic view of the underlying data, whereas TYPE_SCROLL_INSENSITIVE would not reflect any change in the database record with the opened Resultset. An insensitive ResultSet only provides a static view of the underlying data, i.e. columns values of rows. Subsequently, we will require a new ResultSet object to view the database changes.

Let us summarize the three ResultSet types: 1) The ResultSet.TYPE_ FORWARD_ONLY is neither scrollable and nor sensitive. 2) The ResultSet.TYPE_SCROLL_SENSITIVE is scrollable and sensitive to the database changes, and 3) The ResultSet.TYPE_SCROLL_INSENSITIVE is scrollable but not sensitive to the underlying changes to the database.

16.2.2 ResultSet Concurrency

The ResultSet Concurrency evaluates the ResultSet status through the concurrency mode, which provides the two levels. The first mode, where the ResultSet is updatable, i.e. ResultSet.CONCUR_UPDATABLE() and we can update the columns of each row, or we can insert new rows. The other level is the read-only, i.e. ResultSet.CONCUR_READ_ONLY() which is a default ResultSet concurrency. To incorporate the ResultSet Concurrency, we may require to use the database lock mechanisms.

Again, not all the JDBC drivers support the concurrency modes so that we can check the supported mode through DatabaseMetaData.supportsResultSetConcurrency(int concurrency) method.

The ResultSet Concurrency is independent of scrollability, but generally, we need both ResultSet.TYPE_SCROLL_SENSITIVE and ResultSet Concurrency to satisfy so that we can reach to a particular row to update. There are six ResultSet categories in total, which are as follows:

ResultSet.TYPE_FORWARD_ONLY ()/ ResultSet.CONCUR_READ_ONLY()
ResultSet.TYPE_FORWARD_ONLY() / ResultSet.CONCUR_UPDATABLE()
ResultSet.TYPE_SCROLL_SENSITIVE / ResultSet.CONCUR_READ_ONLY()
ResultSet.TYPE_SCROLL_SENSITIVE / ResultSet.CONCUR_UPDATABLE()
ResultSet.TYPE_SCROLL_INSENSITIVE / ResultSet.CONCUR_READ_ONLY()
ResultSet.TYPE_SCROLL_INSENSITIVE / ResultSet.CONCUR_UPDATABLE()

Let us take an example to understand the ResultSet interface where we fetch the last record, all records, absolute row record and relative row record. In this example, when we try to update the record of the last row, we get an error - ResultSet.CONCUR_READ_ONLY(), records are not updatable.

Example 16.2.1

```
import java.sql.*;
import java.sql.DriverManager;
class scrollableRS
{
publicstaticvoid main(String args[])
{
Connection jdbcconn = null;
try
{
Class.forName("com.mysql.cj.jdbc.Driver");
jdbcconn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/emp",
"root", "bpit");

// Scrollable ResultSet but insensitive to any change made by others
// Read-only Resultset, not updatable
Statement jdbcstmt =
jdbcconn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
{
ResultSet jdbcresultset = jdbcstmt.executeQuery("select * from
emp10");

jdbcresultset.last();
System.out.println("--- Show the Last Row ---");
System.out.println(jdbcresultset.getRow() + ": " +
+jdbcresultset.getInt("id") + ", "
+ jdbcresultset.getString("first") + ", " +
jdbcresultset.getString("last") + ", "
+ jdbcresultset.getInt("age") + ", " +
jdbcresultset.getString("city") + ", "
+ jdbcresultset.getInt("salary"));

jdbcresultset.beforeFirst();
System.out.println("--- Show all the Rows ---");
while (jdbcresultset.next())
{
System.out.println(jdbcresultset.getRow() + ": " +
+jdbcresultset.getInt("id") + ", "
+ jdbcresultset.getString("first") + ", " +
jdbcresultset.getString("last") + ", "
+ jdbcresultset.getInt("age") + ", " +
```

```
jdbcresultset.getString("city") + ", "
+ jdbcresultset.getInt("salary"));
}

jdbcresultset.absolute(5); // starting row number is 1
System.out.println("--- Show Absolute Row 5 ---");
System.out.println(jdbcresultset.getRow() + ": " +
+jdbcresultset.getInt("id") + ", "
                                    +
jdbcresultset.getString("first") + ", " +
jdbcresultset.getString("last") + ", "
+ jdbcresultset.getInt("age") + ", " +
jdbcresultset.getString("city") + ", "
+ jdbcresultset.getInt("salary"));

jdbcresultset.relative(-2);
System.out.println("--- Show Relative Row -2 ---");
System.out.println(jdbcresultset.getRow() + ": " +
+jdbcresultset.getInt("id") + ", "
+ jdbcresultset.getString("first") + ", " +
jdbcresultset.getString("last") + ", "
+ jdbcresultset.getInt("age") + ", " +
jdbcresultset.getString("city") + ", "
+ jdbcresultset.getInt("salary"));

// Update a row
jdbcresultset.last();
System.out.println("--- Try to Update a row ---");
System.out.println(jdbcresultset.getRow() + ": " +
+jdbcresultset.getInt("id") + ", "
+ jdbcresultset.getString("first") + ", " +
jdbcresultset.getString("last") + ", "
+ jdbcresultset.getInt("age") + ", " +
jdbcresultset.getString("city") + ", "
+ jdbcresultset.getInt("salary"));
jdbcresultset.updateInt("salary", 9000); // update cells via column
name
jdbcresultset.updateInt("age", 50);
jdbcresultset.updateRow(); // update the row in the data source
System.out.println(jdbcresultset.getRow() + ": " +
+jdbcresultset.getInt("id") + ", "
+ jdbcresultset.getString("first") + ", " +
jdbcresultset.getString("last") + ", "
+ jdbcresultset.getInt("age") + ", " +
jdbcresultset.getString("city") + ", "
+ jdbcresultset.getInt("salary"));

}
jdbcconn.close();
}
catch (Exception e)
{
//Handle errors for Class.forName
System.out.println(e);
} // end try
System.out.println("Executed!");
}// end main

} // end Example
```

Output:

```
--- Show the Last Row ---
8: 621, Kushagr, Mahajan, 31, Pune, 6000
--- Show all the Rows ---
1: 501, Aman, Sharma, 40, Delhi, 8000
2: 502, Ajay, Tandon, 40, Mumbai, 9000
3: 505, Suraj, Gupta, 45, Pune, 8000
4: 509, Gaurav, Pant, 49, Delhi, 5000
5: 512, Gaurav, Mehta, 44, Delhi, 5000
6: 521, Aman, Sharma, 33, Delhi, 8000
7: 525, Raghav, Gupta, 32, Pune, 8000
8: 621, Kushagr, Mahajan, 31, Pune, 6000
--- Show Absolute Row 5 ---
5: 512, Gaurav, Mehta, 44, Delhi, 5000
--- Show Relative Row -2 ---
3: 505, Suraj, Gupta, 45, Pune, 8000
--- Try to Update a row ---
8: 621, Kushagr, Mahajan, 31, Pune, 6000
com.mysql.cj.jdbc.exceptions.NotUpdatable: Result Set not
updatable.This result set must come from a statement that was
created with a result set type of ResultSet.CONCUR_UPDATABLE, the
query must select only one table, can not use functions and must
select all primary keys from that table. See the JDBC 2.1 API
Specification, section 5.6 for more details.
Executed!
```

16.2.3 ResultSet Holdability

The ResultSet Holdability evaluates the ResultSet instance if it has a closed status when the underlying connection commit(). The commit() method is used to ensure the consistent database states. In the case of ResultSet Holdability, there are two types of holdability. The first one is where all the ResultSet instances are closed on commit(), i.e. CLOSE_CURSORS_OVER_COMMIT and the other one where all the ResultSet instances are not closed but kept open to accommodate the updates with the same ResultSet on commit(), i.e. HOLD_CURSORS_OVER_COMMIT.

The ResultSet Interface methods are as follows:

boolean next(): The next() method is used for advancing the cursor to next row.

boolean previous(): The previous() method precedes the cursor to the previous row.

boolean first(): The first() method moves the cursor to the first row.

boolean last(): The last() method moves the cursor to the last row.

boolean absolute(int row): The absolute() method moves the cursor to the specific row.

boolean relative(int row): The relative() method moves the cursor to the relative row number from the current row.

Again, not all the JDBC drivers support the concurrency modes so that we can check the supported mode through a DatabaseMetaData.supportsResultSetHoldability(int holdability) method. The Connection interface getHoldability() method is used to return an integer value 1 or 2. The ResultSet.HOLD_CURSORS_OVER_COMMIT returns the value 1, and the ResultSet.CLOSE_CURSORS_AT_COMMIT returns the value 2.

We can check the Default cursor holdability using DatabaseMeta Data.get ResultSetHoldability() and also check the driver support for HOLD_CURSORS_ OVER_COMMIT and CLOSE_CURSORS_AT_COMMIT using DatabaseMeta Data.supportsResultSetHoldability(ResultSet.HOLD_CURSORS_OVER_COMMI)

and DatabaseMetaData.supports ResultSetHoldability (ResultSet. CLOSE_ CURSORS_AT_COMMIT) respectively.

Let us take an example where the holdability value is set to HOLD_CURSORS_OVER_COMMIT. In this example, first, we have to set the auto-commit to false and thenretrieve the employee table's content to a ResultSet object and insert a new row in the ResutlSet and to the employee table. At the end, we will commit the execution, but due to the holdability property HOLD_CURSORS_OVER_COMMIT, we can check that the ResultSet object is open.

Example 16.2.3:

```
import java.sql.*;
import java.sql.DriverManager;
class holdabilityRS
{
 publicstaticvoid main(String args[])
 {
  Connection jdbcconn = null;
  try
  {
   Class.forName("com.mysql.cj.jdbc.Driver");
   jdbcconn =
   DriverManager.getConnection("jdbc:mysql://localhost:3306/emp",
   "root", "admin");
   // By default auto commit is true so setting the auto commit
   false
   jdbcconn.setAutoCommit(false);
   // Holdability is set to HOLD_CURSORS_OVER_COMMIT
   jdbcconn.setHoldability(ResultSet.HOLD_CURSORS_OVER_COMMIT);
   // Creating a Statement object
   Statement jdbcstmt =
   jdbcconn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
   ResultSet.CONCUR_UPDATABLE);
   // Execute query to fetch the data
   ResultSet jdbcresultset = jdbcstmt.executeQuery("select * from
   emp10");
   System.out.println("Employee table");
   while (jdbcresultset.next())
   {
    System.out.print("ID: " + jdbcresultset.getInt("id") + ", ");
    System.out.print("First_Name: " +
    jdbcresultset.getString("first") + ", ");
    System.out.print("Last_Name: " +
    jdbcresultset.getString("last"));
    System.out.print("Age: " + jdbcresultset.getInt("age") + ", ");
    System.out.print("City: " + jdbcresultset.getString("city") + ",
    ");
    System.out.print("Salary: " + jdbcresultset.getInt("salary"));
    System.out.println("");
   }
   // Insert new row
   jdbcresultset.moveToInsertRow();
   jdbcresultset.updateInt(1, 622);
   jdbcresultset.updateString(2, "Poonam");
   jdbcresultset.updateString(3, "Sharma");
   jdbcresultset.updateInt(4, 34);
   jdbcresultset.updateString(5, "Delhi");
   jdbcresultset.updateInt(6, 6000);
   jdbcresultset.insertRow();
   // commit transaction
   jdbcconn.commit();
```

```
   booleanstatus = jdbcresultset.isClosed();
   if (status)
   {
    System.out.println("ResultSet object close");
   }
   else
   {
    System.out.println("ResultSet object open");
   }
  }
  catch (Exceptione)
  {
   //Handle errors for Class.forName
   System.out.println(e);
  }
  // end try
  System.out.println("Executed!");
 }
 // end main
}
// end Example
```

Output:

```
Employee table
ID: 501, First_Name: Aman, Last_Name: SharmaAge: 40, City: Delhi,
Salary: 8000
ID: 502, First_Name: Ajay, Last_Name: TandonAge: 40, City: Mumbai,
Salary: 9000
ID: 505, First_Name: Suraj, Last_Name: GuptaAge: 45, City: Pune,
Salary: 8000
ID: 509, First_Name: Gaurav, Last_Name: MehtaAge: 44, City: Delhi,
Salary: 5000
ID: 512, First_Name: Gaurav, Last_Name: MehtaAge: 44, City: Delhi,
Salary: 5000
ID: 521, First_Name: Aman, Last_Name: SharmaAge: 33, City: Delhi,
Salary: 8000
ID: 525, First_Name: Suraj, Last_Name: GuptaAge: 45, City: Pune,
Salary: 8000
ID: 621, First_Name: Kushagr, Last_Name: MahajanAge: 31, City: Pune,
Salary: 6000
ResultSet object open
Executed!
```

16.2.4 ResultSet MetaData

ResultSetMetaData interface provides data about the data available in the ResultSet. It includes all the column and their characteristics. We can get the metadata object through the method from the ResultSet object.

syntax:

public ResultSetMetaData getMetaData()

The ResultSetMetaData interface supports the following methods:

ResultSetMetaData.getColumnCount(): This method is used to fetch the number of columns in the ResultSet object.

ResultSetMetaData.getColumnName(int indexno): This method is used to fetch the name of the columns from the ResultSet object.

ResultSetMetaData.getColumnTypeName (int indexno): This method is used to get the column datatype from the ResultSet object.

ResultSetMetaData getColumnDisplaySize (int indexno): This method is used to get the column size from the ResultSet object.

Let us take an example to understand the concept of ResultSetMetaData:

Example 16.2.4:

```
import java.sql.*;
import java.sql.DriverManager;
public class resultMeta
{
 public static void main(String[] args)
 {
  Connection jdbcconn = null;
  try
  {
   Class.forName("com.mysql.cj.jdbc.Driver");
   jdbcconn =
   DriverManager.getConnection("jdbc:mysql://localhost:3306/emp",
   "root", "admin");
   Statement jdbcstmt = jdbcconn.createStatement();

   ResultSet jdbcresultset = jdbcstmt.executeQuery("Select * from
   Emp");
   ResultSetMetaData jdbcrsmd = jdbcresultset.getMetaData();

   int countno = jdbcrsmd.getColumnCount();
   System.out.println("Total number of columns are " + countno);
   // Display column name, data type and the column size
   for (inti = 1; i<= countno; i++)
   {
    System.out.println(jdbcrsmd.getColumnName(i) + " " +
    jdbcrsmd.getColumnTypeName(i) + " "
    + jdbcrsmd.getColumnDisplaySize(i));

    System.out.println();
   }
   jdbcconn.close();
  }
  catch (Exception e)
  {
   System.out.println(e);
  }
  System.out.println("Success!");
 }
}
```

Output:

```
Total number of columns are 6
id INT 10
first VARCHAR 255
last VARCHAR 255
age INT 10
city VARCHAR 255
salary INT 10
Success!
```

☞Check Your Progress - 1

1) What is a ResultSet Object, and how is it used to fetch the data from the database ?

……..

……..

……..

2) Which is the default ResultSet type in the JDBC application ?

……..

……..

……..

3) What is ResultSet holdability in JDBC ?

……..

……..

……..

4) How do you iterate the data in both forward and backward direction?

……..

……..

……..

16.3 JDBC TRANSACTIONS

A transaction is a logical unit that may contain one or more SQL query executed in totality to perform a specific task. Generally, any change to the database state is due to a transaction execution like creating, updating or deleting a tuple from the table. It is essential to run the transactions in a controlled environment to ensure system integrity and consistency.

We can understand the transaction concept by taking a simple example of updating the fund transfer from one account to another. If the balance from one account is deducted but couldn't transfer to another, then we would like the first execution to roll back; otherwise, the amount is lost in cyber-space. This example clarifies that at times we don't like even the first statement should take effect if the second statement didn't execute correctly; else, we will end up with inconsistent data.

The mechanism to ensure that either all dependent queries or batch of queries execute successfully or none is through transactions. In general, a transaction is referred to as a single unit, and the transaction management supports ACID properties – Atomicity, Consistency, Isolation and Durability.

16.3.1 rollback() and commit()

In JDBC by default, after the query the execution result will be automatically saved, i.e. after the connection establishment with the database, the connection remains in an auto-commit mode. The auto-commit mode here leads to each statement being treated as a transaction, and all updates due to query execution are made permanent. However, at times, to improve the efficiency, we may require to group two or more statements, and for that, we need to turn-off the auto-commit mode. This turning-off

of the auto-commit mode is achieved through the setAutoCommit() method of the connection interface.

syntax:

jdbcconn.setAutoCommit(false);

When we disable the auto-commit mode, no SQL query will commit automatically, and we need to explicitly call the commit() method. This explicit call of jdbcconn.commit() method is through the connection object. Also, if any problem occurs during commit, then a simple set of queries will roll back using jdbcconn.rollback() method.

We can summarize the necessary step needed for any transaction management in JDBC as:

1. We need to change the auto-commit option as false for our connection object.
2. When all the statements in a transaction are completed, we use commit() method to make all the changes permanent.
3. When there is some error while executing statements in a transaction, we need rollback all the changes using rollback() method.

Let us take a simple example to understand the concept of transaction commit and rollback. In this example, we consider the transaction case where the amount from one account is transferred to another account. The MYSQL database is used in this example with the schema “transaction” and table name “account”. The two methods used for transferring amounts are debit() to deduct and credit() to add the amount and combinedly form a transaction. If we get some error while executing any one of these methods, there will be rollback through connection.rollback() else if all goes well it will commit using connection.commit(). In this example, account number 309 is not available, so the query debited the amount from account 1 is also rollback.

Example 16.3.1:

```
import java.sql.*;
import java.sql.DriverManager;
public class transaction
{
public static void main(String[] args)
{
transaction jdbctran = new transaction();
jdbctran.transAmount(1, 309, 10000);
}
// Establishing connection using getConnection() Method
public static Connection getConnection()
{
Connection jdbcconn = null;
try
{
Class.forName("com.mysql.cj.jdbc.Driver");
jdbcconn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/transaction
", "root", "admin");
}
catch (ClassNotFoundException e)
{
e.printStackTrace();
}
catch (SQLException e)
{
```

```
e.printStackTrace();
}
return jdbcconn;
}

// Transfer Amount @parameter from_Account, to_Account, value
public void transAmount(intfrom_Account, intto_Account, intvalue)
{
Connection jdbcconn = null;
try
{
jdbcconn = getConnection();
// For Transaction execution we make Auto-commit disable
jdbcconn.setAutoCommit(false);
// Start the transaction
debit(jdbcconn, from_Account, value);// deduct the amount from the
account
credit(jdbcconn, to_Account, value); // add the amount to the
account
// Complete the transaction
jdbcconn.commit(); // Commit the transaction
System.out.println("Transaction Completed successfully");
}
catch (SQLException e)
{
e.printStackTrace();
if (jdbcconn != null)
{
try
{
System.out.println("Transaction Rollback due to error");
jdbcconn.rollback();
}
catch (SQLException e1)
{
e1.printStackTrace();
}
}
// if condition
}
finally
{
if (jdbcconn != null)
{
// close the connection
try
{
jdbcconn.close();
}
catch (SQLException e)
{
e.printStackTrace();
}
}
// if condition
}
// finally
}


private void debit(Connection jdbcconn, intaccountno, int
debit_amount) throws SQLException
{
```

```
String debitSQL = "update account as Tab_Deduct JOIN "
+ " (select accountno, (amount - ?) as balance from account" + "
where accountno = ?) As tab "
+ " ON  Tab_Deduct.accountno = tab.accountno" + " set
Tab_Deduct.amount = tab.balance"
+ " where Tab_Deduct.accountno = ?";

PreparedStatement jdbcprepstmt = null;
try
{
jdbcprepstmt = jdbcconn.prepareStatement(debitSQL);
jdbcprepstmt.setInt(1, debit_amount);
jdbcprepstmt.setInt(2, accountno);
jdbcprepstmt.setInt(3, accountno);
int countno = jdbcprepstmt.executeUpdate();
if (countno == 0)
{
throw new SQLException("Account number not found " + accountno);
}
}
finally
{
if (jdbcprepstmt != null)
{
jdbcprepstmt.close();
}
}
}
private void credit(Connection jdbcconn, intaccountno,
int credit_amount) throws SQLException
{
String creditSQL = "update account as Tab_Credit JOIN "
+ " (select accountno, (amount + ?) as balance from account" + "
where accountno = ?) As tab "
+ " ON  Tab_Credit.accountno = tab.accountno" + " set
Tab_Credit.amount = tab.balance"
+ " where Tab_Credit.accountno = ?";

PreparedStatement jdbcprepstmt = null;
try
{
jdbcprepstmt = jdbcconn.prepareStatement(creditSQL);
jdbcprepstmt.setInt(1, credit_amount);
jdbcprepstmt.setInt(2, accountno);
jdbcprepstmt.setInt(3, accountno);
int countno = jdbcprepstmt.executeUpdate();
if (countno == 0)
{
throw new SQLException("Account number not found " + accountno);
}
}
finally
{
if (jdbcprepstmt != null)
{
jdbcprepstmt.close();
}
}
}
}
```

Output:

```
java.sql.SQLException: Account number not found 309
Transaction Rollback due to error
        at transaction.credit(transaction.java:97)
        at transaction.transAmount(transaction.java:34)
        at transaction.main(transaction.java:8)
```

16.3.2 JDBC Transaction Isolations

The auto-commit disable is essential during the transaction mode as otherwise, we need to hold database locks during the execution of multiple queries and may end up with some sort of deadlock situation. This situation avoided using the concept of transactions, and even transaction management provides some kind of protection for conflicting statements. Generally, the transaction management uses the lock mechanism to deal with the problem of conflicting statements. DBMS uses locks to check transactional access and manage concurrency control by avoiding problems like dirty read etc.

During the concurrent access of the database generally, the dirty read, non-repeatable reads and phantom reads make the database inconsistent. The problem of dirty read occurs when a transaction reads an uncommitted value. The non-repeatable read problem occurs when one Transaction reread the data after an update made by another transaction. The phantom reads problem happens when records are added or removed by another transaction; then, the transaction is executed on the database. These problems are resolved using a locking mechanism by applying locks and is controlled by the JDBC transaction Isolation level.

There are different transaction levels ranging from TRANSACTION_NONE, which does not support transactions to TRANSACTION_SERIALIZABLE, which strictly supports transactions following access rules and handles problems like dirty read non-repeatable reads or phantom reads.

JDBC provides five different transaction-level through Connection interface.

TRANSACTION_NONE: This level does not support Transaction and is denoted by 0.

TRANSACTION_READ_UNCOMMITTED: - This level supports transaction but has no control over dirty read, non-repeatable and phantom reads and is denoted by 1.

TRANSACTION_READ_COMMITTED: - This level supports transaction and prevent dirty read but allows non-repeatable and phantom reads. This level is denoted by 2.

TRANSACTION_REPEATABLE_READ: - This level supports transaction and allows phantom read but prevents dirty and non-repeatable reads. This level is denoted by 4

TRANSACTION_SERIALIZABLE: - This level supports transaction and strictly prevents dirty read, non-repeatable reads or phantom reads. This level is denoted by 8.

Let us take an example to understand the isolation level numbers and set the level for the underlying database.

Example 16.3.2:

```
import java.sql.*;
```

```
public class transIsolation
{
public static void main(String args[])
{
Connection jdbcconn = null;
try
{
Class.forName("com.mysql.cj.jdbc.Driver");
jdbcconn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/transaction
", "root", "admin");
System.out.println("Connection established:- " + jdbcconn);
System.out.println("Transactions levels for the Connection interface
\"TRANSACTION_NONE\":- "
+ Connection.TRANSACTION_NONE);
System.out.println("Transactions levels for the Connection interface
\"TRANSACTION_READ_UNCOMMITTED\":- "
+ Connection.TRANSACTION_READ_UNCOMMITTED);
System.out.println("Transactions levels for the Connection interface
\"TRANSACTION_READ_COMMITTED\":- "
+ Connection.TRANSACTION_READ_COMMITTED);
System.out.println("Transactions levels for the Connection interface
\"TRANSACTION_REPEATABLE_READ\":- "
+ Connection.TRANSACTION_REPEATABLE_READ);
System.out.println("Transactions levels for the Connection interface
\"TRANSACTION_SERIALIZABLE\":- "
+ Connection.TRANSACTION_SERIALIZABLE);
// Setting the transaction isolation level
jdbcconn.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE
);
int setLevel = jdbcconn.getTransactionIsolation();
System.out.println("Set the Transaction isolation level the current
database is:- " + setLevel);
jdbcconn.close();
}
catch (Exception e)
{
System.out.println(e);
}
System.out.println("Success!");
}
}
```

Output:

```
Connection established:- com.mysql.cj.jdbc.ConnectionImpl@49ec71f8
Transactions levels for the Connection interface
"TRANSACTION_NONE":- 0
Transactions levels for the Connection interface
"TRANSACTION_READ_UNCOMMITTED":- 1
Transactions levels for the Connection interface
"TRANSACTION_READ_COMMITTED":- 2
Transactions levels for the Connection interface
"TRANSACTION_REPEATABLE_READ":- 4
Transactions levels for the Connection interface
"TRANSACTION_SERIALIZABLE":- 8
Set the Transaction isolation level the current database is:- 8
Success!
```

Again, not all the JDBC drivers support the transaction isolation levels so that we can check the sustained status through database MetaData.supports ransaction IsolationLevel method. If a driver request of setTransactionIsolation is not supporting a specific isolation level, then the driver can substitute a higher and more

restrictive transaction isolation level. Also, if the substitution to higher transaction level fails, it throws an SQLException.

16.3.3 JDBC Savepoints

The process of rollback is a saviour at times, but it's not very efficient when a transaction contains a large number of queries to rollback. The Savepoint is one option that sets a Savepoint, i.e. checkpoint in the Transaction and ensures that rollback to only the marked checkpoint and not of the complete Transaction. The connection interface in the current Transaction offers two methods to set Savepoint. The setSavepoint() and setSavepoint (String name) method returns the Savepoint object and creates unnamed Savepoint and given name Savepoint respectively. The rollback() method is overloaded and takes the Savepoint argument. The Savepoint is automatically released on commit, but if we need to release it, we can use the releaseSavepoint() method, which takes the parameter Savepoint.

Let us extend our transaction example to understand the concept of Savepoint. In this example, we preserve the transaction data in another table *all_transactions* with the first column as *from_account*, second column as *to_account*, and the transfer value as the third column. The Savepoint is created when the transfer of the amount is completed from one account to another account. Now, suppose we encounter some error while preserving the transaction details in a new table. In that case, the rollback operation will take place till the checkpoint only and the portion of queries before the Savepoint need not to rollback. This process will avoid the rollback of the complete Transaction.

Example 16.3.3:

```
import java.sql.*;
import java.sql.DriverManager;
import java.sql.Savepoint;
public class transSavepoint
{
public static void main(String[] args)
{
transSavepoint jdbctran = new transSavepoint();
jdbctran.transAmount(4, 3, 10000);
}
// Establishing connection using getConnection() Method
public static Connection getConnection()
{
Connection jdbcconn = null;
try
{
Class.forName("com.mysql.cj.jdbc.Driver");
jdbcconn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/transaction
", "root", "admin");
}
catch (ClassNotFoundException e)
{
e.printStackTrace();
}
catch (SQLException e)
{
e.printStackTrace();
}
returnjdbcconn;
}
```

```
// Transfer Amount @parameter from_Account, to_Account, value
public void transAmount(int from_Account, int to_Account, int value)
{
Connection jdbcconn = null;
Savepoint jdbcsp = null;
try
{
jdbcconn = getConnection();
// For Transaction execution we make Auto-commit disable
jdbcconn.setAutoCommit(false);
// Start the transaction
debit(jdbcconn, from_Account, value);// deduct the amount from the
account
credit(jdbcconn, to_Account, value); // add the amount to the
account
// Complete the transaction
// setting-up save point
jdbcsp = jdbcconn.setSavepoint("DebitCreditDoneSP");
// Preserving transaction
preserveTrans(jdbcconn, from_Account, to_Account, value);
jdbcconn.commit(); // Commit the transaction
}
catch (SQLException e)
{
e.printStackTrace();
if (jdbcconn != null)
{
try
{
// complete rollback if no Savepoint
if (jdbcsp == null)
{
System.out.println("Transaction Rollback due to error");
jdbcconn.rollback();
}
else
{
System.out.println("Transaction Rollback to Savepoint");
jdbcconn.rollback(jdbcsp);
// commit the transaction till the Savepoint
jdbcconn.commit();
}
}
catch (SQLException e1)
{
e1.printStackTrace();
}
}
// if condition
}
finally
{
if (jdbcconn != null)
{
// closing the connection
try
{
jdbcconn.close();
}
catch (SQLException e)
{
e.printStackTrace();
}
```

```
}
// if condition
}
// finally
}

private void debit(Connection jdbcconn, int accountno,
int debit_amount) throws SQLException
{
String debitSQL = "update account as Tab_Deduct JOIN "
+ " (select accountno, (amount - ?) as balance from account" + "
where accountno = ?) As tab "
+ " ON  Tab_Deduct.accountno = tab.accountno" + " set
Tab_Deduct.amount = tab.balance"
+ " where Tab_Deduct.accountno = ?";

PreparedStatement jdbcprepstmt = null;
try
{
jdbcprepstmt = jdbcconn.prepareStatement(debitSQL);
jdbcprepstmt.setInt(1, debit_amount);
jdbcprepstmt.setInt(2, accountno);
jdbcprepstmt.setInt(3, accountno);
intcountno = jdbcprepstmt.executeUpdate();
System.out.println("Amount debited and  Count of rows updated " +
countno);
if (countno == 0)
{
throw new SQLException("Account number not found " + accountno);
}
}
finally
{
if (jdbcprepstmt != null)
{
jdbcprepstmt.close();
}
}
}

private void credit(Connection jdbcconn, int accountno,
int credit_amount) throws SQLException
{
String creditSQL = "update account as Tab_Credit JOIN "
+ " (select accountno, (amount + ?) as balance from account" + "
where accountno = ?) As tab "
+ " ON  Tab_Credit.accountno = tab.accountno" + " set
Tab_Credit.amount = tab.balance"
+ " where Tab_Credit.accountno = ?";

PreparedStatement jdbcprepstmt = null;
try
{
jdbcprepstmt = jdbcconn.prepareStatement(creditSQL);
jdbcprepstmt.setInt(1, credit_amount);
jdbcprepstmt.setInt(2, accountno);
jdbcprepstmt.setInt(3, accountno);
int countno = jdbcprepstmt.executeUpdate();
System.out.println("Amount credited and Count of  rows updated  " +
countno);
if (countno == 0)
{
throw new SQLException("Account number not found " + accountno);
}
```

```
}
finally
{
if (jdbcprepstmt != null)
{
jdbcprepstmt.close();
}
}

}

private void preserveTrans(Connection jdbcconn, intfrom_account,
int to_account, intvalue) throws SQLException
{
String preserve = "Insert into all_transaction (from_account,
to_account, value) values ( ?, ?, ?)";
PreparedStatement jdbcprepstmt = null;
try
{
jdbcprepstmt = jdbcconn.prepareStatement(preserve);
jdbcprepstmt.setInt(1, from_account);
jdbcprepstmt.setInt(2, to_account);
jdbcprepstmt.setInt(3, value);
int countno = jdbcprepstmt.executeUpdate();
System.out.println("Count of rows inserted " + countno);
if (countno == 0)
{
throw new SQLException(
"Data insertion Problem - " + " From Account: " + from_account + "
To Account: " + to_account);
}
}
finally
{
if (jdbcprepstmt != null)
{
jdbcprepstmt.close();
}
}
}
}
```

Output:

```
Amount debited and  Count of rows updated 1
Amount credited and Count of  rows updated  1
java.sql.SQLSyntaxErrorException: Table 'emp.all_transaction'
doesn't exist
	at
com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.ja
va:120)
	at
com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.ja
va:97)
	at
com.mysql.cj.jdbc.exceptions.SQLExceptionsMapping.translateException
(SQLExceptionsMapping.java:122)
	at
com.mysql.cj.jdbc.ClientPreparedStatement.executeInternal(ClientPrep
aredStatement.java:953)
	at
com.mysql.cj.jdbc.ClientPreparedStatement.executeUpdateInternal(Clie
ntPreparedStatement.java:1092)
```

```
        at
com.mysql.cj.jdbc.ClientPreparedStatement.executeUpdateInternal(ClientPreparedStatement.java:1040)
        at
com.mysql.cj.jdbc.ClientPreparedStatement.executeLargeUpdate(ClientPreparedStatement.java:1347)
        at
com.mysql.cj.jdbc.ClientPreparedStatement.executeUpdate(ClientPreparedStatement.java:1025)
        at transSavepoint.preserveTrans(transSavepoint.java:130)
        at transSavepoint.transAmount(transSavepoint.java:40)
        at transSavepoint.main(transSavepoint.java:8)
Transaction Rollback to Savepoint
```

☞Check Your Progress - 2

1) What is a Transaction, and how is it maintained in the JDBC applications?

……

……

……

2) What is SavePoint in the JDBC application?

……

……

……

3) How do the dirty read, non-repeatable reads and phantom reads problems make the database inconsistent? What mechanisms we can use to resolve them?

……

……

……

16.4 JDBC BATCH PROCESSING

A JDBC batch contains a group of SQL statements that are processed together with one database call. JDBC batch is handy when a large number of SQL statements are executed concurrently after putting them in a set. The collection of queries are executed with one call. This reduces the network communication overhead and facilitates some of the queries to run in parallel and improve the overall efficiency.

The following are the commonly used methods for Batch Processing:

addBatch(): The addBatch() method of statement interface is used to add the queries to the batch.

executeBatch(): The executeBatch() method used to process the batch.

clearBatch(): The clearBatch() method is used to remove the batch, i.e. it will remove all the queries added in the batch, but we can't remove queries selectively.

The JDBC drivers support is not required for batch processing, but not all the database supports the batch processing.We can check the supported databases through DatabaseMetaData.supportsBatchUpdates() method. If this method returns

true implies that the database in context supports the batch processing else, it is not.

In batch processing, the database is executing each update separately, which implies there may be a possibility that out of many queries in the set, one of them may fail. Even if one of the queries fails from the collection, it may lead to an inconsistent database state. All successfully executed queries applied to the database but not those which fail. We can solve this unstable database problem that occurs due to the failure of any query by keeping the batch update inside the JDBC transaction. This step will ensure that the Transaction will execute in totality or none will be updated.

We can summarize the basic step needed to use the JDBC Batch Processing:

1. createStatement() method creates a Statement object.
2. setAutoCommit() method is used to set the auto-commit false.
3. addBatch() method to add multiple similar queries into a batch.
4. executeBatch() method to execute all the queries within the batch.
5. commit() method used to commit all the changes.

Let us take an example to understand the concept of batch processing.

Example 16.4:

```
import java.sql.*;
import java.sql.DriverManager;
public class batchProcess
{
public static void main(String[] args)
{
Connection jdbcconn = null;
PreparedStatement jdbcprepstmt  = null;
try
{
Class.forName("com.mysql.cj.jdbc.Driver");
System.out.println("Connecting to database...");
jdbcconn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/emp",
"root", "admin");
String insertSQL = "INSERT INTO EMP10"
+ "(id, first, last, age, city, salary) VALUES"
                              + "(?,?,?,?,?,?)";
jdbcprepstmt=jdbcconn.prepareStatement(insertSQL);

jdbcconn.setAutoCommit(false);

jdbcprepstmt=jdbcconn.prepareStatement(insertSQL);

jdbcprepstmt.setInt(1, 101);// 1 specify first parameter
jdbcprepstmt.setString(2, "Dinesh");
jdbcprepstmt.setString(3, "Sharma");
jdbcprepstmt.setInt(4, 40);
jdbcprepstmt.setString(5, "Delhi");
jdbcprepstmt.setInt(6, 8000);
jdbcprepstmt.addBatch();

jdbcprepstmt.setInt(1, 102);
jdbcprepstmt.setString(2, "Abhimanu");
jdbcprepstmt.setString(3, "Singh");
jdbcprepstmt.setInt(4, 50);
jdbcprepstmt.setString(5, "Mumbai");
jdbcprepstmt.setInt(6, 9000);
```

```
jdbcprepstmt.addBatch();

jdbcprepstmt.setInt(1, 105);
jdbcprepstmt.setString(2, "Abhijit");
jdbcprepstmt.setString(3, "Nayak");
jdbcprepstmt.setInt(4, 45);
jdbcprepstmt.setString(5, "Pune");
jdbcprepstmt.setInt(6, 8000);
jdbcprepstmt.addBatch();

jdbcprepstmt.executeBatch();

jdbcconn.commit();
}
catch (SQLException | ClassNotFoundException e)
{
e.printStackTrace();
if(jdbcconn != null)
{
try
{
System.out.println("Transaction Rollback due to error");
jdbcconn.rollback();
}
catch (SQLException e1)
{
e1.printStackTrace();
}
}
// if condition
}
finally
{
if(jdbcconn != null)
{
//closing the connection
try
{
jdbcconn.close();
System.out.println("Batch executed Successfully!");
}
catch (SQLException e)
{
e.printStackTrace();
}
}
// if condition
}
// finally
}
// main
}
//example
```

Output:

```
Connecting to database...
Batch executed Successfully!
```

16.5 JDBC ROWSET INTERFACE

The JDBC RowSet interface is an extension of the RowSet interface and is a wrap around a ResultSet object. RowSet objects inherently derive the capabilities of ResultSet. It allows to use the ResultSet as a JavaBeans component and supports JavaBeans event notification mechanism. It provides the data storage in the tabular form with some additional functionality like sending a notification to other registered components when a particular event is triggered. These additional capabilities make it more flexible and easier to use than a ResultSet.

RowSet types

The two types of RowSet objects are connected RowSet object and a disconnected RowSet object. A connected RowSet object establishes a connection with the database using a JDBC driver and sustains it till the application terminates. JDBCRowSet implementation is the only standard implementation that offers a connected RowSet object, and thus it is similar to ResultSet object.

On the contrary, a disconnected RowSet object only establishes a connection to the data source for reading data from the ResultSet or writing back to the data source. After executing the query, it disconnects and closes the connection. The other four implementations CachedRowSet, WebRowSet, JoinRowSet and FilteredRowSet, offers disconnected RowSet object. The advantage of a disconnected RowSet object is that they are lightweight as they need not maintain a permanent connection with the data source but still have all other functionalities of the connected RowSet object. These require a connection to establish every time to ensure the updates, and so it is a little bit slower in comparison to the connected RowSet object. But still, these are not only lightweight but also serializable and thus very suitable for network transmission.

Implementation Types

RowSet interface offers five implementation classes JDBCRowSet, CachedRowSet, WebRowSet, JoinRowSet and the FilteredRowSet. The RowSet objects are derived from the ResultSet, and it is the only connected RowSet. It offers additional capabilities like Scrollability and Updatability and JavaBeans Component-based development model.

The connectivity between JDBC RowSet and the datasource is preserved through its life cycle, which enables JdbcRowSet to take calls that invoke and, in return, call them on the ResultSet object. A ResultSet object can be made scrollable and updatable by the JdbcRowSet object, and this is useful when the driver and database are not offering these properties. The RowSet objects are scrollable and updatable by default and facilitate the ResultSet with these capabilities after populating the RowSet object with the content of the ResultSet, and this implies that we can traverse the records through the ResultSet object.

The RowSet objects are also used as a JavaBeans component and have properties and JavaBeans Notification Mechanism. The RowSet object property have getter and setter methods to fetch and set values like setInt(), getInt() etc. RowSet objects utilize the JavaBeans event model, in which registered components, i.e. all listeners notified when certain events like cursor movement, insert, update, delete operations or change in row content occur.

RowSets support JavaBeans events like cursorMoved, rowChanged and rowSetChanged. The cursorMoved event initiated due to the next() or previous(), i.e.

the rowChanged event-triggered due to a row insert, update, or delete. Whenever the execute method is called to create or change the RowSet, the rowSetChanged event is initiated.

An application component implements a RowSet listener, and when the event occurs, it performs a triggered action. These application components must register the listener objects with a RowSet.addRowSetListener method and implement the standard RowSetListener interface. The RowSet.removeRowSetListener method is used to unregistered the listener.

A CachedRowSet object offers all the necessary capabilities of the JDBC RowSet objects and the disconnected RowSet objects. The other three implementations are the extensions of this interface. The CachedRowSet provides the following in addition to what JDBCRowSet object provides:

1. connection establishment with the data source and query execution
2. reading data from the ResultSet and populating itself with that data
3. manipulating data during disconnected state
4. reconnection and write back to the data source
5. conflict resolution if any

A WebRowSet object provides the capability of reading and writing the XML document in addition to what CachedRowSet object provides:

A JoinRowSet object offers the ability to formulate an equivalent SQL JOIN without connecting to a data source and also provides all the capabilities that WebRowSet object offers.

A FilteredRowSet object provides the filtering capability to enable the visibility of selected data and also offers all the capabilities that WebRowSet object offers.

The event handling is used by calling the instance of **RowSetListener** in the addRowSetListener() method of JdbcRowSet interface. This interface three methods cursorMoved(RowSetEvent event, rowChanged(RowSetEvent event and rowSetChanged(RowSetEvent event).

Let us take an example to understand the concept of JDBC RowSet. In this example, we have fetched the data and performed the cursor movement task; also, we have used the event handling concept, which is not possible using ResultSet.

Example 16.5:

```
import javax.sql.RowSetEvent;
import javax.sql.RowSetListener;
import javax.sql.rowset.JdbcRowSet;
import javax.sql.rowset.RowSetProvider;
public class rowSet
{
public static void main(String[] args)
{
try
{

Class.forName("com.mysql.cj.jdbc.Driver");
System.out.println("Connecting to database...");
// RowSet Creation and Execution
JdbcRowSet jdbcRS = RowSetProvider.newFactory().createJdbcRowSet();
jdbcRS.setUrl("jdbc:mysql://localhost:3306/emp");
```

```
jdbcRS.setUsername("root");
jdbcRS.setPassword("admin");

jdbcRS.setCommand("select * from emp10");
jdbcRS.execute();

// Listener is added
jdbcRS.addRowSetListener(new MyListener());

while (jdbcRS.next())
{
// Event generated for cursor Movement
System.out.println("Id: " + jdbcRS.getInt(1));
System.out.println("first: " + jdbcRS.getString(2));
System.out.println("Salary: " + jdbcRS.getInt(6));
}
}
catch (Exception e)
{
//Handle errors for Class.forName
System.out.println(e);
}
// end try
System.out.println("Executed!");
}
// end main
}
// end Example

// listener implementation
class MyListener implements RowSetListener
{
public void cursorMoved(RowSetEvent event)
{
System.out.println("Cursor is Moved");
}

public void rowChanged(RowSetEvent event)
{
System.out.println("Cursor is Changed");
}
public void rowSetChanged(RowSetEvent event)
{
System.out.println("RowSet is changed");
}
}
```

```
Connecting to database...
Cursor is Moved
Id: 501
first: Aman
Salary: 8000
Cursor is Moved
Id: 502
first: Ajay
Salary: 9000
Cursor is Moved
Id: 505
first: Suraj
Salary: 8000
Cursor is Moved
Id: 509
first: Gaurav
```

```
Salary: 5000
Cursor is Moved
Executed!
```

16.6 INTRODUCTION TO JAVA DATA OBJECT (JDO)

Java Data Objects (JDO) is a specification which enables storage and retrieval of persistence data of Java objects. The complex applications require these persistent data or the information to be available beyond the life cycle of the program for various purposes.

JDBC API and JDO API allow us to access data through the java application. In JDBC, the relational database is used as a data source, but JDO uses any data source, maybe a relational database, object-oriented database or even flat files. JDBC mostly uses structured query language, while JDO uses Java code for the purpose.

In the case of JDBC, we have to write SQL queries and fetch the ResultSet into data objects, but JDO doesn't require execution of queries, copying JDBC ResultSet into Java objects etc. as all this is taken care of by JDO.

There are various other options available to the users to store and retrieve persistent data like JDBC, object databases and entity EJBs, but they have their limitations. JDO adds many useful features which are offered by other persistence mechanisms. The JDO provides many features like ease of creating persistence classes, supporting large datasets, also the object-oriented concepts, consistency, concurrency and query capabilities etc. JDO is flexible in terms of data storage as it may be a relational or object-oriented, or simple flat-file database, and the implementation is wholly hidden from the user.

The following are the advantage of using the JDO API:

a. Portable: The application based on JDO API are portable and can run on different vendors implementations without recompiling or changing any code.
b. Data Source access is transparent: The code to access the data source is independent of the database.
c. Ease of use: The JDO API facilitates the users to emphasis on the domain object model and the JDO implementation for persistence details.
d. High performance: JDO implementations look for efficient data access for performance optimization.
e. EJB Integration: EJB features are available for the application using the same domain object model throughout the enterprise.

The JDO offers encapsulation, transparency, and portability, and it supports various databases as per the requirements of the application. The JDO API consists of fewer interfaces and a standard for object persistence which makes it simple to implement. These all features offer greater flexibility to choose deployment environment and fast development.

☞Check Your Progress - 3

1) What is a JDBC RowSet, and how it is different from ResultSet?

……

……

……

2) What is batch processing in JDBC, and how does it improves efficiency?

……

……

……

3) What is Java Data Object API, and how is it different from JDBC API?

……

……

……

16.7 SUMMARY

In this unit, we have seen how the JDBC ResultSet interface is used to store the tabular data after the query execution. The data access using cursor subsequently is used by the java application. The cursor movement is controlled through various navigational methods. The ResultSet interface also provides concurrency mode to keep the data read-only or in the updatable mode as per the requirements. The ResultSet holdability provide another essential consideration of maintaining the context of data updates through the ResultSet instance.

The batch processing requires the execution of the transaction management, where we have discussed the usage of the commit and rollback methods. The Transaction management for Java applications is needed to satisfy the ACID property and to maintain the consistency.We have also discussed the various transaction isolation levels. These transaction isolation levels offer multiple methods to check the database inconsistencies and are vital while concurrent access of the database is taking place.

The JDBC RowSet interface provides connected and disconnected RowSet objects. The connected RowSet objects work like the ResultSet object to maintain the connectivity throughout the execution. In contrast, the disconnected one provides the advantage of establishing the connection only while accessing the data read/write and thus saves network usage and also is very lightweight for this reason. We have also discussed various implementation types of the RowSet interface. The Java Data objects are discussed briefly and providing an insight into their usage and advantages over JDBC.

16.8 SOLUTIONS/ANSWERS TO CHECK YOUR PROGRESS

☞Check Your Progress - 1

1) ResultSet is an object which is automatically created when we execute an SQL query and manage the fetched data from the database in the JDBC applications. With this ResultSet object, a cursor is automatically created to read the data from the object. We need to check the availability of the data before accessing it, so next() method is useful in it, which will return true if the next record is available and move the cursor to that record. We can fetch the data using getter methods.

2 The default ResultSet type in JDBC application is Read only and forward only. In JDBC applications, we can use ResultSet combination of Types and concurrency, and the following are the possible cases:

ResultSet.CONCUR_READ_ONLY(), ResultSet.TYPE_FORWARD_ONLY,

ResultSet.CONCUR_READ_ONLY(), ResultSet.TYPE_SCROLL_SENSITIVE

ResultSet.CONCUR_READ_ONLY(), ResultSet.TYPE_SCROLL_INSENSITIVE.

ResultSet.CONCUR_UPDATABLE(), ResultSet.TYPE_FORWARD_ONLY,

ResultSet.CONCUR_UPDATABLE(), ResultSet.TYPE_SCROLL_SENSITIVE

ResultSet.CONCUR_UPDATABLE(), ResultSet.TYPE_SCROLL_INSENSITIVE.

We can use a particular ResutlSet object type based on application requirements, and then we can choose any of the above combinations and pass as a parameter to createStatement() method.

3) The ResultSet holdability evaluates the ResultSet instance if it has a closed status when the underlying connection commit(). There are two types of holdability:

a. CLOSE_CURSORS_OVER_COMMIT: All the ResultSet instances are closed on commit()
b. HOLD_CURSORS_OVER_COMMIT: All the ResultSet instances are not closed but kept open to accommodate the updates with the same ResultSet on commit().

4. The ResultSet object will use ResultSet.TYPE_SCROLL_SENSITIVE and the following two methods to iterate the data in the forward direction and backward direction

Boolean next() and Boolean previous()

The following example explains this concept:

```
import java.sql.*;
import java.sql.DriverManager;
publicclass forwBack
{
publicstaticvoid main(String[] args)
{
Connectionjdbcconn = null;
Try
{
Class.forName("com.mysql.cj.jdbc.Driver");
jdbcconn = DriverManager.getConnection("jdbc:mysql://localhost:3306/emp", "root",
"admin");
Statement jdbcstmt =
jdbcconn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_U
PDATABLE);

ResultSet jdbcresultset = jdbcstmt.executeQuery("Select * from Emp");
System.out.println("data in forward direction");

System.out.println("ENO      ENAME      ESAL      EADDR");

System.out.println("**********************************");
while(jdbcresultset.next())
{
```

```
//Retrieve data
System.out.println("id : " + jdbcresultset.getInt("id") + " First : " +
jdbcresultset.getString("first")  + " Last : " + jdbcresultset.getString("last") +  " Age : " +
jdbcresultset.getInt("age") + " City : " + jdbcresultset.getString("city") + " Salary : " +
jdbcresultset.getInt("salary"));
}

System.out.println("data in forward direction");

System.out.println("ENO      ENAME      ESAL      EADDR");

while(jdbcresultset.previous())
{
//Retrieve data
System.out.println("id : " + jdbcresultset.getInt("id") + " First : " +
jdbcresultset.getString("first")  + " Last : " + jdbcresultset.getString("last") +  " Age : " +
jdbcresultset.getInt("age") + " City : " + jdbcresultset.getString("city") + " Salary : " +
jdbcresultset.getInt("salary"));
}
jdbcconn.close();
}
catch(Exception e)
{
System.out.println(e);
}
System.out.println("Success!");
}
}
```

☞Check Your Progress-2

1) A transaction is a logical unit containing one or more SQL queries and is executed in totality to perform a specific task. This mechanism is to ensure that either all dependent queries or batch of queries executed successfully or none. In general, transaction management supports ACID properties – Atomicity, Consistency, Isolation and Durability. By default, every query execution result automatically saves in JDBC applications, i.e. after the connection establishment with the database, the connection remains in an auto-commit mode. But at times, to improve the efficiency, we may require to group two or more statements to form as a part of the Transaction, and for that, we need to turn-off the auto-commit mode. This turning-off of the auto-commit mode is through setAutoCommit() method of the connection interface. This non-commit nature maintained the transactions in our JDBC applications.

2) The SavePoint option sets a Savepoint, i.e. checkpoint in the Transaction, which ensures that if rollback is required in the Transaction, it will only be up to the checkpoint instead of the complete rollback of the Transaction.

3) The concurrent access of the database may lead to the problem of the dirty read, non-repeatable reads and phantom reads which may make the database inconsistent. When a transaction reads an uncommitted value, it is a dirty read problem. When a transaction re-read a data item that is updated by another transaction, it is the non-repeatable read problem. The phantom reads problem happens when records are added or removed by another transaction, and then the transaction is executed on the database. These problems were resolved using a locking mechanism and controlled by the JDBC transaction Isolation level.

   There are different transaction levels ranging from TRANSACTION_NONE, which does not support transactions to TRANSACTION_SERIALIZABLE,

which strictly supports transactions following access rules and handles problems like dirty read, non-repeatable reads or phantom reads. The other Isolation level is TRANSACTION_READ_COMMITTED which support transactions and prevent dirty read but allows non-repeatable and phantom reads. The other one is the TRANSACTION_READ_UNCOMMITTED which support transactions but has no control over dirty read, non-repeatable and phantom reads. The fifth isolation level is TRANSACTION_ REPEATABLE_READ which also supports transactions and allow phantom read but prevent dirty and non-repeatable reads.

☞Check Your Progress -3

1) The JDBC RowSet interface is an extension of the ResultSet interface. It is a wrapper around a ResultSet object, and inherently derive the capabilities of ResultSet and thus offer more flexibility. A RowSet is broadly divided into Connected RowSet Objects and Disconnected RowSet Objects.

   The Connected RowSet Object always remains connected with the database, and it is very similar to the ResultSet object. The Disconnected RowSet Objects are not always connected to a database which makes it very lightweight and serializable.

2) A JDBC batch contains a group of SQL statements that are processed together with one database call. This is very useful when a large number of SQL statements are executed together after putting them in a batch. The batch queries are executed with one call, and this not only reduce the network communication overhead but alsosome of the queries may run in parallel and thus improve the overall efficiency.

3) JDO API implicitly enables the user to access a database, offering an abstraction of persistence through a standard interface-based Java model. JDBC API and JDO API allow us to access data through the java application. In JDBC, the relational database is used as a data source, but JDO may be using any data source may be a relational database, object-oriented database or even flat files. JDBC mostly uses structured query language, while JDO uses Java code for the purpose. In the case of JDBC, we have to write SQL queries and fetch the ResultSet into data objects, but JDO doesn't require execution of queries copying JDBC ResultSet into Java objects etc. as all this is taken care by JDO.

16.9 REFERENCES/FURTHER READING

- Herbert Schildt "Java The Complete Reference", McGraw-Hill,2017
- Horstmann, Cay S., and Gary Cornell, " *Core Java: Advanced Features*" Vol. 2. Pearson Education, 2013.
- Prasanalakshmi, "*Advanced Java Programming*", CBS Publishers 2015
- Sagayaraj, Denis, Karthik and Gajalakshmi ,"*Java Programming – for Core and Advanced Users*", Universities Press 2018
- Sharan, Kishori, "Beginning Java 8 APIs, Extensions and Libraries: Swing, JavaFX, JavaScript, JDBC and Network Programming APIs", Apress, 2014.
- Parsian, Mahmoud, "*DBC metadata, MySQL, and Oracle recipes: a problem-solution approach"* Apress, 2006.
- https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html
- https://www.roseindia.net/jdbc/