Generics

Objectives:

In this lab, student will be able to:

1. Understand the benefits of generics
2. Create generic classes and methods

Introduction to Generics:

Generics are a powerful extension to Java because they streamline the creation of type-safe, reusable code. The term *generics* means *parameterized types*. Parameterized types are important because they enable the programmer to create classes, interfaces, and methods in which the type of data upon which they operate is specified as a parameter. Using generics, it is possible to create a single class, for example, that automatically works with different types of data. A class, interface, or method that operates on a parameterized type is called *generic,* as in *generic class* or *generic method.*

Solved exercise:

1. Program defines two classes, the generic class Gen, and the second is GenDemo, which uses Gen. Here, T is a type parameter that will be replaced by a real type when an object of type Gen is created.

```
class Gen<T> {
        T ob;           // declare an object of type T
        // Pass the constructor a reference to an object of type T.
        Gen(T o) {
                ob = o;
          }
    // Return ob.
        T getob() {
                returnob;
          }
    // Show type of T.
void showType() {
    System.out.println("Type of T is " +ob.getClass().getName());
        }
}
// Demonstrate the generic class.
Class GenDemo {
        public static void main(String args[]) {
         // Create a Gen reference for Integers.
               Gen<Integer>iOb;
         // Create a Gen<Integer> object and assign its
         // reference to iOb. Notice the use of autoboxing
        // to encapsulate the value 88 within an Integer object.
               iOb = new Gen<Integer>(88);
        // Show the type of data used by iOb.
               iOb.showType();
        // Get the value in iOb. Notice that no cast is needed.
               int v = iOb.getob(); System.out.println("value:
               " + v); System.out.println();
         // Create a Gen object for Strings.
               Gen<String>strOb = new Gen<String>("Generics Test");
         // Show the type of data used by strOb.
               strOb.showType();
         // Get the value of strOb. Again, notice that no cast is needed.
               String   str   =   strOb.getob();
               System.out.println("value: " + str);
         }
}
```

Output:

```
        Type of T is java.lang.Integer
        value: 88
        Type of T is java.lang.String
        value: Generics Test
```

Here, T is the name of a type parameter. This name is used as a placeholder for the actual type that will be passed to Gen when an object is created. The GenDemo class demonstrates the generic Gen class. It first creates a version of Gen for integers, as shown here:

```
Gen<Integer>iOb;
```

The type Integer is specified within the angle brackets after Gen. In this case, Integer is a type argument that is passed to Gen's type parameter, T. This effectively creates a version of Gen in which all references to T are translated into references to Integer. Thus, for this declaration, ob is of type Integer, and the return type of getob( ) is of type Integer.

The next line assigns to **iOb**a reference to an instance of an **Integer** version of the **Gen**class:

```
                iOb = new Gen<Integer>(88);
```

Generics Work Only with Objects. Therefore, the following declaration is illegal:

```
        Gen<int>strOb = new Gen<int>(53);      // Error, can't use primitive type
```

Lab exercises:

1. Write a generic method to exchange the positions of two different elements in an array.
2. Define a simple generic stack class and show the use of the generic class for two different class types Student and Employee class objects.
3. Define a generic List class to implement a singly linked list and show the use of the generic class for two different class types **Integer and Double** class objects.
4. Write a program to demonstrate the use of wildcard arguments.

Additional exercises:

1. Write a generic method that can print array of different type using a single Generic method:
2. Write a generic method to return the largest of three Comparable objects.

JavaFX and Event Handling

Objectives:

Objectives:
In this lab, student will be able to:

1. Understand importance of light weight components and pluggable look-and-feel
2. Create, compile and run JavaFX applications
3. Know the fundamentals of event handling and role of layout managers

JavaFX:
JavaFX components are lightweight and events are handled in an easy-to-manage, straightforward manner. The JavaFX framework is contained in packages that begin with the javafx prefix. The packages we will use in our code are : javafx.application, javafx.stage, javafx.scene, and javafx.scene.layout.

The Stage and Scene Classes:
Stage is a top-level container. All JavaFX applications automatically have access to one Stage, called the primary stage. The primary stage is supplied by the run-time system when a JavaFX application is started.
Scene is a container for the items that comprise the scene. These can consist of controls, such as push buttons and check boxes, text, and graphics. To create a scene, you will add those elements to an instance of Scene.

Layouts:
JavaFX provides several layout panes that manage the process of placing elements in a scene. For example, the FlowPane class provides a flow layout and the GridPane class supports a row/column grid-based layout.

The Application Class and Life-Cycle methods:

A JavaFX application must be a subclass of the Application class, which is packaged in javafx.application. Thus, your application class will extend Application. The Application class defines three life-cycle methods that your application can override.

1. **void init( )** - The init( ) method is called when the application begins execution. It is used to perform various initializations.

2. **abstract void start(Stage primaryStage)** - The start( ) method is called after init( ). This is where the application begins and it can be used to construct and set the scene. This method is abstract and hence must be overridden by the application.

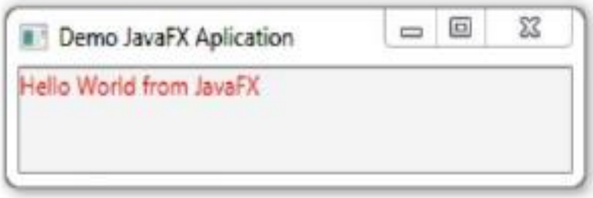
3. **void stop( )** - When the application is terminated, the stop( ) method is called. It can be used to handle any cleanup or shutdown chores.

Solved exercise:

1. Write a JavaFX Application program to display a simple message.

```
import javafx.scene.control.*;
import javafx.application.Application;
import javafx.stage.Stage;
import      javafx.scene.Scene;
import    javafx.scene.layout.*;
import javafx.scene.paint.Color;

public class HelloWorld extends Application {

  public void start(Stage primaryStage) { // entry point for the application
    primaryStage.setTitle("Demo JavaFX Aplication"); // set the title of top level
    //container.
    Label lbl = new Label();    // create a label control
    lbl.setText("Hello    World    from    JavaFX");
    lbl.setTextFill(Color.RED);

    FlowPane   root   =   new   FlowPane();   //   create   a   root   node.
    root.getChildren().add(lbl); // add the label to the node
    Scene myScene = new Scene(root, 300, 50); // create a scene using the node
    primaryStage.setScene(myScene);   //   set   the   scene   on   the   stage
    primaryStage.show(); // show the stage
    }
public static void main(String[] args) {
      launch(args); // Start the JavaFX application by calling launch( ).
  }
 }
```

Output:



Hello World from JavaFX

Event Handling:

The JavaFX controls that respond to either the external user input events or the internal events need to be handled. The delegation event model is the mechanism of handling such events. The advantage of this model is that application logic that processes the events is cleanly separated from the User Interface logic that generates the event.

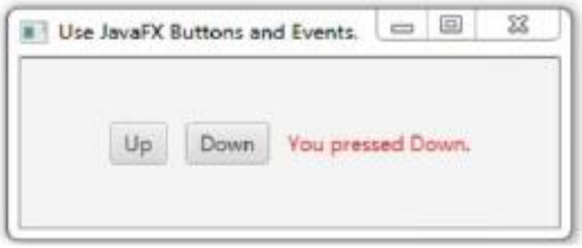
Solved exercise:

2. Write a JavaFX-Application program that handles the event generated by Button control.

```
import javafx.application.*;
import javafx.scene.*;
import javafx.stage.*;
import javafx.scene.layout.*;
import javafx.scene.control.*;
import javafx.event.*;
import javafx.geometry.*;
public class JavaFXEventDemo extends Application {
Label response;
public static void main(String[] args) {
// Start the JavaFX application by calling launch().
launch(args);
}

// Override the start() method.
public void start(Stage myStage) {
```

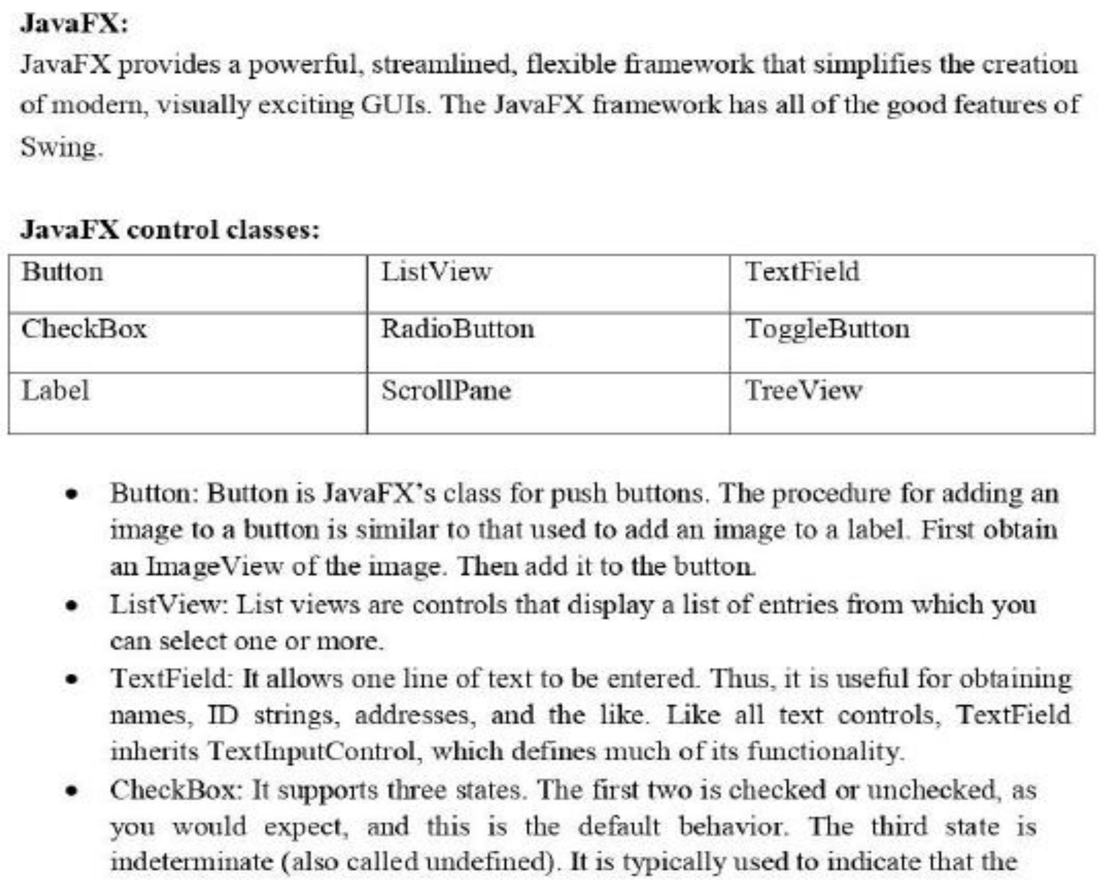
```
// Give the stage a title.
myStage.setTitle("Use JavaFX Buttons and Events.");
// Use a FlowPane for the root node. In this case,
// vertical and horizontal gaps of 10. FlowPane
 rootNode = new FlowPane(10, 10);
// Center the controls in the scene.
rootNode.setAlignment(Pos.CENTER);
// Create a scene.
Scene myScene = new Scene(rootNode, 300, 100);

// Set the scene on the stage.
myStage.setScene(myScene);
// Create a label.
response = new Label("Push a Button");
// Create two push buttons.
Button btnUp = new Button("Up");
Button btnDown = new Button("Down");
// Handle the action events for the Up button.
btnUp.setOnAction(new EventHandler<ActionEvent>() {
public void handle(ActionEvent ae) {
response.setText("You pressed Up."); } });
// Handle the action events for the Down button.
btnDown.setOnAction(new EventHandler<ActionEvent>() {
public void handle(ActionEvent ae) { response.setText("You
pressed Down."); } });
// Add the label and buttons to the scene graph.
rootNode.getChildren().addAll(btnUp, btnDown, response);
 // Show the stage and its scene.
myStage.show(); } }
```

Output:



You pressed Down.

JAVA FX UI Controls:



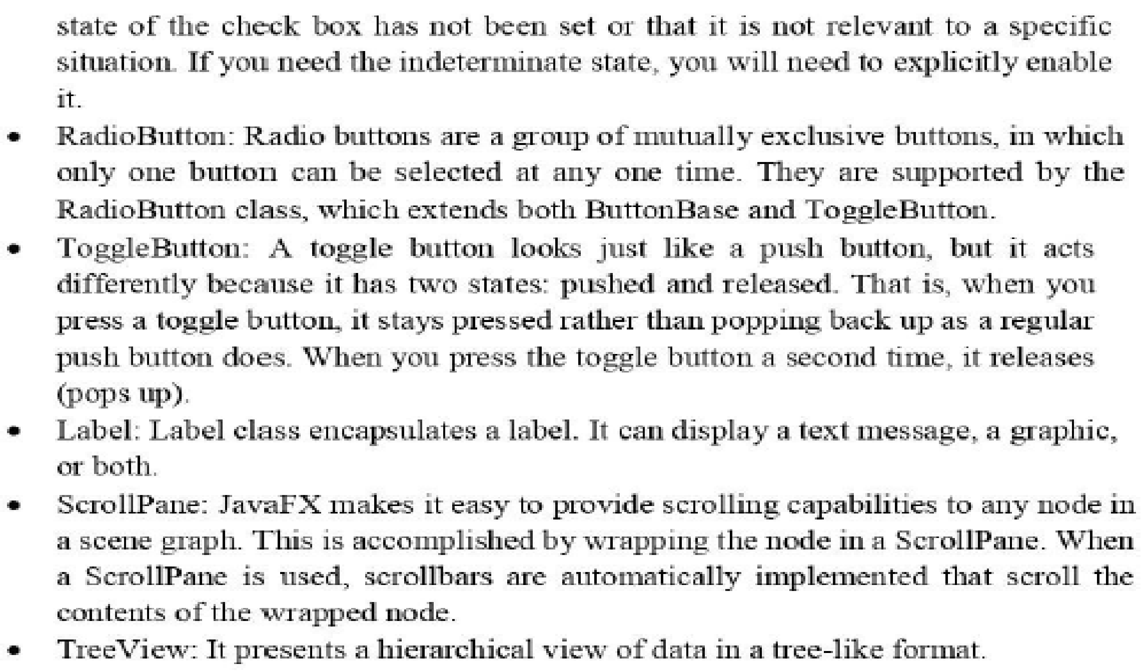
JavaFX:

JavaFX provides a powerful, streamlined, flexible framework that simplifies the creation of modern, visually exciting GUIs. The JavaFX framework has all of the good features of Swing.

JavaFX control classes:

Button	ListView	TextField
CheckBox	RadioButton	ToggleButton
Label	ScrollPane	TreeView

- Button: Button is JavaFX's class for push buttons. The procedure for adding an image to a button is similar to that used to add an image to a label. First obtain an ImageView of the image. Then add it to the button.
- ListView: List views are controls that display a list of entries from which you can select one or more.
- TextField: It allows one line of text to be entered. Thus, it is useful for obtaining names, ID strings, addresses, and the like. Like all text controls, TextField inherits TextInputControl, which defines much of its functionality.
- CheckBox: It supports three states. The first two is checked or unchecked, as you would expect, and this is the default behavior. The third state is indeterminate (also called undefined). It is typically used to indicate that the



state of the check box has not been set or that it is not relevant to a specific situation. If you need the indeterminate state, you will need to explicitly enable it.

- RadioButton: Radio buttons are a group of mutually exclusive buttons, in which only one button can be selected at any one time. They are supported by the RadioButton class, which extends both ButtonBase and ToggleButton.
- ToggleButton: A toggle button looks just like a push button, but it acts differently because it has two states: pushed and released. That is, when you press a toggle button, it stays pressed rather than popping back up as a regular push button does. When you press the toggle button a second time, it releases (pops up).
- Label: Label class encapsulates a label. It can display a text message, a graphic, or both.
- ScrollPane: JavaFX makes it easy to provide scrolling capabilities to any node in a scene graph. This is accomplished by wrapping the node in a ScrollPane. When a ScrollPane is used, scrollbars are automatically implemented that scroll the contents of the wrapped node.
- TreeView: It presents a hierarchical view of data in a tree-like format.

Lab exercises:

1. Write a JavaFX application program to do the following:
   a. Display the message "Welcome to JavaFX programming" using Label in the Scene.
   b. Set the text color of the Label to Magenta.
   c. Set the title of the Stage to "This is the first JavaFX Application".
   d. Set the width and height of the Scene to 500 and 200 respectively.
   e. Use FlowPane layout and set the hgap and vgap of the FlowPane to desired values.
2. Write a JavaFX program to accept an integer from the user in a text field and display the multiplication table (up to number *10) for that number. Use FlowPane layout for the application.
3. Write a JavaFX program to display a window as shown below. Use TextField for UserName and PasswordField for Password input. On click of "Sign in" Button the message "Welcome UserName" should be displayed in a Text Control. Use GridPane layout for the application.

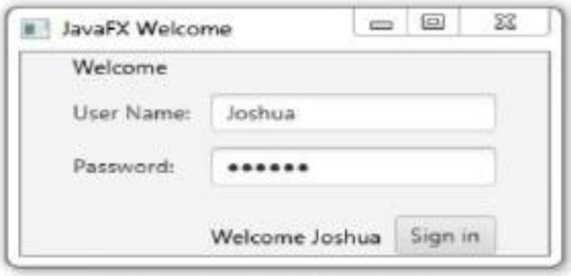


Fig 11.1 Welcome Window

4. Write a JavaFX program that obtains two positive integers passed from the user in two text fields and displays the numbers and their GCD as the result on a Label.

Additional exercises:

1. Define a class called Employee with the attributes name, empID, designation, basicPay, DA, HRA, PF, LIC, netSalary. DA is 40% of basicPay, HRA is 15% of basicPay, PF is 12% of basicPay. Display all the employee information in a JavaFX application.
2. Design a simple calculator to show the working for simple arithmetic operations. Use Grid layout.

Lab Exercises:

1. Write a JavaFX application program that obtains two floating point numbers in two text fields from the user and displays the sum, product, difference and quotient of these numbers using Canvas on clicking compute button with a calculator image placed on it.
2. Write a JavaFX application program to create your resume. Use checkbox to select the languages which you can speak. On clicking the Submit button all the details of the resume should be displayed using Canvas.
3. Write a JavaFX application program that creates a thread which will scroll the message from right to left across the window or left to right based on RadioButton option selected by the user.
4. Write a JavaFX application program that displays a Circle whose diameter is entered by the user in a text field and calculates and displays the area, radius, diameter and circumference using Canvas based on the option chosen in List View (Area or radius and so on).

Additional exercises:

1. Write a JavaFX program to simulate a static analog clock whose display is controlled by a user controlled static digital clock.
2. Write a JavaFX program to implement a simple calculator using the Toggle buttons.