**ASSIGNMENT-2**

**1. How to use java GUI features of JavafX.**

**JavaFX Applications**

Since the JavaFX library is written as a Java API, JavaFX application code can reference APIs from any Java library. For example, JavaFX applications can use Java API libraries to access native system capabilities and connect to server-based middleware applications.

The look and feel of JavaFX applications can be customized. Cascading Style Sheets (CSS) separate appearance and style from implementation so that developers can concentrate on coding. Graphic designers can easily customize the appearance and style of the application through the CSS. If you have a web design background, or if you would like to separate the user interface (UI) and the back-end logic, then you can develop the presentation aspects of the UI in the FXML scripting language and use Java code for the application logic. If you prefer to design UIs without writing code, then use JavaFX Scene Builder. As you design the UI, Scene Builder creates FXML markup that can be ported to an Integrated Development Environment (IDE) so that developers can add the business logic.

**Key Features**

The following features are included in JavaFX 8 and later releases. Items that were introduced in JavaFX 8 release are indicated accordingly:

* **Java APIs**. JavaFX is a Java library that consists of classes and interfaces that are written in Java code. The APIs are designed to be a friendly alternative to Java Virtual Machine (Java VM) languages, such as JRuby and Scala.
* **FXML and Scene Builder**. FXML is an XML-based declarative markup language for constructing a JavaFX application user interface. A designer can code in FXML or use JavaFX Scene Builder to interactively design the graphical user interface (GUI). Scene Builder generates FXML markup that can be ported to an IDE where a developer can add the business logic.
* **WebView**. A web component that uses WebKitHTML technology to make it possible to embed web pages within a JavaFX application. JavaScript running in WebView can call Java APIs, and Java APIs can call JavaScript running in WebView. Support for additional HTML5 features, including Web Sockets, Web Workers, and Web Fonts, and printing capabilities have been added in JavaFX 8. See [Adding HTML Content to JavaFX Applications](https://docs.oracle.com/javase/8/javafx/embedded-browser-tutorial/overview.htm#JFXWV135).
* **Swing interoperability**. Existing Swing applications can be updated with JavaFX features, such as rich graphics media playback and embedded Web content. The SwingNode class, which enables you to embed Swing content into JavaFX applications, has been added in JavaFX 8. See the [SwingNode API javadoc](https://docs.oracle.com/javase/8/javafx/api/) and [Embedding Swing Content in JavaFX Applications](https://docs.oracle.com/javase/8/javafx/interoperability-tutorial/embed-swing.htm#JFXIP566) for more information.
* **Built-in UI controls** **and CSS**. JavaFX provides all the major UI controls that are required to develop a full-featured application. Components can be skinned with standard Web technologies such as CSS. The DatePicker and TreeTableView UI controls are now available with the JavaFX 8 release. See [Using JavaFX UI Controls](https://docs.oracle.com/javase/8/javafx/user-interface-tutorial/ui_controls.htm#JFXUI336) for more information. Also, the CSS Styleable\* classes have become public API, allowing objects to be styled by CSS.
* **Modena theme*.*** The Modena theme replaces the Caspian theme as the default for JavaFX 8 applications. The Caspian theme is still available for your use by adding the setUserAgentStylesheet(STYLESHEET\_CASPIAN) line in your Application start() method. For more information, see the [Modena blog](http://fxexperience.com/2013/01/modena-new-theme-for-javafx-8/) at fxexperience.com
* **3D Graphics Features**. The new API classes for Shape3D (Box, Cylinder, MeshView, and Sphere subclasses), SubScene, Material, PickResult, LightBase (AmbientLight and PointLight subclasses), and SceneAntialiasing have been added to the 3D Graphics library in JavaFX 8. The Camera API class has also been updated in this release. For more information, see the [Getting Started with JavaFX 3D Graphics](https://docs.oracle.com/javase/8/javafx/graphics-tutorial/javafx-3d-graphics.htm#JFXGR256) document and the corresponding [API javadoc](https://docs.oracle.com/javase/8/javafx/api/) for javafx.scene.shape.Shape3D, javafx.scene.SubScene, javafx.scene.paint.Material, javafx.scene.input.PickResult, and javafx.scene.SceneAntialiasing.
* **Canvas API**. The Canvas API enables drawing directly within an area of the JavaFX scene that consists of one graphical element (node).
* **Printing API.** The javafx.print package has been added in Java SE 8 release and provides the public classes for the [JavaFX Printing API](https://docs.oracle.com/javase/8/javafx/api/).
* **Rich Text Support**. JavaFX 8 brings enhanced text support to JavaFX, including bi-directional text and complex text scripts, such as Thai and Hindu in controls, and multi-line, multi-style text in text nodes.
* **Multitouch Support**. JavaFX provides support for multitouch operations, based on the capabilities of the underlying platform.
* **Hi-DPI support.** JavaFX 8 now supports Hi-DPI displays.
* **Hardware-accelerated graphics pipeline**. JavaFX graphics are based on the graphics rendering pipeline (Prism). JavaFX offers 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 Prism defaults 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.

package helloworld;

import javafx.application.Application;

import javafx.event.ActionEvent;

import javafx.event.EventHandler;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.layout.StackPane;

import javafx.stage.Stage;

public class HelloWorld extends Application {

public static void main(String[] args) {

launch(args);

}

@Override

public void start(Stage primaryStage) {

primaryStage.setTitle("Hello World!");

Button btn = new Button();

* btn.setText("Say 'Hello World'");

btn.setOnAction(new EventHandler<ActionEvent>() {

@Override

public void handle(ActionEvent event) {

System.out.println("Hello World!");

}

});

StackPane root = new StackPane();

root.getChildren().add(btn);

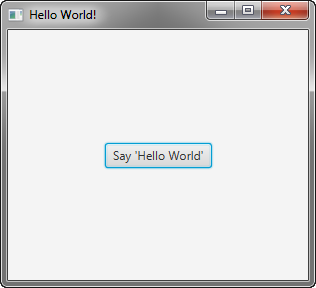
primaryStage.setScene(new Scene(root, 300, 250));

primaryStage.show();

}

}

**OUTPUT:**



1. **Volatile and transient keywords.**

**Volatile:** Volatile variable in Java is a special variable which is used to signal threads, a compiler that this particular variables value are going to be updated by multiple threads inside Java application. By making a variable volatile using the volatile keyword in Java, application programmer ensures that its value should always be read from [main memory](http://javarevisited.blogspot.sg/2011/05/java-heap-space-memory-size-jvm.html) and thread should not use cached value of that variable from their own stack. With the introduction of Java memory model from Java 5 onwards along with introduction of [CountDownLatch](http://javarevisited.blogspot.sg/2012/07/countdownlatch-example-in-java.html), [CyclicBarrier](http://javarevisited.blogspot.sg/2012/07/cyclicbarrier-example-java-5-concurrency-tutorial.html), [Semaphore](http://javarevisited.blogspot.sg/2012/05/counting-semaphore-example-in-java-5.html) and [ConcurrentHashMap](http://javarevisited.blogspot.sg/2011/04/difference-between-concurrenthashmap.html), volatile variable also guarantees "happens-before" relationship, which means not only another thread has visibility of latest value of volatile variable but also all the variable is seen by the thread which has updated value of volatile variable before these threads sees it.

**Transient:** Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.

import java.io.Serializable;

public class Student implements Serializable{

 int id;

 String name;

 transient int age;//Now it will not be serialized

 public Student(int id, String name,int age) {

  this.id = id;

  this.name = name;

  this.age=age;

 }

}