UNIT 1 INTRODUCTION TO GUI IN JAVA

Structure

1.0 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 is also used to be called as 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.

Java provides 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) programming are : 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 develop enterprise 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.

1.1 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.

1.2 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 see these Java APIs one by one, which are used for Graphical User Interface (GUI) programming in the Java family.

1.2.1 Abstract Window Toolkit (AWT)

AWT is 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 like textboxes, 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 contains a set of graphs libraries and classes used 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 1.

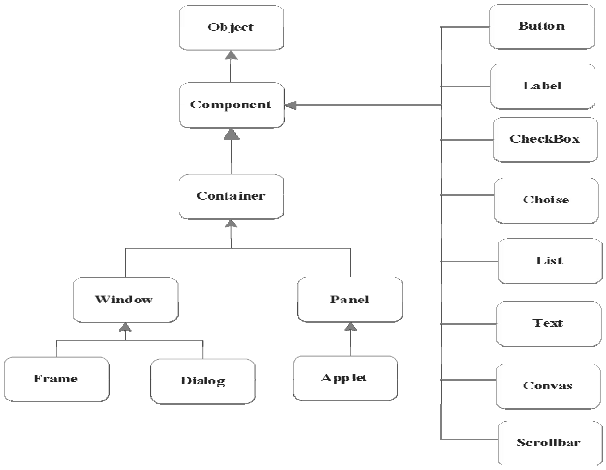

Figure 1: Hierarchy of Java AWT classes

Component Class

As we can see in figure 1, the Component class is on 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 the additional 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 can not 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 for window. 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 AWTFirstExample extends 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(WindowEvent e)
  {
    myFrame.dispose();
  }

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

Output:

Figure 1 : Output screen of Example 1

1.2.2 Swing

In section 1.2.1 we learned about AWT in Java, the first GUI API provided by Java programming lanuage. 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 only provides 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 2.

Table 2: Major Differences between AWT and Swing

S.No.	Java Swing	Java AWT
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 does not support pluggable look and feel.
4	It provides more advanced components than AWT e.g. tables, lists, scrollpanes, colorchooser, tabbedpane etc.	It provides fewer components than Swing
5	It supports the MVC design pattern	It does not follow the 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 Java is given in figure 3.

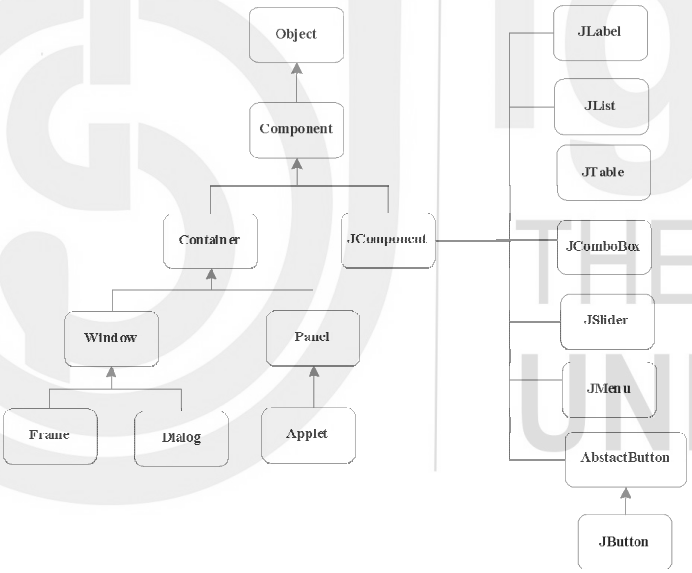

Figure 2: Hierarchy of Swing Package/Class

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 Java which make it platform-independent and does not map directly to platform-specific peers. Another reason is 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 is controlled 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 use a 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 and feel. 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 using Swing in Java

Example 2: Use of Swing components

```
package org.ignou.gui;
import Javax.swing.*;
import Java.awt.*;
public class MySwingExample extends 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 3: Output screen of above example 2

1.2.3 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 built on 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 GUI APIs (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 IntelliJ IDEA
    1. Open IntelliJ IDEA 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 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 no 6.
    4. Enter the Project Name(JavaFXFirstProgramme), Location, Language, select SDK, Change Group and Artifects as per your requyirment then click Next button to .

Figure 4: Create New Project in IntelliJ IDEA

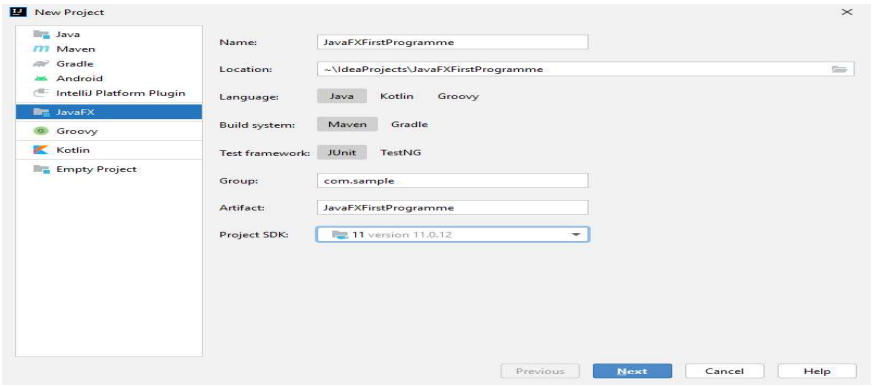

Figure 5: Creating Java FX Application in IntelliJ IDEA

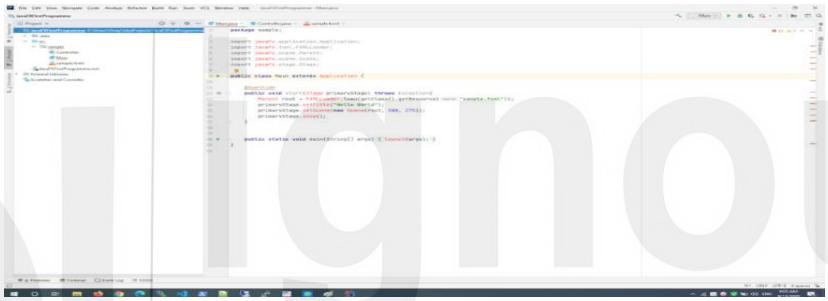

Figure 6: First look of default JavaFX Project with Error

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 no 8 .

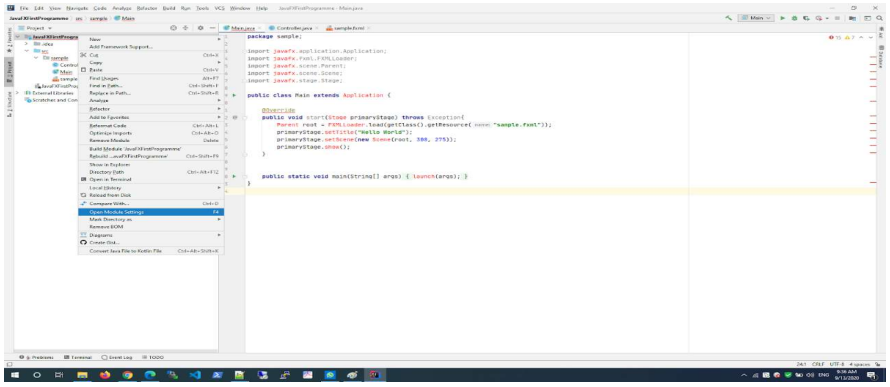

Figure 7: Open Project Setting

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 no 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.

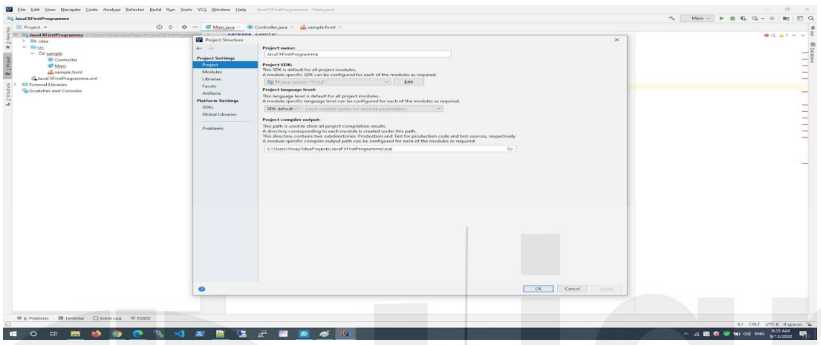

Figure 8: Project settings in IntelliJ IDEA

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 no. 10. Here we need to click on Plus(+) icon to add new Global Library. Doing this we are configuring IntelliJ for the specific libraries that we need to use in this particular Project.

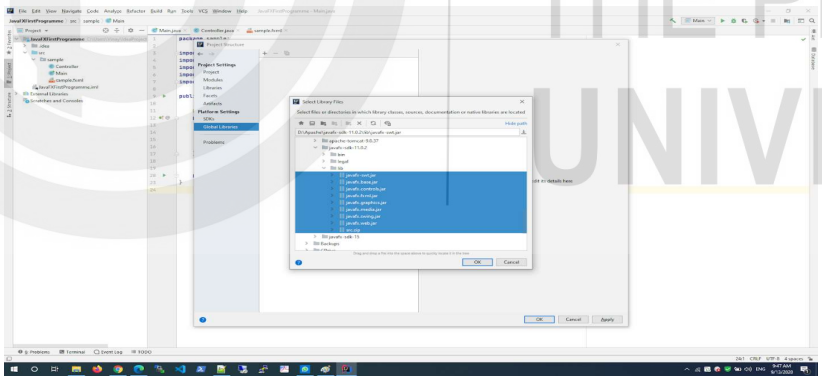

Figure10: Setting Global Libraries in Project Structure

Now you will no longer get any errors as you observed in the figure no 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 JDK 8 you will not have this issue.

Figure 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 define 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;
}
```

No test run the application; you will get the following output:

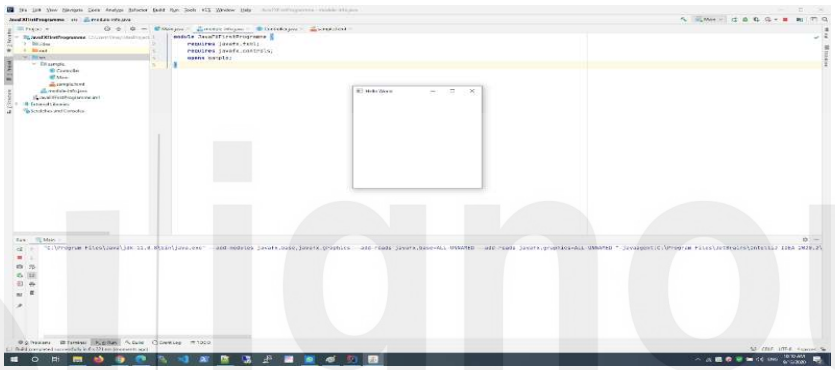

Figure 9(b): Output screen of First Run JavaFX Project

Finally, we have a directory structure of the First JavaFX Project as shown in figure no 12.

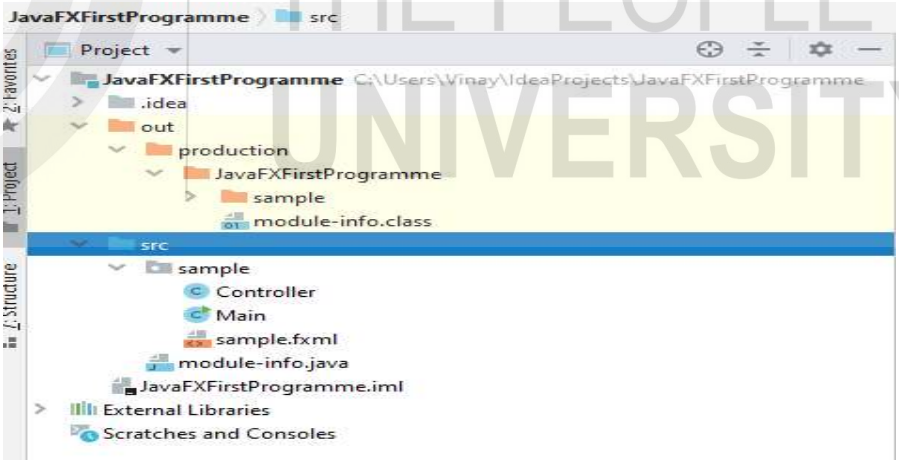

Figure 10: JavaFX Project Directory Structure

1.3 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 TreeTableView controls 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 (AmbientLight, PointLight subclasses) are added in 3D Graphics libraries. It also comprises of the Camera API class.
- Canvas API: This API allows 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 is included 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.

1.4 USER INTERFACE COMPONENTS OF JAVAFX

Your familiarity with the form components available in HTML will help you learn JavaFX components. In JavaFX 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
- SplitMenuButton
- 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 MyAccordionControlExample extends 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);
      VBox vBox = new VBox(accordion);
      Scene scene = new Scene(vBox);
      primaryStage.setScene(scene);
      primaryStage.show();
   }
   public static void main(String[] args)
   {
      launch(args);
   }
}
```

Output:

Figure 11: Output screen of Example no 3

- **Label:** It is a JavaFX component that is used to display 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 to allow the user to select from a range of values.
- **FileChooser:** A FileChooser control providess 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.

1.5 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 Layouts and 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 an essential 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. As you are familiar with buttons, especially ones for ok and cancel, which you have seen in various applications. You migh 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 would not 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 3 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.

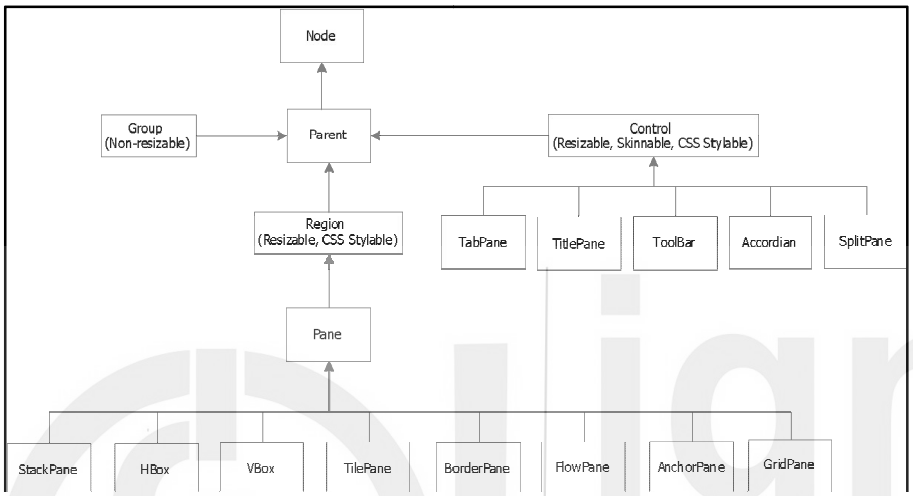

Figure 12: JavaFX 2.0 Layout Hierarchy

GridPane is the default layout of JavaFX Project setup as shown in figure 4.

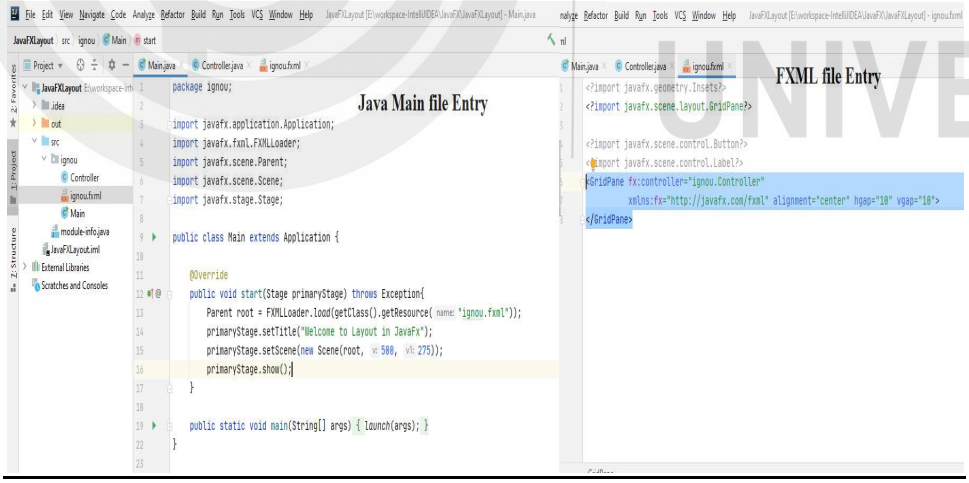

Figure 13: Default Pane coding in JavaFX Project

- **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?>

```
<HBox fx:controller="ignou.Controller" xmlns:fx=http://Javafx.com/fxml
alignment="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");
HBox hbox = 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?>
<VBox fx: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");
VBox vbox = 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?>
<BorderPane fx:controller="ignou.Controller"
xmlns:fx="http://Javafx.com/fxml">
   <bottom>
      <HBox xmlns:fx="http://Javafx.com/fxml">
         <padding>
            <Insets bottom="10" right="10"/>
         </padding>
         <Label text="Welcome to BorderPane Layout"/>
         <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");
BorderPane borderPane = 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?>
<StackPane fx: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");
StackPane stackPane = new StackPane();
stackPane.setMargin(label1, new Insets(50, 50, 50, 50));
ObservableList observableList = 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 TextFlow in the package Javafx.scene.layout denotes the text flow layout.

Syntax in FXML:

```
<?import Javafx.scene.text.TextFlow?>
<TextFlow fx: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));

TextFlow textFlowPane = 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 layout 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?>
<AnchorPane fx: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);
VBox vBox=new VBox(anchorPane1);
```

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

Syntax in FXML:

```
<?import Javafx.scene.layout.TilePane?>
<TilePane fx: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 TilePane 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));
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 organize the nodes in an application as 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?>
<GridPane fx: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 Gridpane 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));
```

```
GridPane gridPane1= 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?>
<FlowPane fx: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 MyGridPaneExample extends 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);

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

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

PasswordField pwBox = 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:

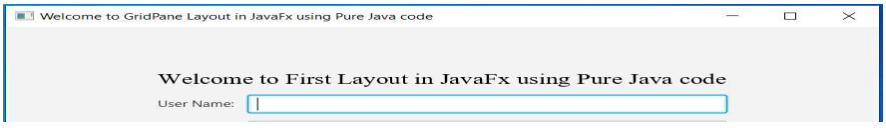

Figure 14: Output Screen of Example no 4

☞ 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?

   ……
   ……
   ……
   ……

1.6 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. 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 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 in JavaFX'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 MyHtmlCSSStyleExample extends 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:

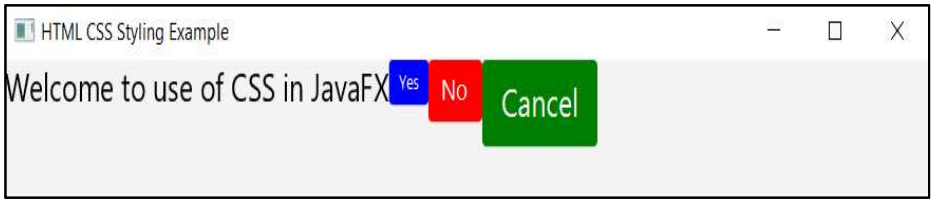

Figure 15: Output screen of Example no 5

1.7 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 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 follow

```
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 MyTextMain extends 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 16: 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.

   ……
   ……
   ……
   ……

1.8 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 given using 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 described how 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.

1.9 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));
```

1.10 REFERENCE/FURTHER READING

- 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.
- 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