Experiment No: 9

AIM: To implement event handling and animation.

Date:

CO mapped: CO-5

Objectives:

To proficiently implement event handling and animation in software applications, fostering user interaction and engagement. Mastery of event-driven programming and animation techniques will empower the creation of dynamic, responsive, and visually captivating software experiences that cater to user needs and preferences.

Background:

Responding to user events

For a GUI application to be interactive, various elements such as buttons have to be able to respond to interactions from the user, such as clicks. In GUI applications user actions such as mouse clicks and key presses are called *events*. To set up an element such as a button to respond to user events, we arrange to connect special *event handling* code to the button.

Our first example demonstrates one way to do this in JavaFX. The first step is to connect an object to the button as the button's event handler via the button's setOnAction() method. The requirement here is that the object that we link to the button has to implement a particular interface, the EventHandler<ActionEvent> interface. That interface has one method in it, a handle() method that will get called when the user clicks on the button.

For the event handler code to do something useful, it will typically need to have access to one or more elements in the scene that will be affected by the button click. In this example, clicking the button will trigger a change in the text displayed in a label in the scene. To make this all work, the class we set up needs to have a member variable that is a reference to the label object. The code in handle() will use that reference to change the text shown in the label when the user clicks on the button.

Also, insert the code for the following class at the bottom of the App.java file:

```
classClickHandler implements EventHandler<ActionEvent> {

publicClickHandler(Label label) {

this.label = label;

   }


public void handle(ActionEventevt) {

label.setText("Hello, World!");

   }
```

```
private Label label;
}
```

A better way to handle events

Although the process for linking an event handler to a button is fairly straightforward, it is a little clunky. This process can get even more tedious when we start building applications with many buttons that need event handlers. As a fix for this, JavaFX allows us to use a simpler mechanism to set up event handlers.

To see how this mechanism works, remove the ClickHandler class completely and replace the line of code in start() that calls the button's setOnAction() method with this:

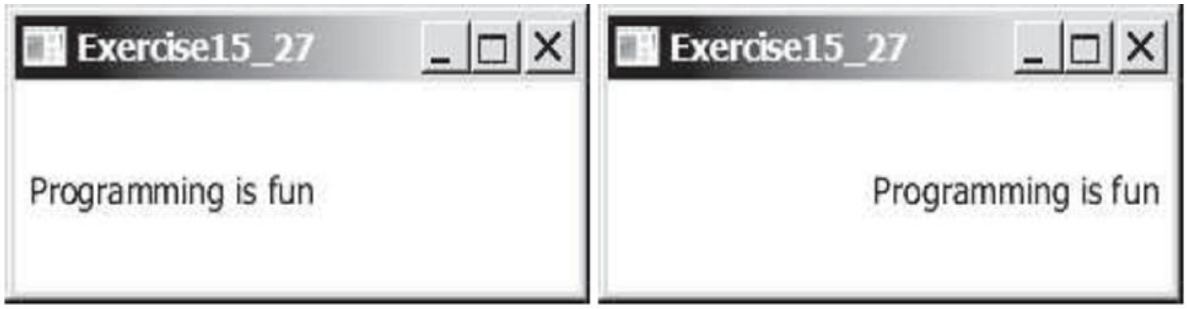
```
btn.setOnAction((e)->{label.setText("Hello, World!");});
```

The ClickHandler class that you just eliminated had a handle() method in it. That method took a single parameter, which was an ActionEvent object, e. The code we just put in place of the original code contains a *lambda expression* mapping that parameter e to a chunk of code that will run when the event takes place. This code is the code that used to live in the body of the handle() method. The new statement saves a lot of space over the original. The lambda expression replaces the ClickHandler class and its handle() method with a simpler alternative.

Practical questions:

1. Write a program that can dynamically change the font of a text in a label displayed on a stack pane. The text can be displayed in bold and italic at the same time. You can select the font name or font size from combo boxes, as shown in Figure. The available font names can be obtained using Font.getFamilies(). The combo box for the font size is initialized with numbers from 1 to 100.

1. Write a program that displays a moving text, as shown in Figure. The text moves from left to right circularly. When it disappears in the right, it reappears from the left. The text freezes when the mouse is pressed and moves again when the button is released.



3. Create animation in Figure to meet the following requirements:

■ Allow the user to specify the animation speed in a text field.

■ Get the number of iamges and image's file-name prefix from the user. For example, if the user enters n for the number of images and L for the image prefix, then the files are L1.gif, L2.gif, and so on, to Ln.gif. Assume that the images are stored in the image directory, a subdirectory of the program's class directory. The animation displays the images one after the other.

■ Allow the user to specify an audio file URL. The audio is played while the animation runs.

Observations: Put Output of the program

Conclusion: (Sufficient space to be provided)

Quiz: (Sufficient space to be provided for the answers)

1. How does event handling work in Java, and what is the event-driven programming model?
2. What is the role of the java.awt.event and javafx.event packages in Java event handling?
3. Explain: MouseEvent, KeyEvent, ActionEvent
4. What are the primary libraries or frameworks for creating animations in Java, and which one do you prefer?

5. How to set the cycle count of an animation to infinite?

Suggested Reference:

1. https://www.tutorialspoint.com/java/
2. https://www.geeksforgeeks.org/
3. https://www.w3schools.com/java/
4. https://www.javatpoint.com/