Object-Oriented Programming

Lecturer: NGUYEN Thi Thu Trang, trangntt@soict.hust.edu.vn
Teaching Assistants: NGUYEN T.T. Giang, giang.ntt194750@sis.hust.edu.vn
VUONG Dinh An, an.vd180003@sis.hust.edu.vn

Lab 07: GUI Programming with Swing

In this lab, you will practice with:

- Create simple GUI applications with Swing
- Convert part of the Aims Project from the console/command-line (CLI) application to the GUI one.

0 Assignment Submission

For this lab class, you will have to turn in your work twice, specifically:

- **Right after the class**: for this deadline, you should include any work you have done within the lab class.
- **10 PM five days after the class**: for this deadline, you should include your work on all sections of this lab, and push it to a branch called "***release/lab07***" of the valid repository.

After completing all the exercises in the lab, you have to update the use case diagram and the class diagram of the AIMS project.
Each student is expected to turn in his or her work and not give or receive unpermitted aid. Otherwise, we would apply extreme methods for measurement to prevent cheating. Please write down answers for all questions into a text file named "**answers.txt**" and submit it within your repository.

1 Swing components

Note: For the exercises in this lab (excluding the AIMS exercises), you will create a new Java project named `GUIProject`, and put all your source code in a package called "**hust.soict.globalict.swing**" (for ICT) or "**hust.soict.dsai.swing**" (for DS & AI).

In this exercise, we revisit the elements of the Swing API and compare them with those of AWT by implementing the same mini-application using the two libraries. The application is an accumulator which accumulates the values entered by the user and displays the sum.

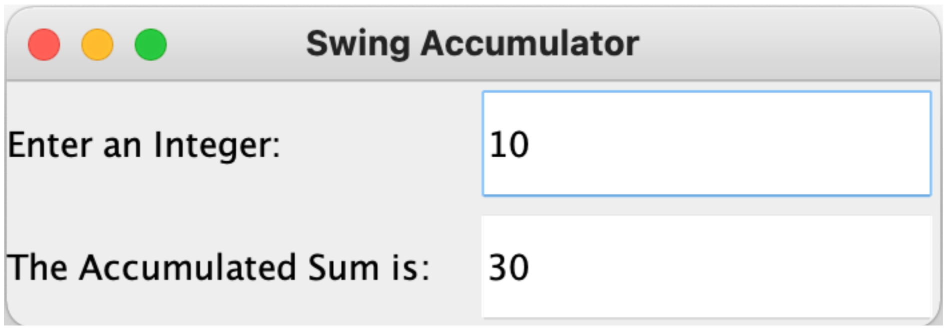
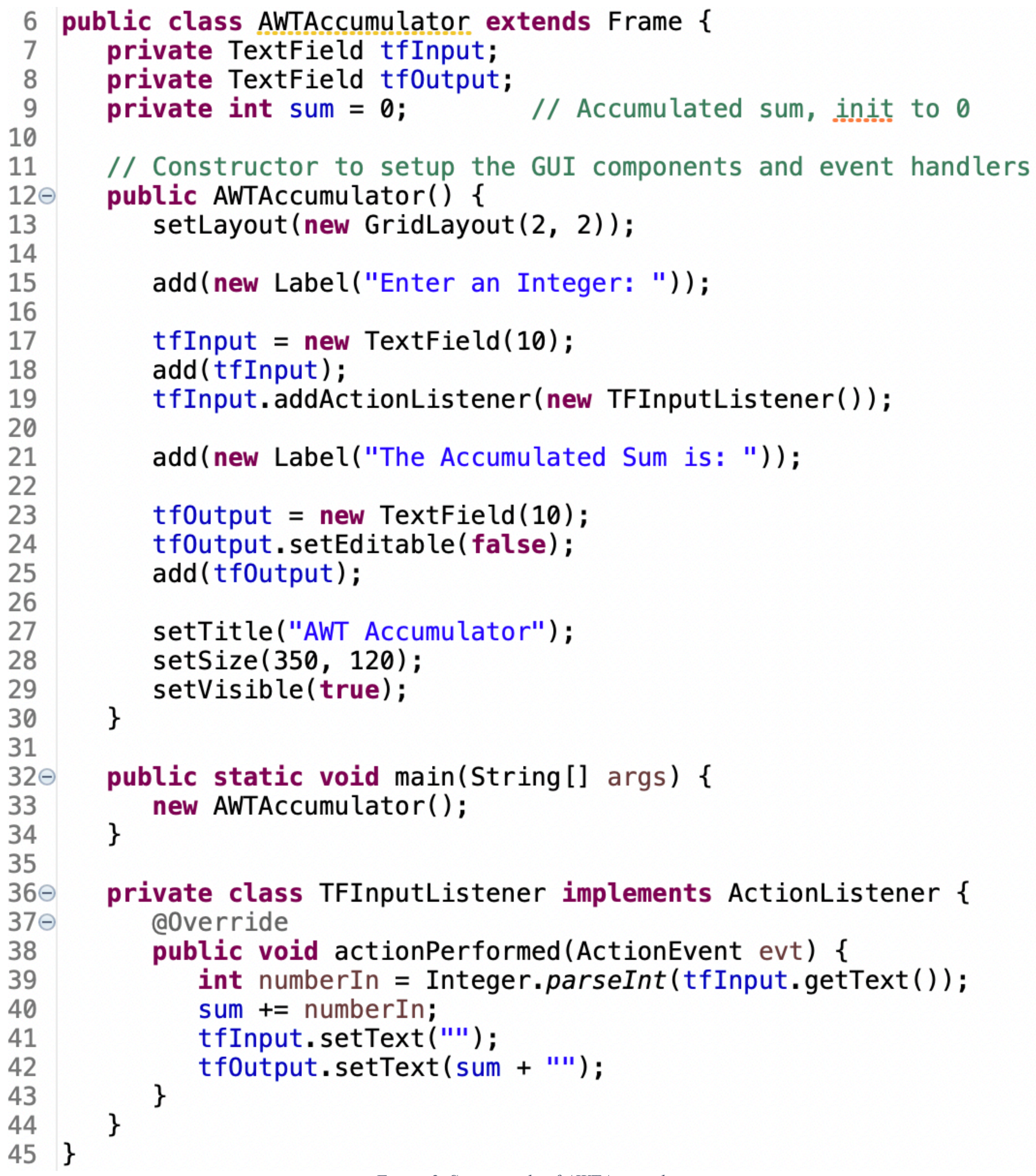


Figure 1. SwingAccumulator

1.1 AWTAccumulator

1.1.1 Create class `AWTAccumulator` with the source code as below



```java
public class AWTAccumulator extends Frame {
   private TextField tfInput;
   private TextField tfOutput;
   private int sum = 0;         // Accumulated sum, init to 0

   // Constructor to setup the GUI components and event handlers
   public AWTAccumulator() {
      setLayout(new GridLayout(2, 2));

      add(new Label("Enter an Integer: "));

      tfInput = new TextField(10);
      add(tfInput);
      tfInput.addActionListener(new TFInputListener());

      add(new Label("The Accumulated Sum is: "));

      tfOutput = new TextField(10);
      tfOutput.setEditable(false);
      add(tfOutput);

      setTitle("AWT Accumulator");
      setSize(350, 120);
      setVisible(true);
   }

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

   private class TFInputListener implements ActionListener {
      @Override
      public void actionPerformed(ActionEvent evt) {
         int numberIn = Integer.parseInt(tfInput.getText());
         sum += numberIn;
         tfInput.setText("");
         tfOutput.setText(sum + "");
      }
   }
}
```

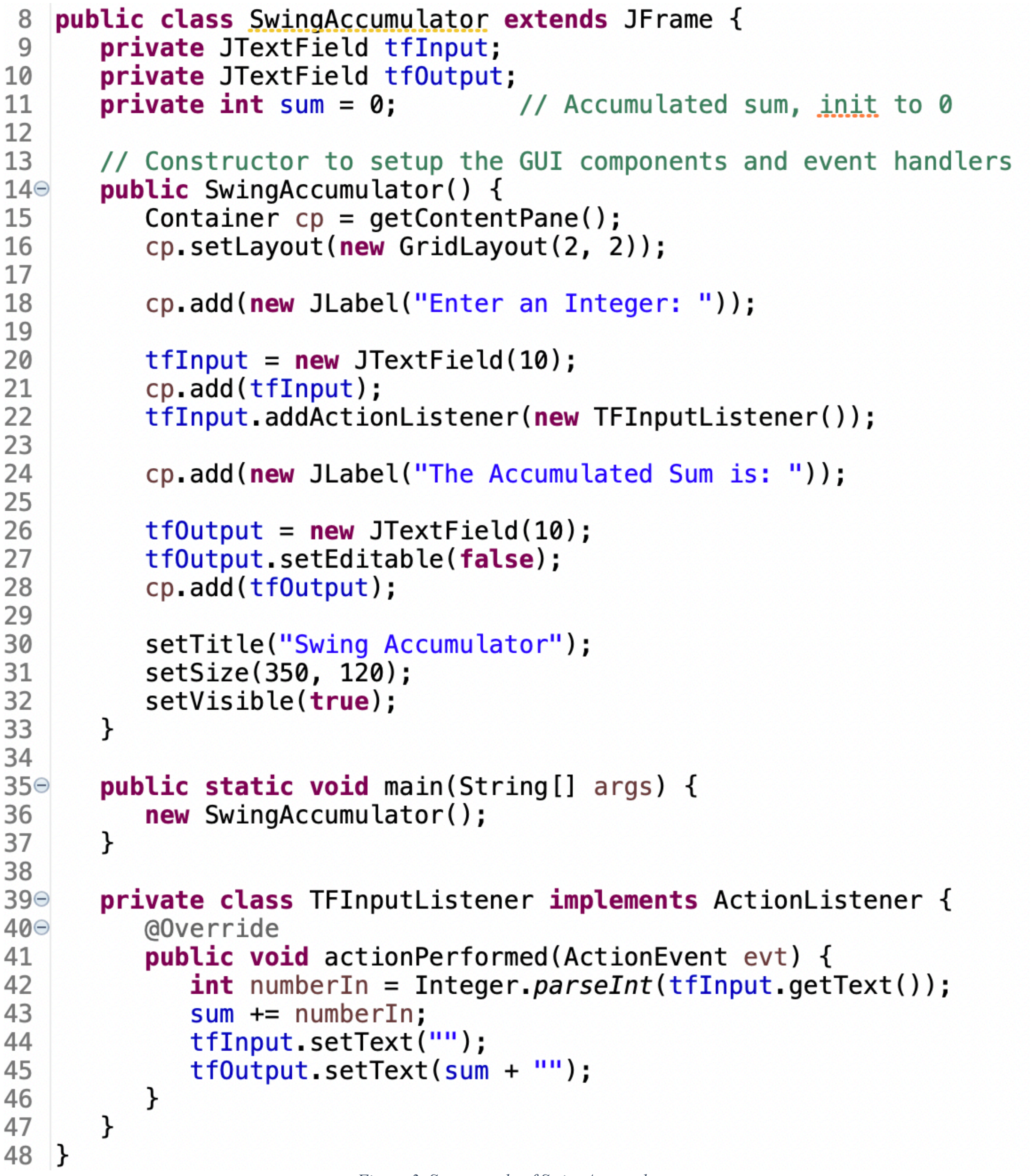
Figure 2. Source code of AWTAccumulator

1.1.2 Explanation

- In AWT, the top-level container is `Frame`, which is inherited by the application class.
- In the constructor, we set up the GUI components in the `Frame` object and the event-handling:
  - In line 13, the layout of the frame is set as `GridLayout`
  - In line 15, we add the first component to our `Frame`, an anonymous `Label`
  - In lines 17-19, we add a `TextField` component to our `Frame`, where the user will enter values. We add a listener which takes this `TextField` component as the source, using a named inner class.
  - In line 21, we add another anonymous `Label` to our `Frame`
  - In lines 23 – 25, we add a `TextField` component to our `Frame`, where the accumulated sum of entered values will be displayed. The component is set to read-only in line 24.
  - In line 27 – 29, the title & size of the `Frame` is set, and the `Frame` visibility is set to true, which shows the `Frame` to us.
- In the listener class (line 36 - 44), the `actionPerformed()` method is implemented, which handles the event when the user hit "Enter" on the source `TextField`.
  - In lines 39-42, the entered number is parsed, added to the sum, and the output `TextField's` text is changed to reflect the new sum.
- In the `main()` method, we invoke the `AWTAccumulator` constructor to set up the GUI

1.2 SwingAccumulator

1.2.1 Create class `SwingAccumulator` with the source code as below:



```java
public class SwingAccumulator extends JFrame {
   private JTextField tfInput;
   private JTextField tfOutput;
   private int sum = 0;          // Accumulated sum, init to 0

   // Constructor to setup the GUI components and event handlers
   public SwingAccumulator() {
      Container cp = getContentPane();
      cp.setLayout(new GridLayout(2, 2));

      cp.add(new JLabel("Enter an Integer: "));

      tfInput = new JTextField(10);
      cp.add(tfInput);
      tfInput.addActionListener(new TFInputListener());

      cp.add(new JLabel("The Accumulated Sum is: "));

      tfOutput = new JTextField(10);
      tfOutput.setEditable(false);
      cp.add(tfOutput);

      setTitle("Swing Accumulator");
      setSize(350, 120);
      setVisible(true);
   }

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

   private class TFInputListener implements ActionListener {
      @Override
      public void actionPerformed(ActionEvent evt) {
         int numberIn = Integer.parseInt(tfInput.getText());
         sum += numberIn;
         tfInput.setText("");
         tfOutput.setText(sum + "");
      }
   }
}
```

Figure 3. Source code of SwingAccumulator

1.2.2 Explanation

- In Swing, the top-level container is `JFrame` which is inherited by the application class.
- In the constructor, we set up the GUI components in the `JFrame` object and the event-handling:

- Not like AWT, the `JComponents` shall not be added onto the top-level container (e.g., `JFrame`, `JApplet`) directly because they are lightweight components. The `JComponents` must be added onto the so-called content-pane of the top-level container. Content-pane is in fact a `java.awt.Container` that can be used to group and layout components.
- In line 15, we get the content pane of the top-level container.
- In line 16, the layout of the content-pane is set as `GridLayout`
- In line 18, we add the first component to our content pane, an anonymous `JLabel`
- In lines 20-22, we add a `JTextField` component to our content-pane, where the user will enter values. We add a listener which takes this `JTextField` component as the source.
- In line 24, we add another anonymous `JLabel` to our content-pane
- In lines 26 – 28, we add a `JTextField` component to our content pane, where the accumulated sum of entered values will be displayed. The component is set to read-only in line 27.
- In line 30 – 32, the title & size of the `JFrame` is set, and the Frame visibility is set to true, which shows the `JFrame` to us.

- In the listener class (lines 39 - 47), the code for event-handling is exactly like the `AWTAccumulator`.
- In the `main()` method, we invoke the `SwingAccumulator` constructor to set up the GUI

1.3 Compare Swing and AWT elements:

Programming with AWT and Swing is quite similar (similar elements including container/components, and event-handling). However, there are some differences that you need to note:

- Compare the top-level containers in Swing and AWT
- Compare the class name of components in AWT and the corresponding class's name in Swing
- Compare the event-handling of Swing and AWT applications

The resulting appearances of the applications developed using Swing and AWT might be different as well. Make comparisons.

2 Change the look & feel of Swing applications

One of the main features of Swing is supporting pluggable look-and-feel (L&F). This means you can choose to change the look and feel of your Swing components. "Look" refers to the appearance of GUI widgets (more formally, `JComponents`) and "feel" refers to the way the widgets behave. We will investigate this feature in the demo below and followed by a small exercise for you to practice

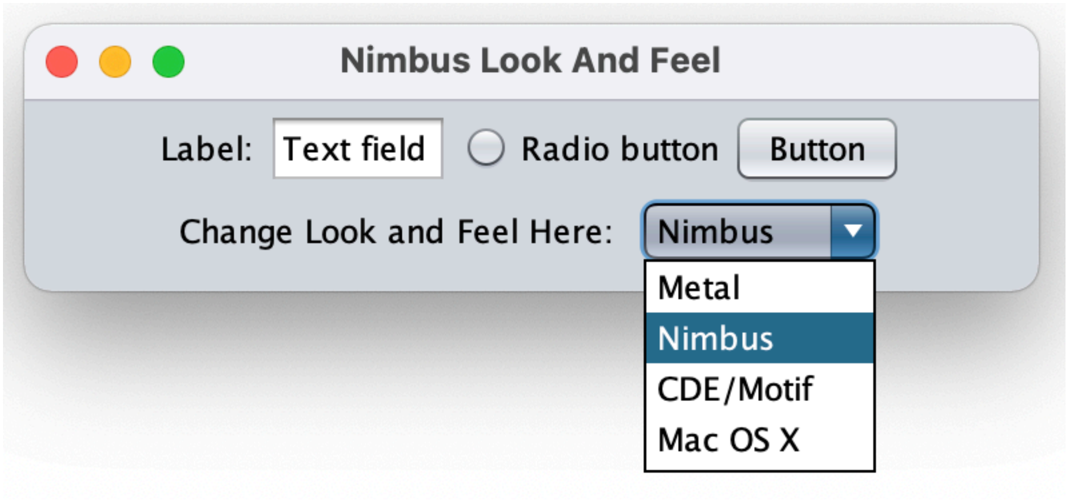


Figure 4. LookAndFeelDemo

2.1 Swing API to change the look and feel:

- `UIManager.setLookAndFeel()`: Sets the L&F used by the `UIManager`. Whenever a Swing component is created, the component asks the UI manager for its L&F.
- `SwingUtilities.updateComponentTreeUI()`: Sets the L&F of the component after it is created as the new one used by `UIManager`.

2.2 Create class `LookAndFeelDemo`:

```
public class LookAndFeelDemo extends JFrame{
    public LookAndFeelDemo() {

        addDemoComponents();
        addLookAndFeelComboBox();

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(380, 100);
        setVisible(true);
    }
```

In the constructor, we set up some demo components in the `addDemoComponents()` method to observe their appearance as the L&F changes. We also add a combo box for users to select among the available L&F in the `addLookAndFeelComboBox()` method.

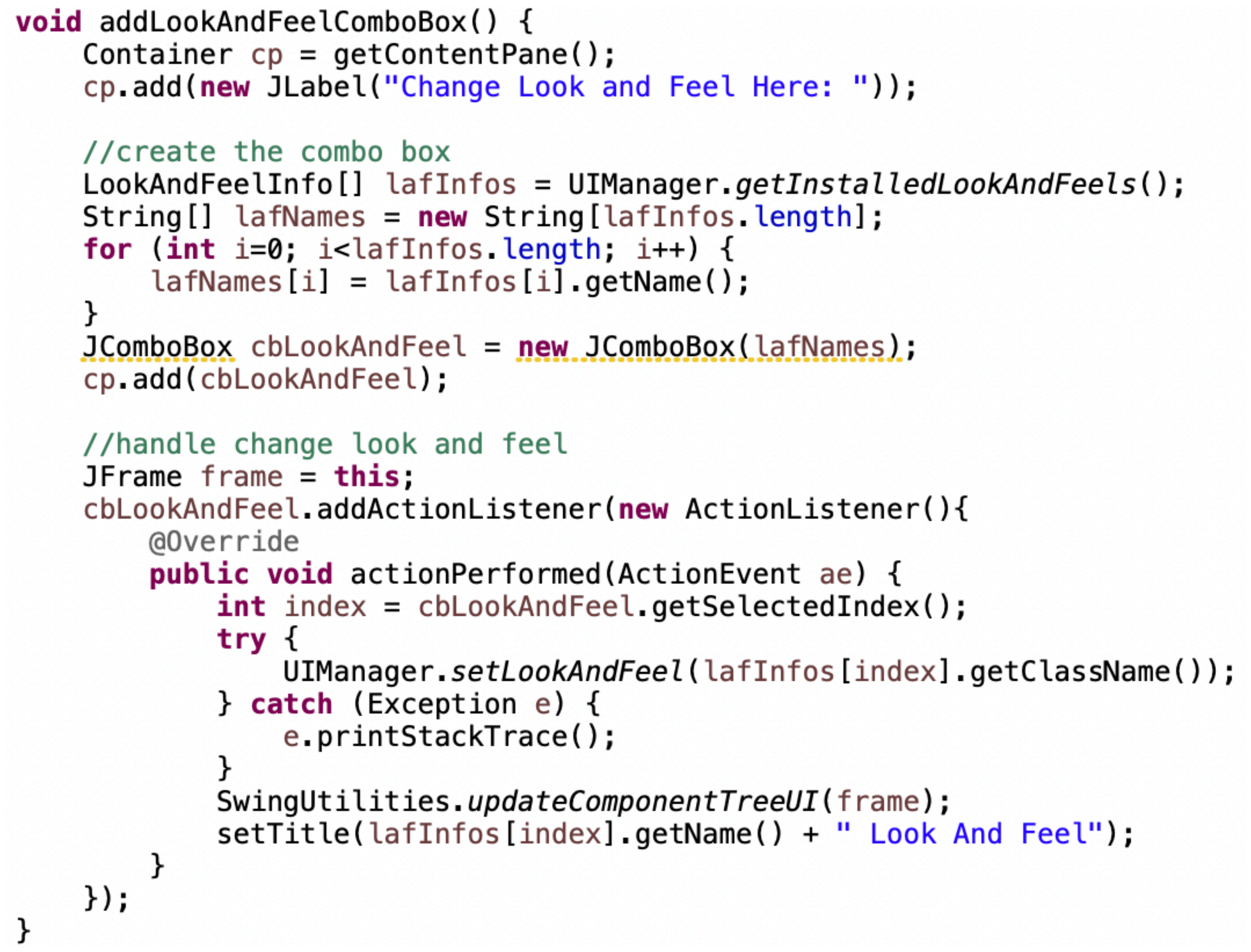
The code for the `addDemoComponents()` method:

```
void addDemoComponents() {
    Container cp = getContentPane();
    cp.setLayout(new FlowLayout());
    cp.add(new JLabel("Label:"));
    cp.add(new JTextField("Text field"));
    cp.add(new JRadioButton("Radio button"));
    cp.add(new JButton("Button"));
}
```

Figure 5. addDemoComponents() source code

Here, we add a label, a text field, a radio button, and a button.

The code for `addLookAndFeelComboBox()` method:



```
void addLookAndFeelComboBox() {
    Container cp = getContentPane();
    cp.add(new JLabel("Change Look and Feel Here: "));

    //create the combo box
    LookAndFeelInfo[] lafInfos = UIManager.getInstalledLookAndFeels();
    String[] lafNames = new String[lafInfos.length];
    for (int i=0; i<lafInfos.length; i++) {
        lafNames[i] = lafInfos[i].getName();
    }
    JComboBox cbLookAndFeel = new JComboBox(lafNames);
    cp.add(cbLookAndFeel);

    //handle change look and feel
    JFrame frame = this;
    cbLookAndFeel.addActionListener(new ActionListener(){
        @Override
        public void actionPerformed(ActionEvent ae) {
            int index = cbLookAndFeel.getSelectedIndex();
            try {
                UIManager.setLookAndFeel(lafInfos[index].getClassName());
            } catch (Exception e) {
                e.printStackTrace();
            }
            SwingUtilities.updateComponentTreeUI(frame);
            setTitle(lafInfos[index].getName() + " Look And Feel");
        }
    });
}
```

Figure 6. addLookAndFeelComboBox() source code

Here, we create an array of all the installed L&Fs and create a `JComboBox` which has its entries corresponding to the array of L&Fs.
Then, we add a listener for the `JComboBox` (we use an anonymous inner class for this listener). In the event-handling code, we get the selected L&F, set the `UIManager` to use it, and update the top-level container, which will subsequently update the L&F of all its children.
The last step is to create a `main()` method, which should invoke the constructor of `LookAndFeelDemo`.

*2.3 Extend the class **`LookAndFeelDemo`** above:*

Among the installed L&Fs, there are two special L&F provided by the Java JRE:

- `CrossPlatformLookAndFeel` — this is the "Java L&F" (also called "Metal") that looks the same on all platforms. It is part of the Java API (`javax.swing.plaf.metal`) and is the default that will be used if you do nothing in your code to set a different L&F.
- `SystemLookAndFeel` — here, the application uses the L&F that is native to the system it is running on. The System L&F is determined at runtime, where the application asks the system to return the name of the appropriate L&F.

Extend the `LookAndFeelDemo` program above to add two new options ("Java" and "System") for the user to select these two L&F.
<u>Hint</u>: The API for getting the cross-platform and system look and feel are:
`UIManager.getCrossPlatformLookAndFeelClassName()`
`UIManager.getSystemLookAndFeelClassName()`

3 Organizing Swing components with Layout Managers

In Swing, there are two groups of GUI classes, the containers and the components. We have worked with several component classes in previous exercises. Now, we will investigate more on the containers.

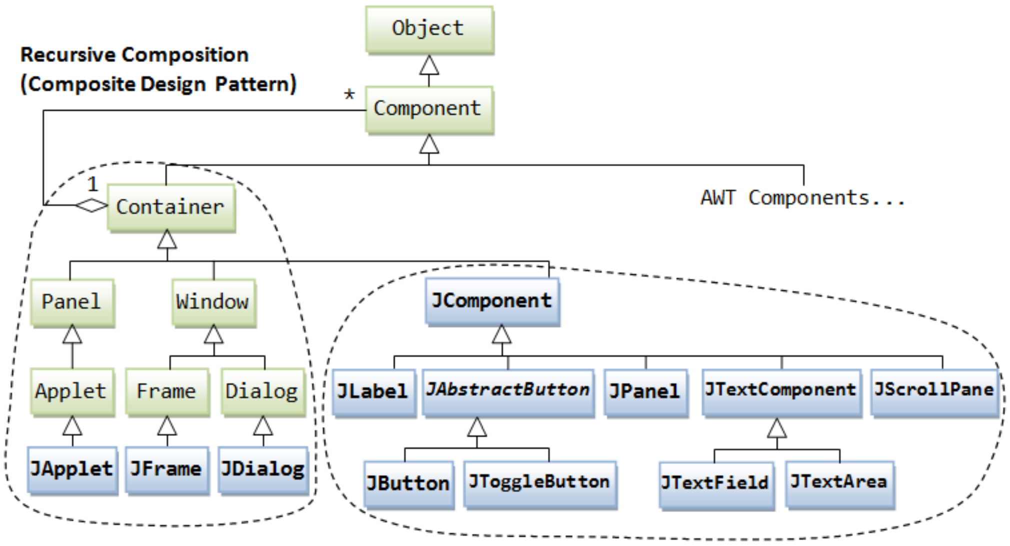


Figure 7. AWT and Swing elements

3.1 Swing top-level and secondary-level containers:

A container is used to hold components. A container can also hold containers because it is a (subclass of) component. Swing containers are divided into top-level and secondary-level containers:

- Top-level containers:
    - `JFrame`: used for the application's main window (with an icon, a title, minimize/maximize/close buttons, an optional menu bar, and a content pane)
    - `JDialog`: used for a secondary pop-up window (with a title, a close button, and a content pane).
    - `JApplet`: used for the applet's display area (content pane) inside a browser's window
- Secondary-level containers can be used to group and layout relevant components (most commonly used is `JPanel`)

3.2 *Using JPanel as a secondary-level container to organize components:*

3.2.1 Create class `NumberGrid`:

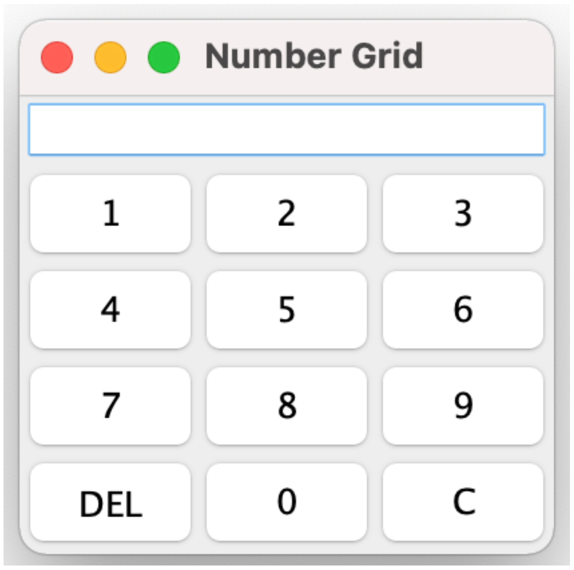


Figure 8. NumberGrid

This class allows us to input a number digit-by-digit from a number grid into a text field display. We can also delete the latest digit or delete the entire number and start over.

```java
public class NumberGrid extends JFrame{
    private JButton[] btnNumbers = new JButton[10];
    private JButton btnDelete, btnReset;
    private JTextField tfDisplay;

    public NumberGrid() {

        tfDisplay = new JTextField();
        tfDisplay.setComponentOrientation(
                ComponentOrientation.RIGHT_TO_LEFT);

        JPanel panelButtons = new JPanel(new GridLayout(4, 3));
        addButtons(panelButtons);

        Container cp = getContentPane();
        cp.setLayout(new BorderLayout());
        cp.add(tfDisplay, BorderLayout.NORTH);
        cp.add(panelButtons, BorderLayout.CENTER);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setTitle("Number Grid");
        setSize(200, 200);
        setVisible(true);
    }
```

Figure 9. NumberGrid source code(1)

The class has several attributes:

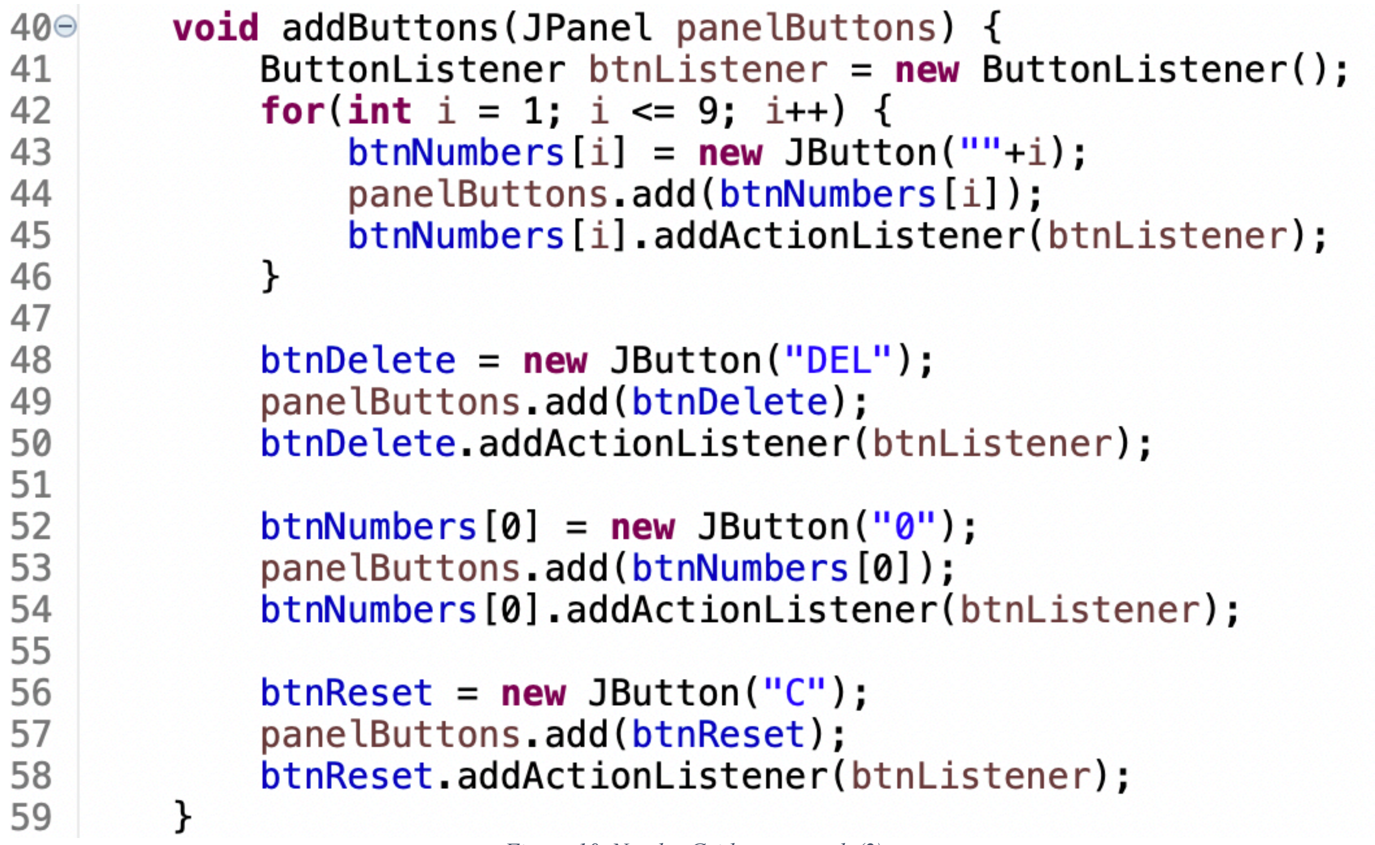
- The `btnNumbers` array for the digit buttons
- The `btnDelete` for the DEL button
- The `btnReset` for the C button
- The `tfDisplay` for the top display

In the constructor, we add two components to the content pane of the `JFrame`:

- A `JTextField` for the display text field
- A `JPanel`, which will group all of the buttons and put them in a grid layout

3.2.2 Adding buttons

We add the buttons to the `panelButtons` in the `addButtons()` method:



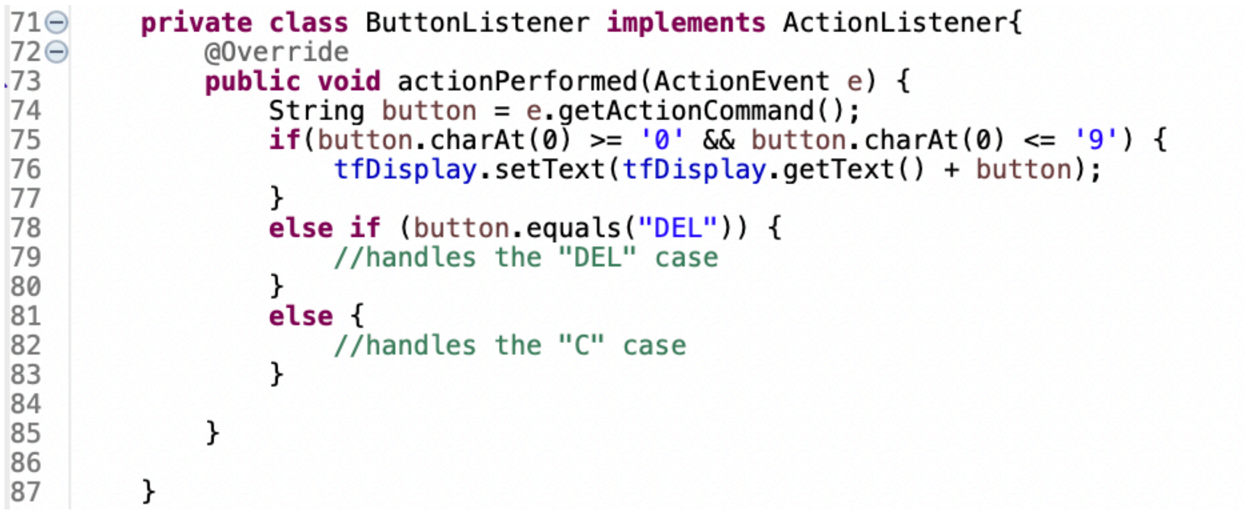
```java
    void addButtons(JPanel panelButtons) {
        ButtonListener btnListener = new ButtonListener();
        for(int i = 1; i <= 9; i++) {
            btnNumbers[i] = new JButton(""+i);
            panelButtons.add(btnNumbers[i]);
            btnNumbers[i].addActionListener(btnListener);
        }

        btnDelete = new JButton("DEL");
        panelButtons.add(btnDelete);
        btnDelete.addActionListener(btnListener);

        btnNumbers[0] = new JButton("0");
        panelButtons.add(btnNumbers[0]);
        btnNumbers[0].addActionListener(btnListener);

        btnReset = new JButton("C");
        panelButtons.add(btnReset);
        btnReset.addActionListener(btnListener);
    }
```

Figure 10. NumberGrid source code(2)

The buttons share the same listener of the `ButtonListener` class, which is a named inner class.

3.2.3 Complete inner class `ButtonListener`

Your task is to complete the implementation of the `ButtonListener` class below:



```java
    private class ButtonListener implements ActionListener{
        @Override
        public void actionPerformed(ActionEvent e) {
            String button = e.getActionCommand();
            if(button.charAt(0) >= '0' && button.charAt(0) <= '9') {
                tfDisplay.setText(tfDisplay.getText() + button);
            }
            else if (button.equals("DEL")) {
                //handles the "DEL" case
            }
            else {
                //handles the "C" case
            }

        }

    }
```

Figure 11. NumberGrid source code(3)

In the `actionPerformed()` method, we will handle the button pressed event. Since we have many sources, we need to determine which source is firing the event (which button is pressed) and handle each case accordingly (change the text of the display text field). Here, we have three cases:

- A digit button: a digit is appended to the end
- DEL button: delete the last digit
- C button: clears all digits

The code for the first case is there for reference, you need to implement it by yourself in the remaining two cases.

4 Create a graphical user interface for AIMS with Swing

Additional Requirement: From this lab, we will split the AIMS system into two applications according to the role of the user: Customer and Store Manager

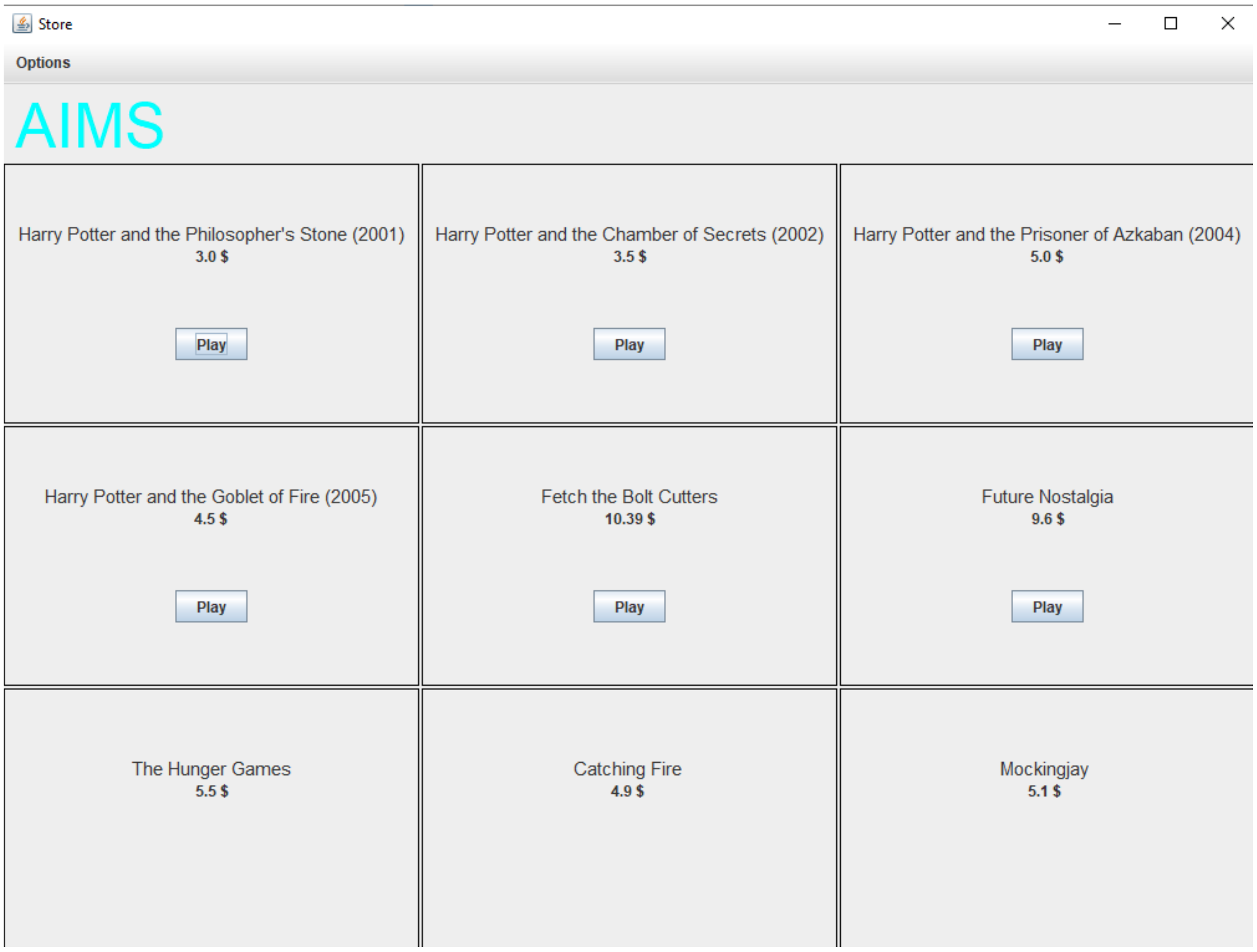
- Store Manager can view and update the store (i.e: Add Book, Add CD, Add DVD).
- Customer can view the store as the store manager, but they can choose to add medias to the cart. Customer can also perform some functions related to the cart.

In this lab, we will implement the application for Store Manager. The application for Customer will be implemented in the next lab.

Please put the source code in this exercises in the "**hust.soict.globalict.aims.screen.manager**" package (for ICT) or the "**hust.soict.dsai.aims.screen.manager**" package (for DS&AI).

4.1 View Store Screen

After starting the application for Store Manager, the user will see a view store screen as below:



Harry Potter and the Philosopher's Stone (2001)
3.0 $

Harry Potter and the Chamber of Secrets (2002)
3.5 $

Harry Potter and the Prisoner of Azkaban (2004)
5.0 $

Harry Potter and the Goblet of Fire (2005)
4.5 $

Fetch the Bolt Cutters
10.39 $

Future Nostalgia
9.6 $

The Hunger Games
5.5 $

Catching Fire
4.9 $

Mockingjay
5.1 $

Figure 126. View Store Screen of Store Manager

For the view store screen, we will use the `BorderLayout`:
In the NORTH component, there will be the menu bar and the header
In the CENTER component, there will be a panel that uses the `GridLayout`, each cell is an item in the store.

4.1.1 Create the `StoreManagerScreen` class:

```java
public class StoreManagerScreen{
    private Store store;
```

Figure 137. Declaration of StoreManagerScreen class

This will be our view store screen while logging in with the role Store Manager.
Declare one attribute in the `StoreManagerScreen` class: `Store store`. This is because we need information on the items in the store to display them.

4.1.2 The `NORTH` component:

Create the method createNorth(), which will create our NORTH component:

```java
JPanel createNorth() {
    JPanel north = new JPanel();
    north.setLayout(new BoxLayout(north, BoxLayout.Y_AXIS));
    north.add(createMenuBar());
    north.add(createHeader());
    return north;
}
```

Figure 148. createNorth() source code

Create the method `createMenuBar()`:

```java
JMenuBar createMenuBar() {
    JMenu menu = new JMenu("Options");

    menu.add(new JMenuItem("View store"));

    JMenu smUpdateStore = new JMenu("Update Store");
    smUpdateStore.add(new JMenuItem("Add Book"));
    smUpdateStore.add(new JMenuItem("Add CD"));
    smUpdateStore.add(new JMenuItem("Add DVD"));
    menu.add(smUpdateStore);

    JMenuBar menuBar = new JMenuBar();
    menuBar.setLayout(new FlowLayout(FlowLayout.LEFT));
    menuBar.add(menu);

    return menuBar;
}
```

Figure 159. createMenuBar() source code

The resulting menu bar will look something like this:

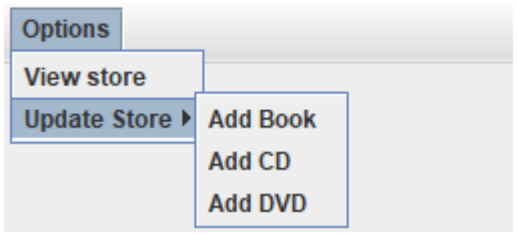
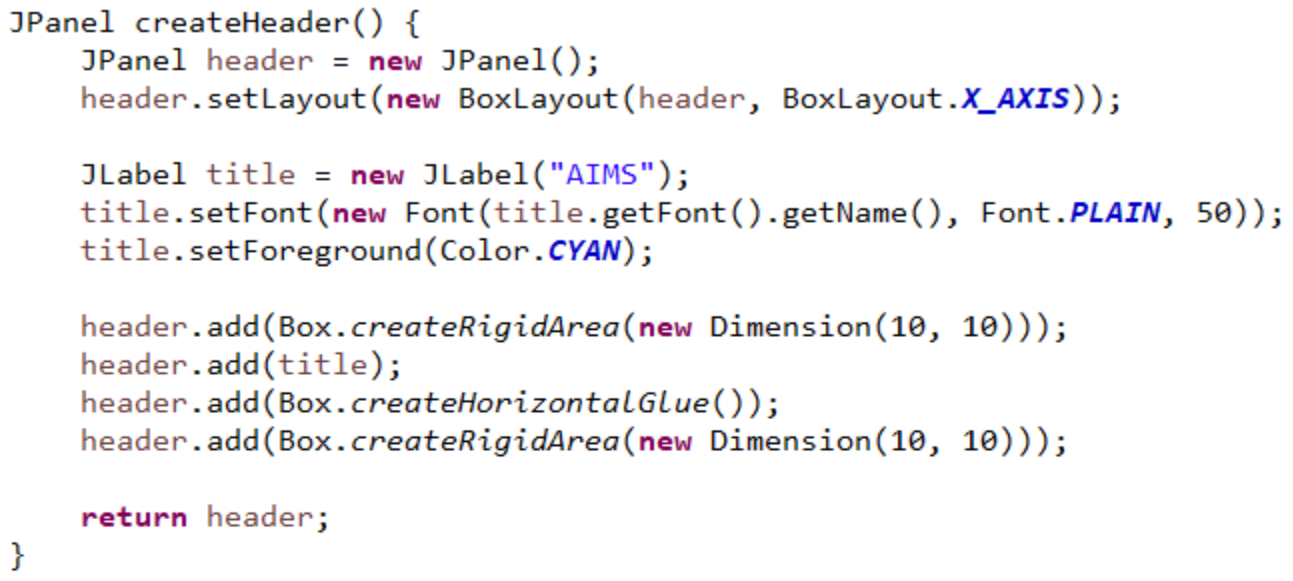


Figure 2016. The resulting menu on the menu bar

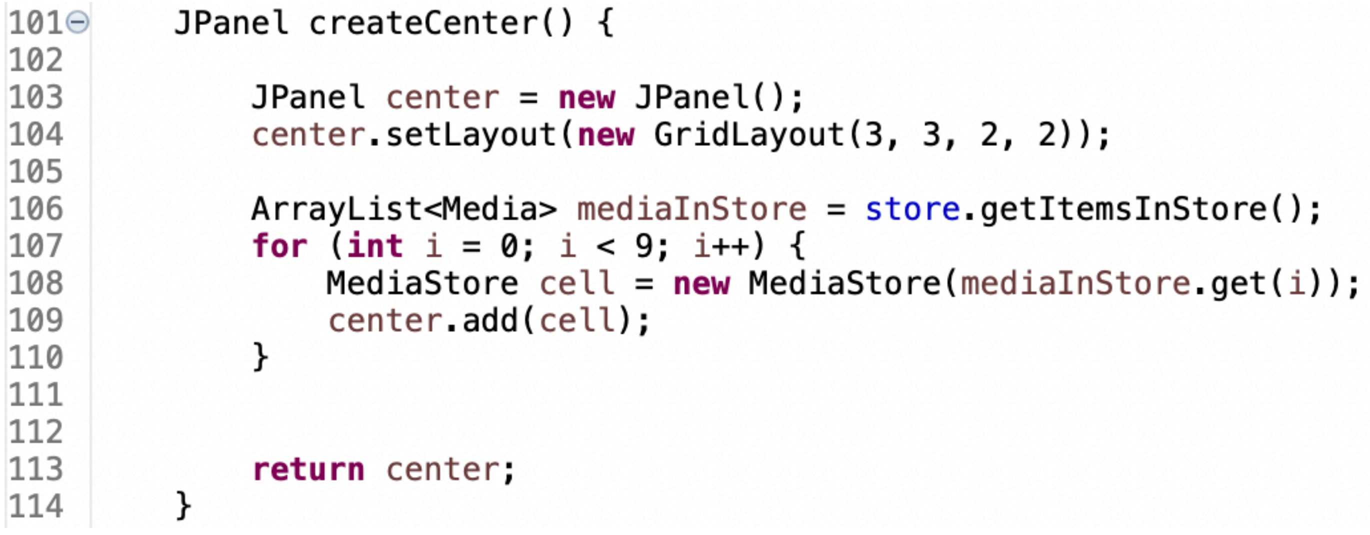
Create the method `createHeader()`:



```java
JPanel createHeader() {
    JPanel header = new JPanel();
    header.setLayout(new BoxLayout(header, BoxLayout.X_AXIS));

    JLabel title = new JLabel("AIMS");
    title.setFont(new Font(title.getFont().getName(), Font.PLAIN, 50));
    title.setForeground(Color.CYAN);

    header.add(Box.createRigidArea(new Dimension(10, 10)));
    header.add(title);
    header.add(Box.createHorizontalGlue());
    header.add(Box.createRigidArea(new Dimension(10, 10)));

    return header;
}
```

Figure 21. createHeader() source code

4.1.3 The `CENTER` component:



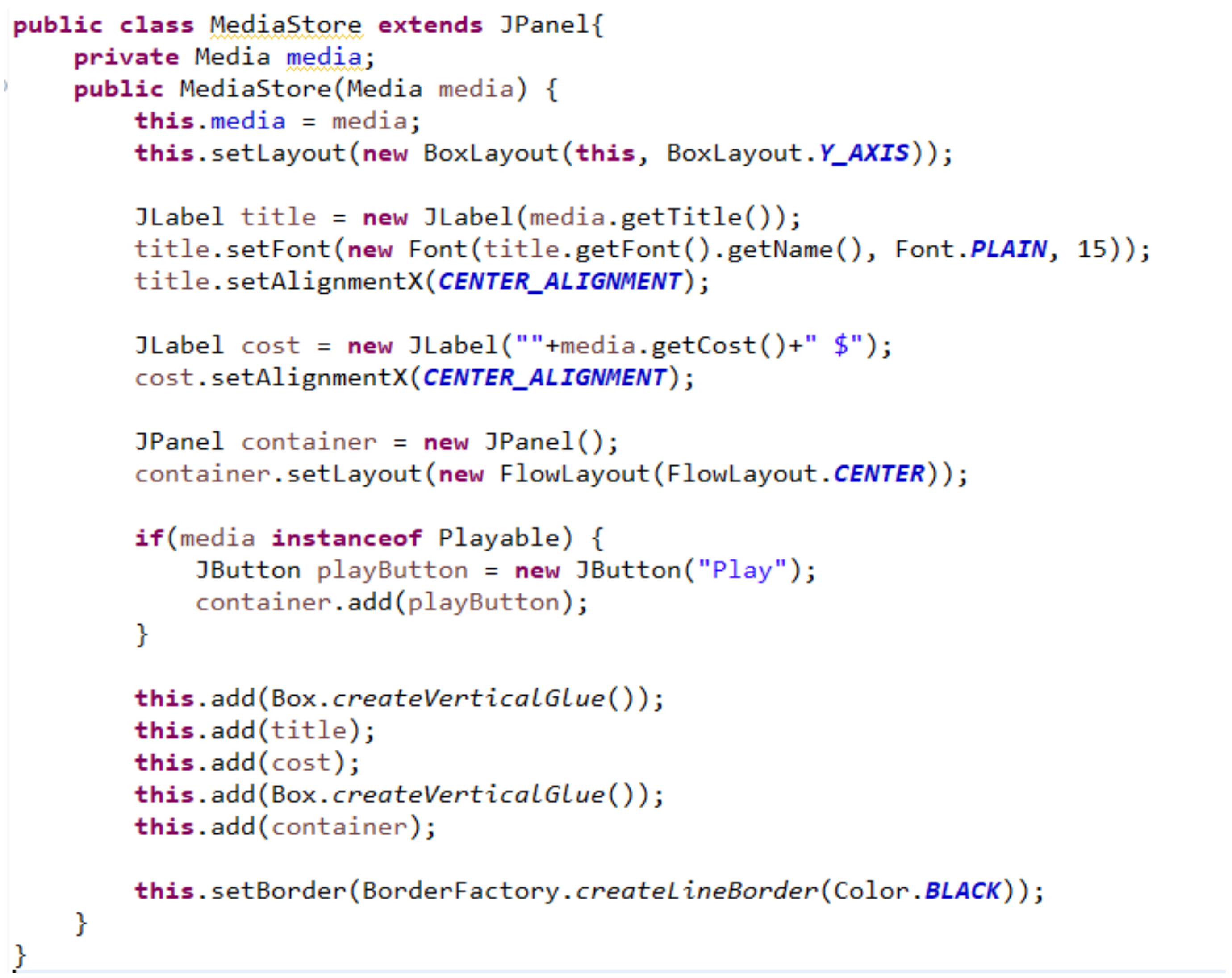
```java
JPanel createCenter() {

    JPanel center = new JPanel();
    center.setLayout(new GridLayout(3, 3, 2, 2));

    ArrayList<Media> mediaInStore = store.getItemsInStore();
    for (int i = 0; i < 9; i++) {
        MediaStore cell = new MediaStore(mediaInStore.get(i));
        center.add(cell);
    }


    return center;
}
```

Figure 22. createCenter() source code

Here, we see that each cell is an object of the class `MediaStore`, which represents the GUI element for a `Media` in the Store Screen.

4.1.4 The `MediaStore` class:

Here, since the `MediaStore` is a GUI element, it extends the `JPanel` class. It has one attribute: `Media media`.



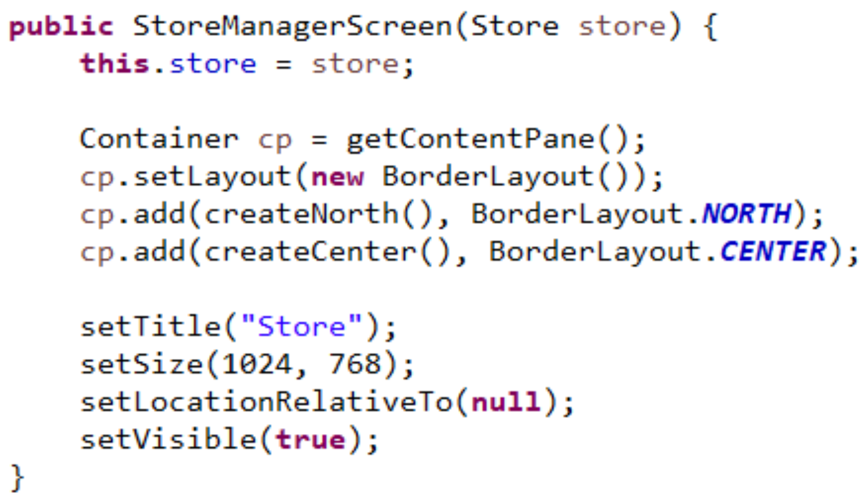
```java
public class MediaStore extends JPanel{
    private Media media;
    public MediaStore(Media media) {
        this.media = media;
        this.setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));

        JLabel title = new JLabel(media.getTitle());
        title.setFont(new Font(title.getFont().getName(), Font.PLAIN, 15));
        title.setAlignmentX(CENTER_ALIGNMENT);

        JLabel cost = new JLabel(""+media.getCost()+" $");
        cost.setAlignmentX(CENTER_ALIGNMENT);

        JPanel container = new JPanel();
        container.setLayout(new FlowLayout(FlowLayout.CENTER));

        if(media instanceof Playable) {
            JButton playButton = new JButton("Play");
            container.add(playButton);
        }

        this.add(Box.createVerticalGlue());
        this.add(title);
        this.add(cost);
        this.add(Box.createVerticalGlue());
        this.add(container);

        this.setBorder(BorderFactory.createLineBorder(Color.BLACK));
    }
}
```

Figure 23. MediaStore source code

Note how the code checks if the `Media` implements the `Playable` interface to create a "Play" button.

4.1.5 Putting it all together:

Finally, we have all the component methods to use in the constructor of `StoreScreen`:



```java
public StoreManagerScreen(Store store) {
    this.store = store;

    Container cp = getContentPane();
    cp.setLayout(new BorderLayout());
    cp.add(createNorth(), BorderLayout.NORTH);
    cp.add(createCenter(), BorderLayout.CENTER);

    setTitle("Store");
    setSize(1024, 768);
    setLocationRelativeTo(null);
    setVisible(true);
}
```

Figure 24. StoreManagerScreen constructor source code

Add main method to start the application for Store Manager.

4.2 Update Store Screen

We have successfully set up all the components for our store, but they are just static – buttons and menu items don't respond when clicked. Now, it's your task to implement the handling of the event when the user interacts with:

- The menu bar:
  - When user clicks on one of the items of the "Update Store" menu on the menu bar (such as "Add Book", "Add CD", or "Add DVD"), the application should switch to an appropriate new screen for the user to add the new item. This screen should have the same menu bar as the View Store Screen, so the user can go back to view the store.
  - When the user clicks on "View Store", the application should switch to the View Store Screen.
- The buttons on MediaHome:
  - When the user clicks on the "Play" button, the Media should be played in a dialog window. You can use JDialog here.

Note:

- For simplicity:
  - You only need to do the "Add item to store" screen, the "Remove item from store" is omitted. As shown above, there is no option of "Remove item from store" in the "Update Store" menu.
  - In the "Add item to store" Screen, you don't need to do data type validation yet.

Suggestion: You might need to create more screens for this task. Make other modifications to the source code above as needed. You should create 3 classes `AddDigitalVideoDiscToStoreScreen`, `AddCompactDiscToStoreScreen`, and `AddBookToStoreScreen`, which will take the necessary inputs from user. Since these GUI classes all share part of their interface, you can apply Inheritance here and create a parent `AddItemToStoreScreen` class and let the above three classes inherit from it.