Asst. Prof. Dr. Sasiporn Usanavasin, Dr. Jessada Karnjana, Dr. Kasorn Galajit and Dr. Akkarawoot Takhom {sasiporn.us, akkharawoot.aj, jessada.aj, kasorn.aj}@siit.tu.ac.th
School of Information, Computer, and Communication Technology, Sirindhorn International Institute of Technology, Thammasat University.

DES103(Y23S2)-LAB08-REVIEW]

Event-driven Programming EP1

Learning Objectives

1. To understand the definition of a listener.
2. To acquire the skill of registering an appropriate listener to the source.
3. To learn the implementation of suitable methods and their respective details for the specified listener to execute the assigned task.

****Remark**** A ***pointer finger (👉)*** refers to an explanation between students and their TA.

8.1 Event-Driven Programming:

Event-driven programming is a programming paradigm where code execution is triggered by the occurrence of ***events***.

8.2 Events

- **Definition:** An event serves as a signal to the program indicating that a particular action or occurrence has taken place.
- **Sources:** Events are typically generated by:
    - **External User Actions**: These include interactions such as:
        - Mouse Movements: When the user moves the mouse cursor.
        - Mouse Clicks: When the user clicks a mouse button.
        - Keystrokes: When the user presses keys on the keyboard.
    - **Operating System**: Events can also be triggered by the operating system, for example:
        - Timer Events: The operating system sends a signal to the program after a specific duration has elapsed, allowing for time-sensitive functionalities or periodic tasks.
        - System Notifications: Such as notifications about changes in system state or availability of resources.
        - Network Events: Signals related to network activity, such as incoming data packets or connection status changes.
- **Role**: Events serve as triggers that drive the flow of execution within event-driven programs. They enable developers to create dynamic and responsive software systems that can

effectively interact with users and adapt to changing environments. By responding to events, programs can perform specific actions or update their state in real-time, enhancing user experience and overall functionality.

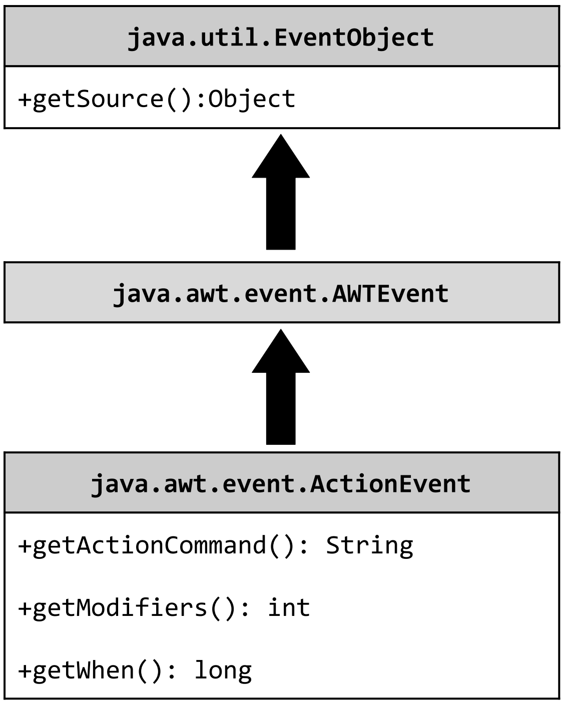
8.3 Examples of sources and events

User actions	Source objects	Type of fired events
Click a button	`JButton`	`ActionEvent`
Click a checkbox	`JCheckBox`	`ItemEvent, ActionEvent`
Click a radio button	`JRadioButton`	`ItemEvent, ActionEvent`
Press return on a text field	`JTextField`	`ActionEvent`
Select a new item	`JComboBox`	`ItemEvent, ActionEvent`
Windows opened, closed, etc.	`Window`	`WindowEvent`
Mouse pressed, released, dragged etc.	`Mouse`	`MouseEvent`
Key released, pressed, etc.	`Keyboard`	`KeyEvent`

Asst. Prof. Dr. Sasiporn Usanavasin, Dr. Jessada Karnjana, Dr. Kasorn Galajit and Dr. Akkarawoot Takhom {sasiporn.us, akkharawoot.aj, jessada.aj, kasorn.aj}@siit.tu.ac.th
School of Information, Computer, and Communication Technology, Sirindhorn International Institute of Technology, Thammasat University.

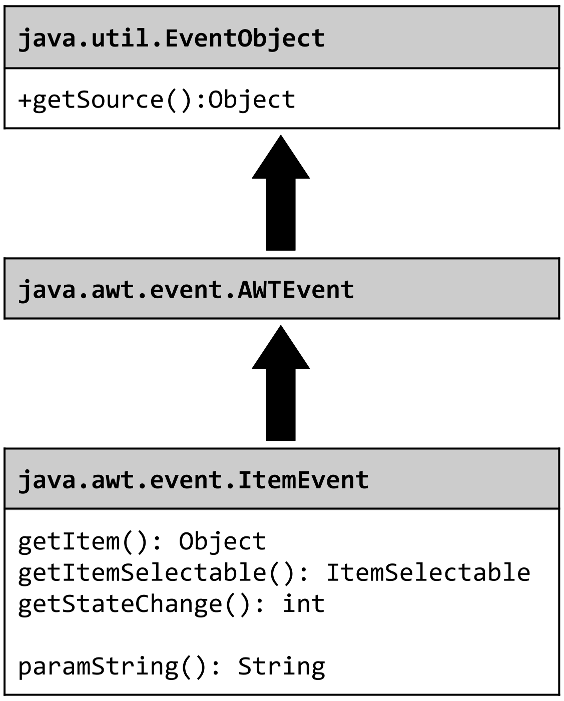
8.4 EventObject

8.4.1 ActionEvent



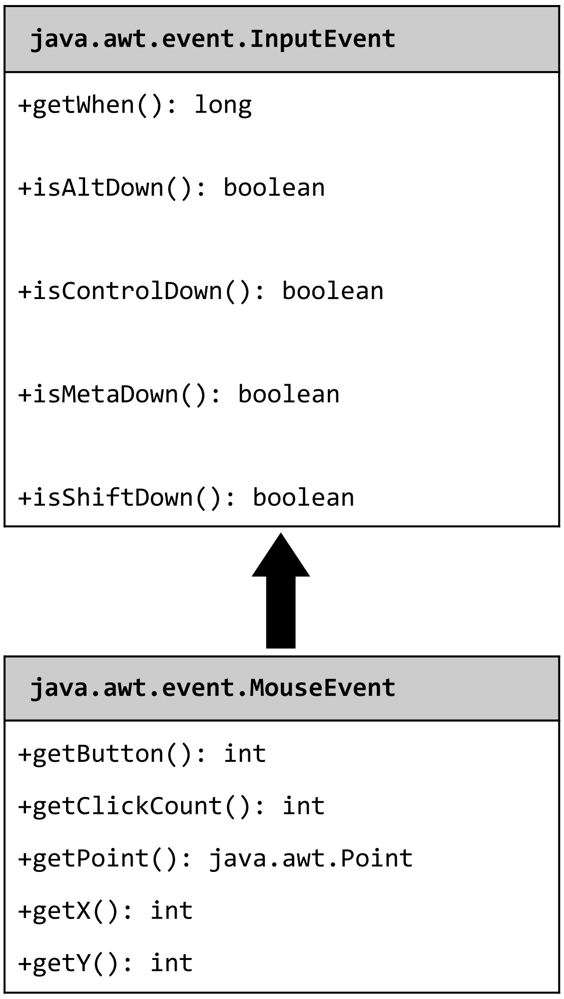
- ❖ Returns the object on which the event initially occurred.
- ❖ Returns the *command* string associated with this action. For a button, its text is the command string.
- ❖ Returns the *modifier keys* held down during this action event.
- ❖ Returns the *timestamp* when this event occurred. The time is the number of milliseconds since January 1, 1970, 00:00:00 GMT.

8.4.2 ItemEvent



- ❖ Returns the object on which the event initially occurred.
- ❖ Returns the item affected by the event
- ❖ Returns the originator of the event
- ❖ Returns the type of state change (selected or deselected).
- ❖ Returns a parameter string identifying this item event.

8.4.3 MouseEvent



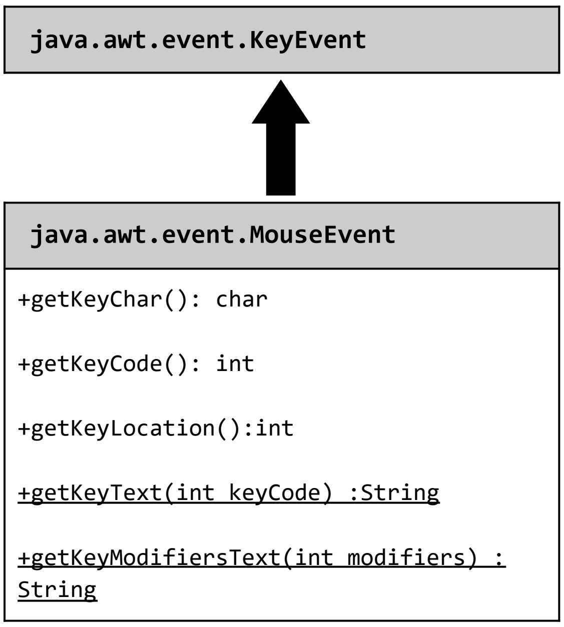
```
java.awt.event.InputEvent
+getWhen(): long
+isAltDown(): boolean
+isControlDown(): boolean
+isMetaDown(): boolean
+isShiftDown(): boolean
```

```
java.awt.event.MouseEvent
+getButton(): int
+getClickCount(): int
+getPoint(): java.awt.Point
+getX(): int
+getY(): int
```

- ❖ Returns the timestamp when this event occurred.
- ❖ Returns whether or not the Alt modifier is down on this event.
- ❖ Returns whether or not the Control modifier is down on this event.
- ❖ Returns whether or not the Meta modifier is down on this event
- ❖ Returns whether or not the Shift modifier is down on this event.

- ❖ Indicates which mouse button has been clicked.
- ❖ Returns the number of mouse clicks associated with this event.
- ❖ Returns a Point object containing the x and y coordinates.
- ❖ Returns the x-coordinate of the mouse point. Returns the y-coordinate of the mouse point.

8.4.3 MouseEvent



```
java.awt.event.KeyEvent
```

```
java.awt.event.MouseEvent
+getKeyChar(): char
+getKeyCode(): int
+getKeyLocation():int
+getKeyText(int keyCode) :String
+getKeyModifiersText(int modifiers) : String
```

- ❖ Returns the character associated with the key in this event.
- ❖ Returns the integer keyCode associated with the key in this event.
- ❖ Returns the location of the key that originated this key event.
- ❖ Returns a String describing the keyCode, Ex., "HOME", "F1" or "A".
- ❖ Returns a String describing the modifier key(s), such as "Shift", or "Ctrl+Shift".

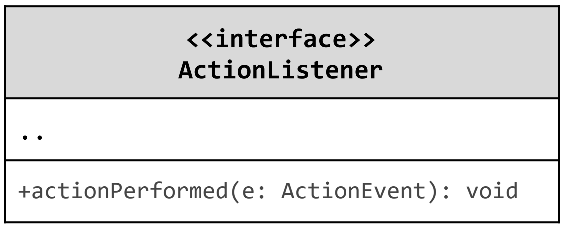
Asst. Prof. Dr. Sasiporn Usanavasin, Dr. Jessada Karnjana, Dr. Kasorn Galajit and Dr. Akkarawoot Takhom {sasiporn.us, akkharawoot.aj, jessada.aj, kasorn.aj}@siit.tu.ac.th
School of Information, Computer, and Communication Technology, Sirindhorn International Institute of Technology, Thammasat University.

8.5 Interaction between

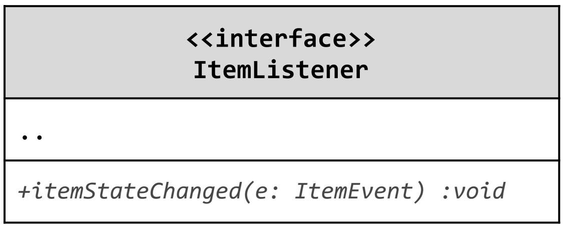
Source and Listener

8.5.1 UML of Listener's Class

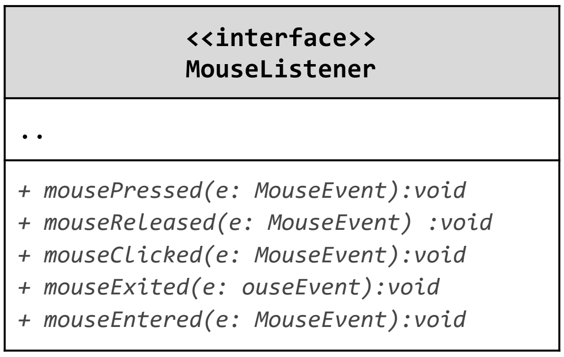
Listeners are defined as <<interface>>



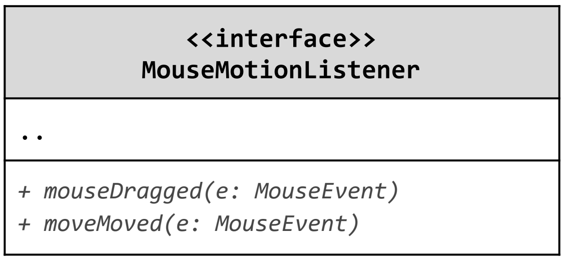
```
<<interface>>
ActionListener
..
+actionPerformed(e: ActionEvent): void
```



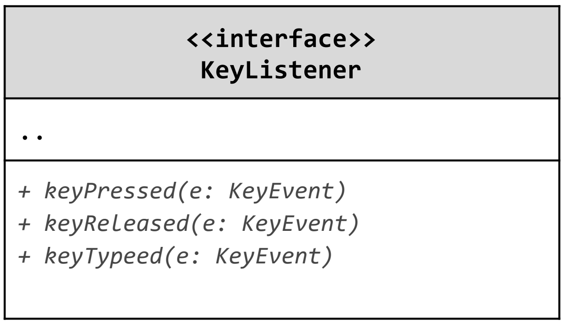
```
<<interface>>
ItemListener
..
+itemStateChanged(e: ItemEvent) :void
```



```
<<interface>>
MouseListener
..
+ mousePressed(e: MouseEvent):void
+ mouseReleased(e: MouseEvent) :void
+ mouseClicked(e: MouseEvent):void
+ mouseExited(e: ouseEvent):void
+ mouseEntered(e: MouseEvent):void
```



```
<<interface>>
MouseMotionListener
..
+ mouseDragged(e: MouseEvent)
+ moveMoved(e: MouseEvent)
```



```
<<interface>>
KeyListener
..
+ keyPressed(e: KeyEvent)
+ keyReleased(e: KeyEvent)
+ keyTypeed(e: KeyEvent)
```

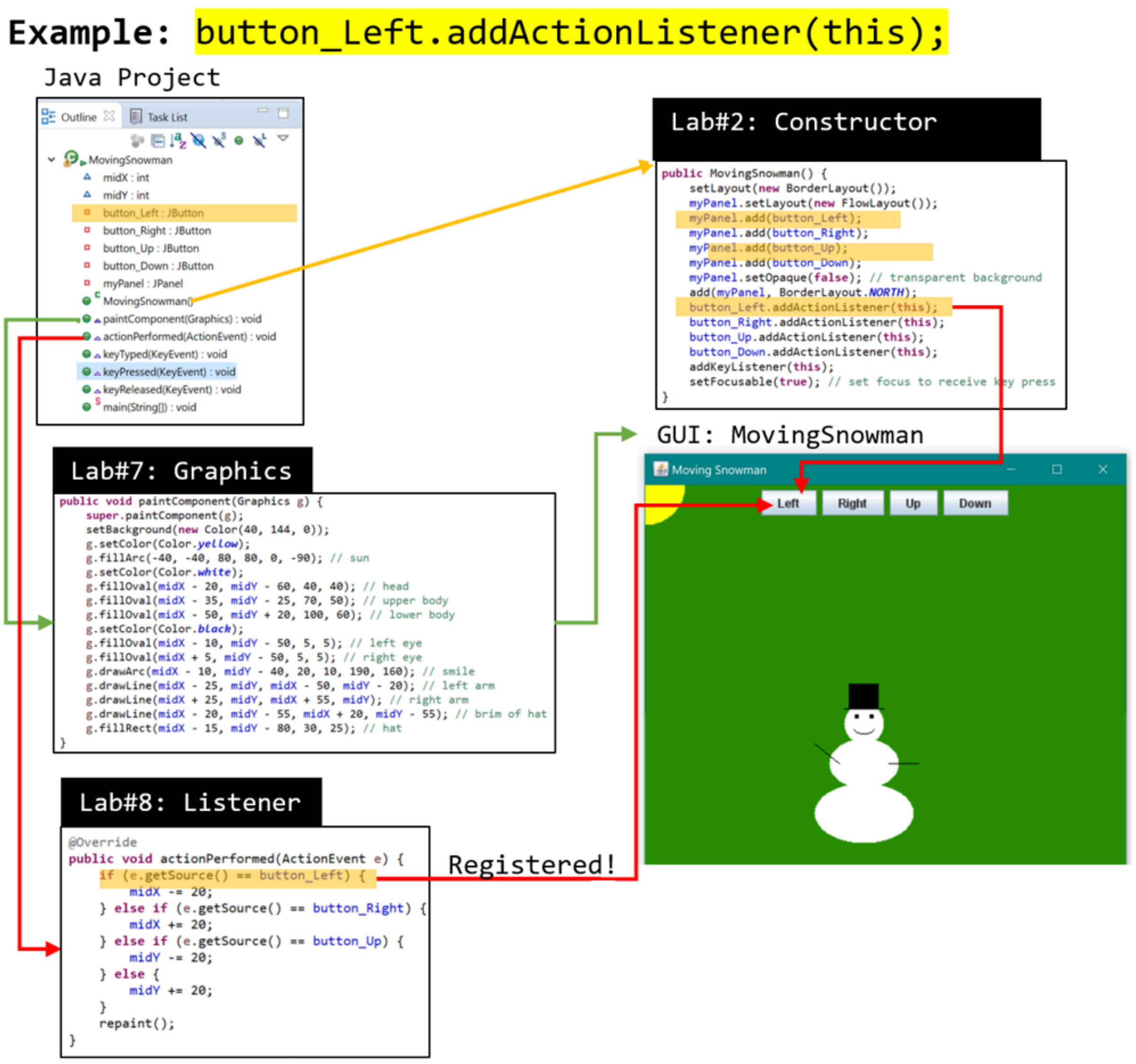
8.5.2 Example: <object>.add<Listener>(this);

☞ To remember type of your object and name, TA suggests to define your variables name as below format:

`<object>_<name>`

For example, `public JButton button_Left = new JButton("Left");`

☞ To explain your TA, student SHOULD try to understand the following example:



Lab#2: Constructor

```java
public MovingSnowman() {
    setLayout(new BorderLayout());
    myPanel.setLayout(new FlowLayout());
    myPanel.add(button_Left);
    myPanel.add(button_Right);
    myPanel.add(button_Up);
    myPanel.add(button_Down);
    myPanel.setOpaque(false); // transparent background
    add(myPanel, BorderLayout.NORTH);
    button_Left.addActionListener(this);
    button_Right.addActionListener(this);
    button_Up.addActionListener(this);
    button_Down.addActionListener(this);
    addKeyListener(this);
    setFocusable(true); // set focus to receive key press
}
```

Lab#7: Graphics

```java
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    setBackground(new Color(40, 144, 0));
    g.setColor(Color.yellow);
    g.fillArc(-40, -40, 80, 80, 0, -90); // sun
    g.setColor(Color.white);
    g.fillOval(midX - 20, midY - 60, 40, 40); // head
    g.fillOval(midX - 35, midY - 25, 70, 50); // upper body
    g.fillOval(midX - 50, midY + 20, 100, 60); // lower body
    g.setColor(Color.black);
    g.fillOval(midX - 10, midY - 50, 5, 5); // left eye
    g.fillOval(midX + 5, midY - 50, 5, 5); // right eye
    g.drawArc(midX - 10, midY - 40, 20, 10, 190, 160); // smile
    g.drawLine(midX - 25, midY, midX - 50, midY - 20); // left arm
    g.drawLine(midX + 25, midY, midX + 55, midY); // right arm
    g.drawLine(midX - 20, midY - 55, midX + 20, midY - 55); // brim of hat
    g.fillRect(midX - 15, midY - 80, 30, 25); // hat
}
```

Lab#8: Listener

```java
@Override
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == button_Left) {
        midX -= 20;
    } else if (e.getSource() == button_Right) {
        midX += 20;
    } else if (e.getSource() == button_Up) {
        midY -= 20;
    } else {
        midY += 20;
    }
    repaint();
}
```

Asst. Prof. Dr. Sasiporn Usanavasin, Dr. Jessada Karnjana, and Dr. Akkarawoot Takhom {sasiporn.us, akkharawoot.aj, jessada.aj}@siit.tu.ac.th

[DES103(Y23S2)-LAB07-EXERCISES]

Students MUST adhere to the lab instructions and regulations provided below.
Please consult your TA to review your completed exercises and submit them on Google Classroom.

Be noticed that for all lab exercises, you need to define your *Java* project as the following name format:

<StudentID>_<Lab number>_<Exercise name>

If your student's ID is 6722300208, the name format of your java project should be:

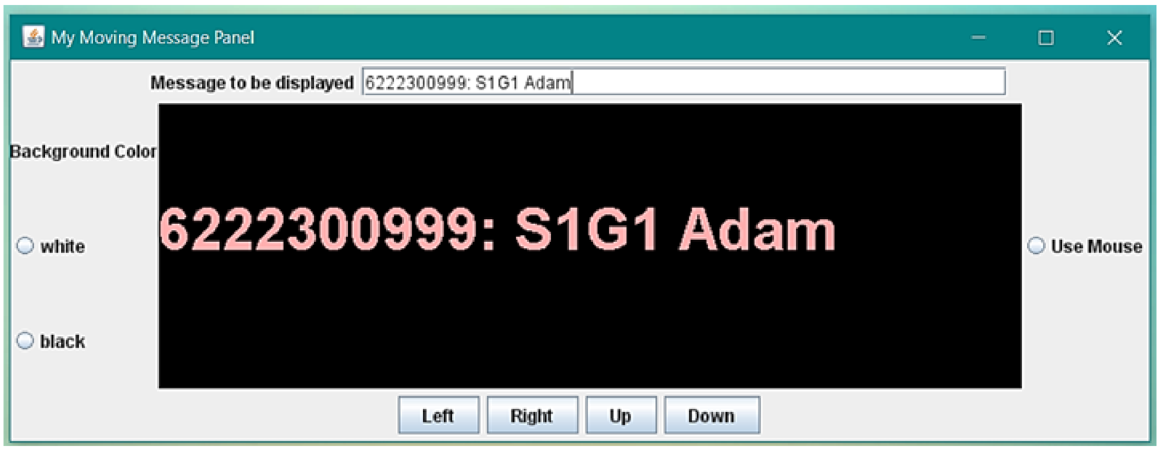
6422300208_LAB08_ MovingMessagePanel for exercise 1,2,3,4 and 5

1. Students MUST explain your understanding to your TA before submitting your finished exercise,
2. Students can attach your finished exercise into Google Class, and click 'Turn-In' when your TA allows you to submit,

For today's exercises, students are going to:

- ❖ draw a panel of moving message by using `paintComponent` and methods of a graphics object learned in class,
- ❖ register appropriate listeners for moving messages.

The final output should look like this:


My Moving Message Panel

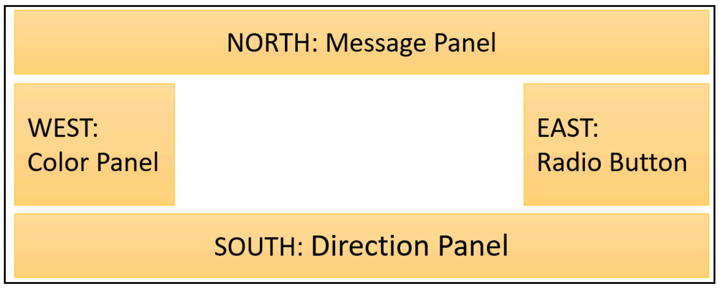
Message to be displayed: 6222300999: S1G1 Adam

6222300999: S1G1 Adam

Exercise 1: (*2 points*)

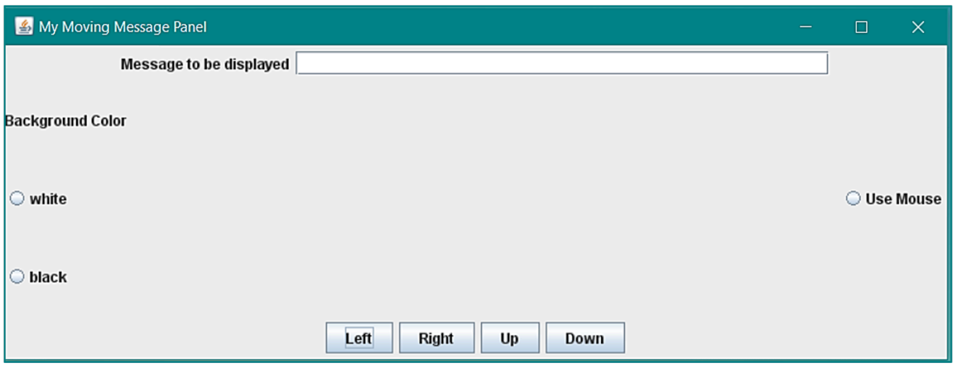
- **Java Project**: <Student_ID>_ LAB08_MovingMessagePanel
- **Objective**: To learn how to use `LayoutManagers` to arrange the components in the Container
- **Instruction**: Write code in the following tasks.

a. Add a new java class `MovingMessagePanel` that makes the following GUI design.



Moving Message Panel

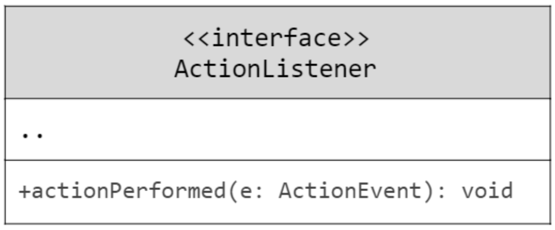
b. Add a new java class `MovingMessagePanelTesting` and write a `main` method for a running output of the `MovingMessagePanel` GUI as shown below:



Exercise 2: (*2 points*)

- **Java Project**: <Student_ID>_ LAB08_MovingMessagePanel
- **Objective**: To learn how to register an appropriate listener to the source, and implement appropriate methods and their details for the specified listener to perform the assigned task.
- **Instruction**: Write code in the following tasks.

a. Make the `MovingMessagePanel` class to be a subclass of the interface `ActionListener`.



b. Register the textfield with itself which acts as the `ActionListener` using an appropriate method.

c. Override the implementation details of the overridden method of `ActionListener`. Your program should get the text from the textfield when the user writes a text into the textfield box and hits enter.

Exercise 3: (*2 points*)

❖ **Java Project**: <Student_ID>_ LAB08_MovingMessagePanel

❖ **Objective**: To learn how to register an appropriate listener to the source, and implement appropriate methods and their details for the specified listener to perform the assigned task.

❖ **Instruction**: Write code in the following tasks.

a. Make the `MovingMessagePanel` class also a subclass of the interface `ItemListener`

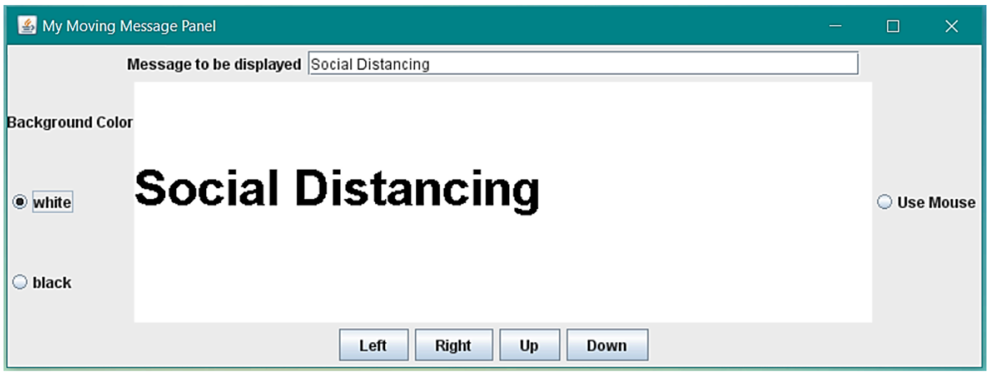
```
<<interface>>
ItemListener
--------------------------------------
..
--------------------------------------
+itemStateChanged(e: ItemEvent) :void
```

b. Register the black and white radio buttons with itself which acts as the `ItemListener` using an appropriate method.

c. Add in the implementation details of the overridden method of `ItemListener`.

d. When the white radio button is selected, your program should change the background of the display panel to `white` and set the font color to `black`.

☞ To check with TA via Line messenger, student SHOULD send the following figure:


My Moving Message Panel

Message to be displayed: Social Distancing

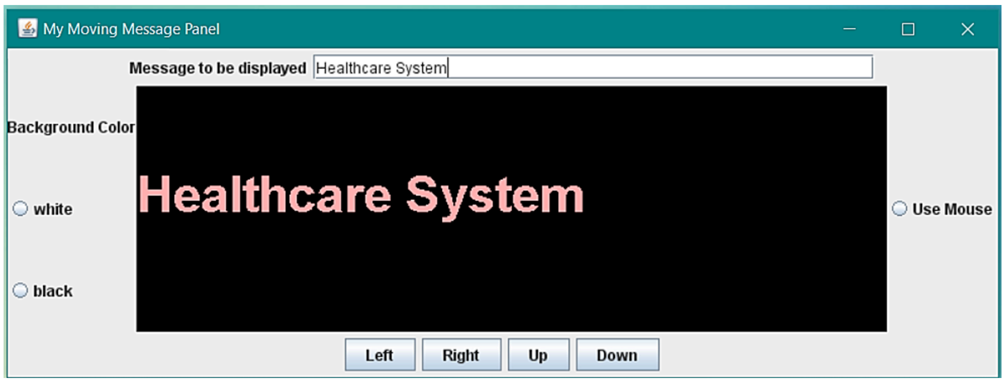
Background Color

Social Distancing

e. When the black radio button is selected, your program should change the background of the display panel to `black` and set the font color to `pink`.

👉 To discuss with TA, student SHOULD send the following figure:

Asst. Prof. Dr. Sasiporn Usanavasin, Dr. Jessada Karnjana, and Dr. Akkarawoot Takhom `{sasiporn.us, akkharawoot.aj, jessada.aj}@siit.tu.ac.th`



Exercise 4: (*2 points*)

- **Java Project**: <Student_ID>_ LAB08_MovingMessagePanel
- **Objective**: To learn how to register an appropriate listener to the source, and implement appropriate methods and their details for the specified listener to perform the assigned task.
- **Suggestion**: To define your variables name as below format:

  `<object>_<name>`

  For example, public `JButton button_Left = new JButton("Left");`

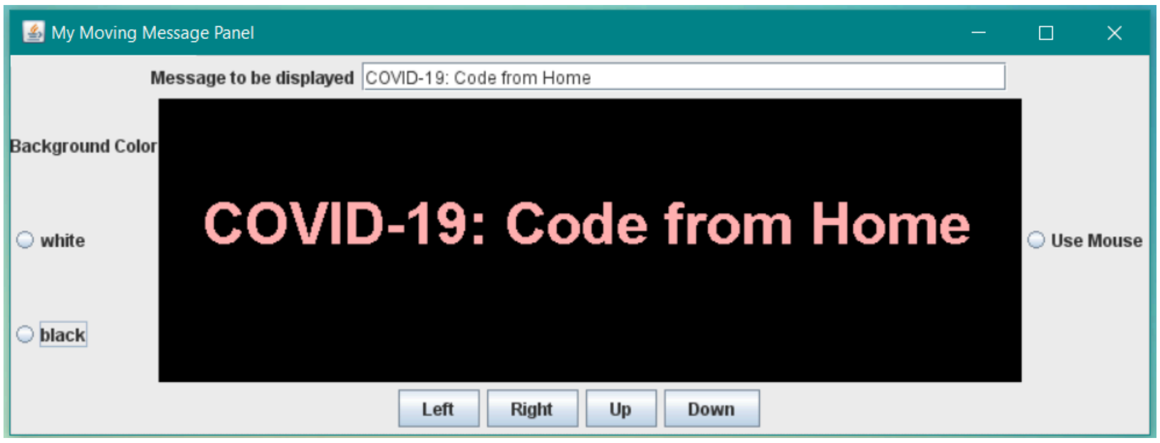
- **Instruction**: Write code in the following tasks that continue from Exercise 3,

  a. Register the four buttons: `Left, Right, Up,` and `Down` with itself which acts as the `ActionListener` using an appropriate method.

  b. Add in the implementation details of the overridden method of `ActionListener`.

  c. Your program should move the message to 4 directions according to corresponding directions from 4 buttons: `Left, Right, Up,` and `Down`.

  ☞ To check with TA via Line messenger, student SHOULD send the following figure:



Exercise 5: (*2 points*)

- **Java Project**: <Student_ID>_ LAB08_MovingMessagePanel
- **Objective**: To learn how to register an appropriate listener to the source, and implement appropriate methods and their details for the specified listener to perform the assigned task.
- **Instruction**: Write code in the following tasks that continue from Exercise 4.

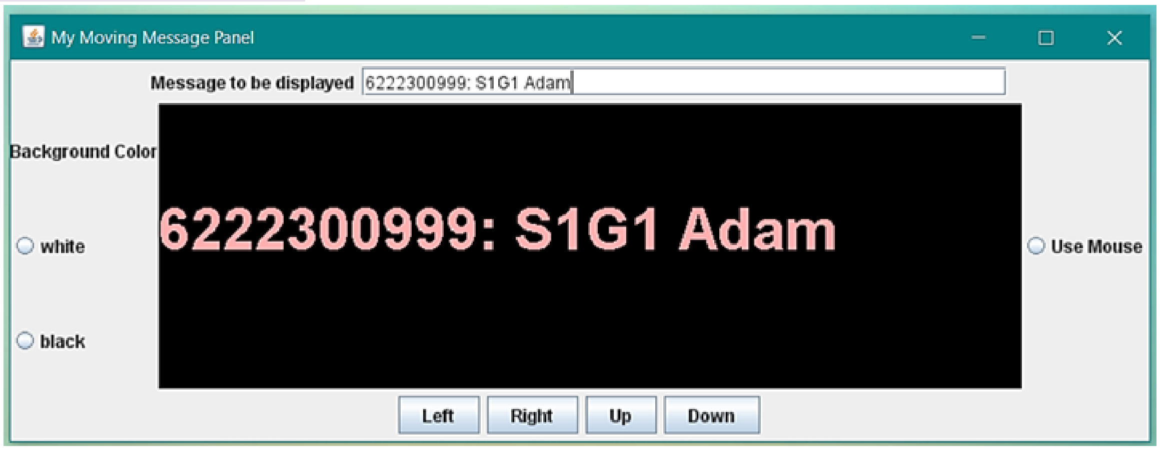
a. Make the `MovingMessagePanel` class also a subclass of the interface `MouseMotionListener`

```
            <<interface>>
         MouseMotionListener
----------------------------------------
..
----------------------------------------
+ mouseDragged(e: MouseEvent)
+ moveMoved(e: MouseEvent)
```

b. Register the display panel(itself) with itself which acts as the `MouseMotionListener` using an appropriate method.

c. Add in the implementation details of the overridden method of `MouseMotionListener`. When the `use-mouse radio button` is selected and the user drags the mouse, your program should move the message at the location of the mouse.

👉 To discuss with TA, student SHOULD send the following figure that displays student ID, section group, and your nickname:


My Moving Message Panel

Message to be displayed: 6222300999: S1G1 Adam

6222300999: S1G1 Adam