Menus

- Swing supports pull-down menus to create a GUI applications
- It contains **menubar** at the top which contains pull-down menus. When it is clicked it opens **menuitems** and **submenus**
- When user clicks on a **menuitem**, all menus are closed and message is sent to program

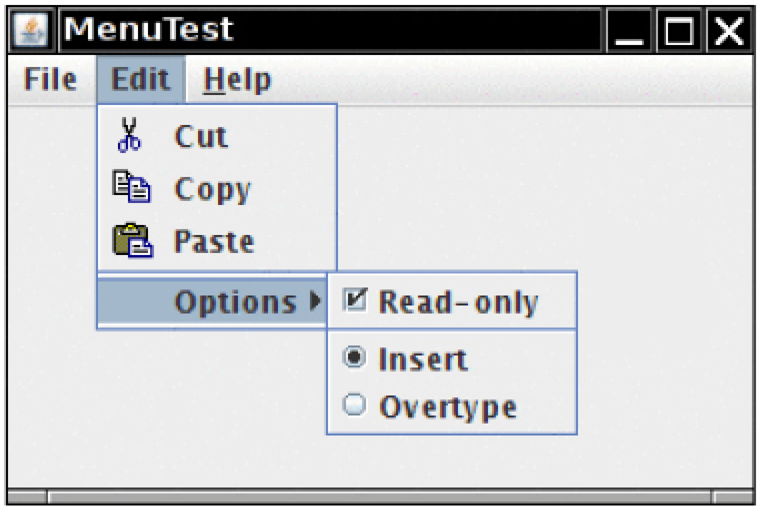


Figure: A menu with a submenu

Building a menu

- First create a menubar
  JMenuBar mb = new JMenuBar();

- Add menubar at the top of a frame using setJMenuBar method
  frame.setJMenuBar(mb)

- For each menu, create a menu object
  JMenu fileMenu = new JMenu("File");
  JMenu editMenu = new JMenu("Edit");
  JMenu helpMenu = new JMenu("Help");

- Add top-level menus to the menu bar
  mb.add(fileMenu);
  mb.add(editMenu);
  mb.add(helpMenu);

- Add menu items, separators and submenus to the menu object

```
JMenuItem cutMenuItem = new JMenuItem(“Cut”);
JMenuItem copyMenuItem = new JMenuItem(“Copy”);
JMenuItem pasteMenuItem = new JMenuItem(“Paste”);
JMenuItem optionsMenuItem = new JMenuItem(“Options”);
```

- We can add listener on menu items to capture events like

```
exitMenuItem.addActionListener(listener);
```

- To add submenus inside options menu we can do the following

```
JMenu optionsMenu = new JMenu("Options");

JCheckBoxMenuItem readOnly = new JCheckBoxMenuItem("Read-only");
readOnly.setSelected(true);
JRadioButtonMenuItem insert = new JRadioButtonMenuItem("Insert");
insert.setSelected(true);
JRadioButtonMenuItem overtype = new JRadioButtonMenuItem("Overtype");

optionsMenu.add(readOnly);
optionsMenu.add(insert);
optionsMenu.add(overtype);
```

- Finally add menuitems inside edit menu along with separator

```
editMenu.add(cutMenuItem);
editMenu.add(copyMenuItem);
editMenu.add(pasteMenuItem);
editMenu.addSeparator();
editMenu.add(optionsMenu);
```

- Some other methods

  `JMenuItem insert(JMenuItem menu, int index)`
  adds a new menu item (or submenu) to the menu at a specific index.

  `void insertSeparator(int index)`
  adds a separator to the menu.

  `void remove(JMenuItem item)`
  removes a specific item from the menu.

JMenuItem(Action a)
constructs a menu item for the given action.

void setAction(Action a)
sets the action for this button or menu item.

void setJMenuBar(JMenuBar menubar)
sets the menu bar for this frame.

Icons in MenuItems

- JMenuItem extends the AbstractButton so they are very similar to buttons

- Just like buttons, menus can have label, icon or both.

- We can specify the icon with
  JMenuItem(String, Icon)
  Or
  JMenuItem(Icon)

- Example:
  ```
  JMenuItem cut = new JMenuItem(“Cut”,new ImageIcon(“cut.jpg”);
  ```

- By default menuitem is placed right to the icon, we can call setHorizontalTextPosition method to place the menuitem left
  ```
  cut.setHorizontalTextPosition(SwingConstants.LEFT);
  ```

- We can define AbstractAction as follows:
  ```
  cutAction = new AbstractAction(“Cut”, new ImageIcon(“cut.jpg”)){
          // statements
  };
  ```

CheckBox and Radio Button Menu Items

- JCheckBoxMenuItem and JRadioButtonMenuItem are used to display checkbox and radio button
- Example:
  ```
  JCheckBoxMenuItem readonlyItem = new JCheckBoxMenuItem("Read-only");
  optionsMenu.add(readonlyItem);
  ```

```
ButtonGroup group = new ButtonGroup();

JRadioButtonMenuItem insertItem = new JRadioButtonMenuItem("Insert");

insertItem.setSelected(true);
JRadioButtonMenuItem overtypeItem = new JRadioButtonMenuItem("Overtype");

group.add(insertItem);
group.add(overtypeItem);
optionsMenu.add(insertItem);
optionsMenu.add(overtypeItem);
```

- We can use isSelected method to check the current state and setSelected method to set the current state to true or false

Pop-Up Menus

- Pop -up menu floats outside of the menu and has no title

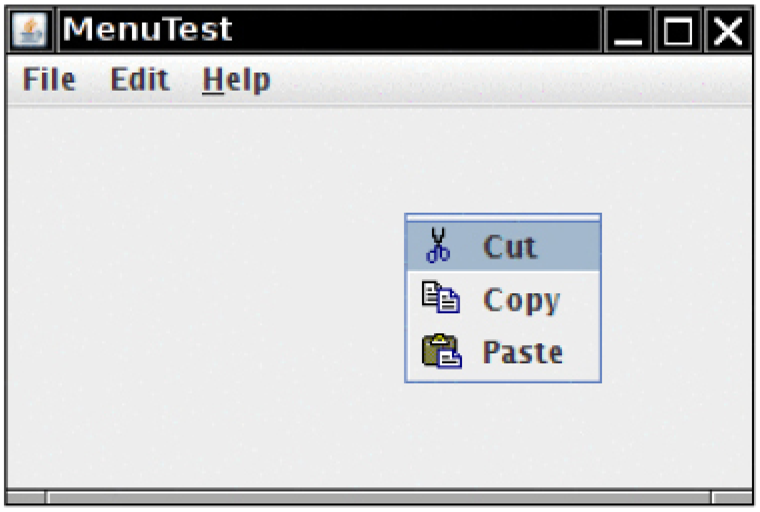


Fig: A pop-up menu

- Creating a pop-up menu

```
JPopupMenu popup = new JPopupMenu();
```

- Add menu items as before:

```
JMenuItem item = new JMenuItem("Cut");
item.addActionListener(listener);
popup.add(item);
```

- We must specify parent component and location of pop-us using show method
  popup.show(panel, x, y);

Keyboard Mnemonics and Accelerators

- Used to select menu items by keyboard mnemonics or keyboard shortcut keys
- Simple example:
  JMenuItem aboutItem = new JMenuItem("About", 'A');
- Letter A is automatically underlined in menu and when the user enters letter 'A', the menu is selected

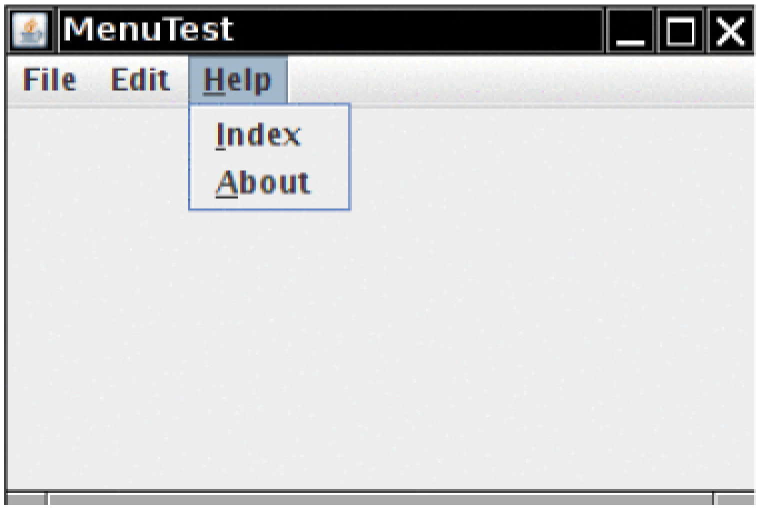


Fig: Keyboard Mnemonics

- We can also use setMnemonic() method for example:
  aboutItem.setMnemonic('H');
- We can also use setDisplayedMnemonicIndex(int index) to show selected mnemonic at particular index
  void setDisplayedMnemonicIndex(int index)
- Accelerators are keyboard shortcuts that let us select menu items without ever opening a menu
- For example to set CTRL + O to open file we can use accelerator using setAccelerator() method
- setAccelerator() method takes an object of type KeyStroke.
- Example for CTRL + O to open menu item :
  openMenuItem.setAccelerator(KeyStroke.getKeyStroke("ctrl   o"));
- We can use accelerator for menuitems but not for menu

Enabling and Disabling Menu Items

- We can use setEnabled() with Boolean argument in order to enable or disable menu items
  saveItem.setEnabled(false);
- It is used in situation like read-only document where save feature must be disabled

Toolbars

- A toolbar is a button that gives quick access to the most commonly used commands in the program

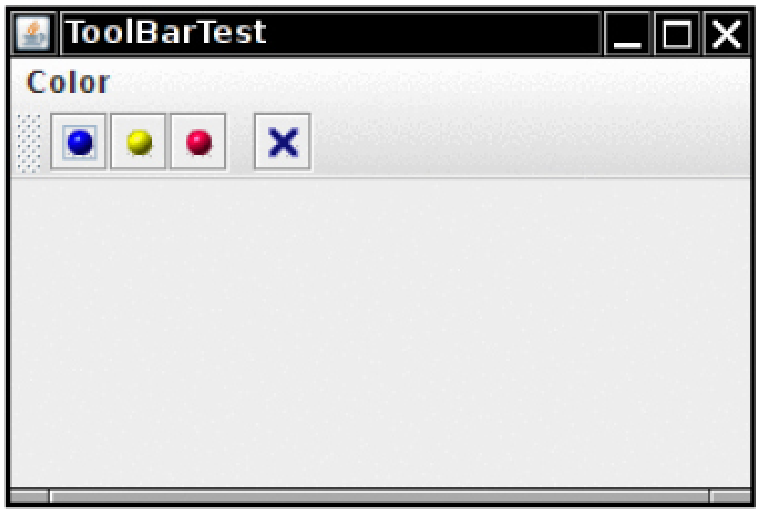


Fig: A toolbar

- We can move the toolbars, drag them to any position into a new location and place it there
- The toolbar dragging is supported by BorderLayout supporting NORTH,EAST,WEST,SOUTH constraints
- Creating a toolbar:
  JToolbar toolbar = new JToolBar();
  toolbar.add(redButton); // adding components to a toolbar
- Action object can also be added to a toolbar like
  toolbar.add(blueAction)
- We can also add separator like
  toolbar.addSeparator();

- To add title to the toolbar we can pass String argument to the JToolBar() constructor
  toolbar = new JToolBar(titleString);
- By default, toolbars are horizontally aligned. We can use the following to set it aligned vertically
  toolbar = new JToolBar(SwingConstants.VERTICAL)
  or
  toolbar = new JToolBar(titleString, SwingConstants.VERTICAL)
- Buttons are most common components inside toolbars. But we can also add combobox to a toolbar and other components too

Tooltips

- In Swing, you can add tooltips to any JComponent simply by calling the setToolTipText method:
  exitButton.setToolTipText("Exit");
- Alternatively, if you use Action objects, you associate the tooltip with the SHORT_DESCRIPTION key:
  exitAction.putValue(Action.SHORT_DESCRIPTION, "Exit");

Dialog Boxes

- Popup boxes that are used to give information to or get information from the user
- There are modal and modeless dialog boxes
- Modeless example include Toolbar where user can interact with both the application window and toolbar keeping the toolbar as it is
- But in case of modal dialog boxes, user must first deal with it in order to interact with remaining windows
- **JOptionPane** class in Swing can create simple dialog boxes without writing any complex code

Option Dialogs

- JOptionPane has four static methods to show simple dialogs
  I. showMessageDialog – show a message and wait for user to click OK
  II. showConfirmDialog – show a message and get a confirmation (OK/Cancel)

III. showOptionDialog – show a message and get a user option
IV. showInputDialog – show a message and get single line of user input

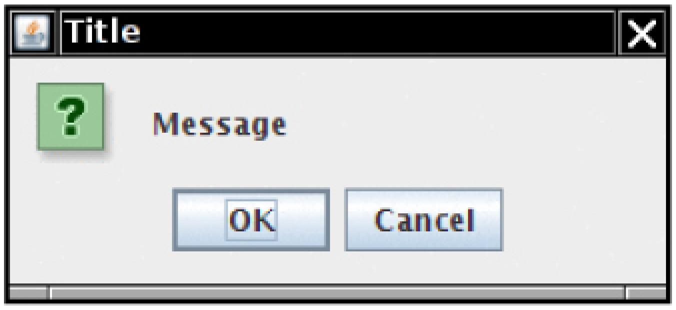


Figure: An option dialog (icon,message,one or more options)

- Input dialog has additional component for user input. It can be text field or combo box
- The icon depends on one of these five message types:
  I. ERROR_MESSAGE
  II. INFORMATION_MESSAGE
  III. WARNING_MESSAGE
  IV. QUESTION_MESSAGE
  V. PLAIN_MESSAGE
- For each dialog we can specify a message as string or icon or a component
- showConfirmDialog can have the following four options:
  I. DEFAULT_OPTION
  II. YES_NO_OPTION
  III. YES_NO_CANCEL_OPTION
  IV. OK_CANCEL_OPTION
- Return type of these functions are given below:
  I. showMessageDialog – None
  II. showConfirmDialog – An integer representing a chosen option
  III. showOptionDialog – An integer representing a chosen option
  IV. showInputDialog – The string user supplied
- showConfirmDialog and showOptionDialog returns integer values representing one of the following:
  I. OK_OPTION
  II. CANCEL_OPTION
  III. YES_OPTION
  IV. NO_OPTION
  V. CLOSED_OPTION
- Example:

```
int selection = JOptionPane.showConfirmDialog(parent,
                  "Message", "Title",
                        JOptionPane.OK_CANCEL_OPTION,
                        JOptionPane.QUESTION_MESSAGE);

    if (selection == JOptionPane.OK_OPTION) . . .
```

- showConfirmDialog(Component parent, Object message, String title, int optionType, int messageType, Icon icon)
- showConfirmDialog(Component parent, Object message, String title, int optionType, int messageType)
- showConfirmDialog(Component parent, Object message, String title, int optionType)
- showConfirmDialog(Component parent, Object message, String title)

- showInputDialog(Component parent, Object message, String title, int messageType, Icon icon, Object[] values, Object default)
- showInputDialog(Component parent, Object message, String title, int messageType)
- showInputDialog(Component parent, Object message)

- showMessageDialog(Component parent, message, String title, int messageType, Icon icon)
- showMessageDialog(Component parent, message, String title, int messageType)
- showMessageDialog(Component parent, message)

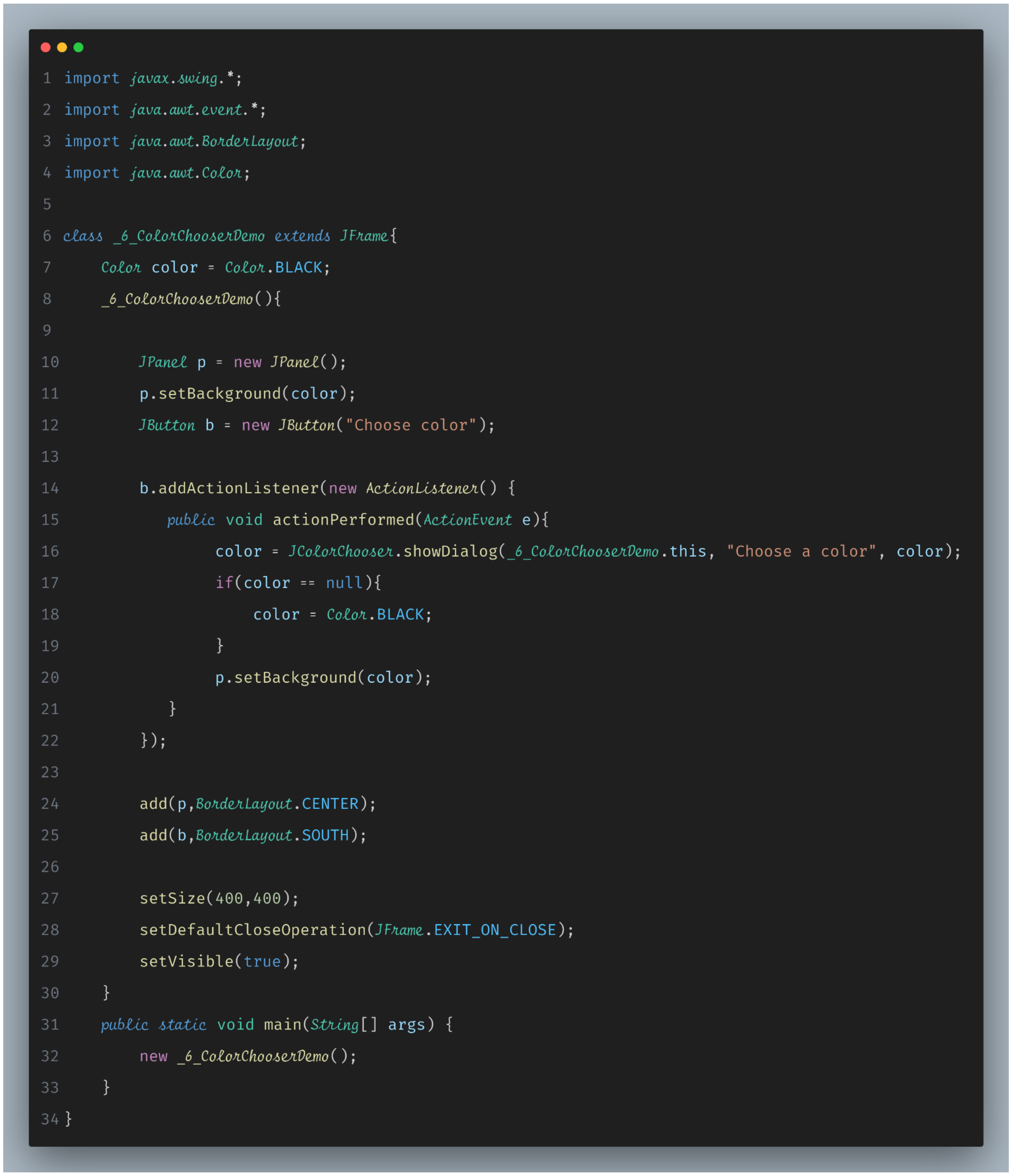
Creating Dialogs

- Creating dialogs manually without the use of JOptionPane class
- To implement dialog box, we extend JDialog class
- Steps:
  - I. In the constructor of your dialog box, call the constructor of the superclass JDialog.
  - II. Add the user interface components of the dialog box.
  - III. Add the event handlers.
  - IV. Set the size for the dialog box.

Color Choosers

- Swing provides the JColorChooser GUI component that enables users to select colors

- The below code demonstrates the JColorChooser dialog. When we click Change Color button, a JColorChooser dialog appears. When we select a color and press the dialog’s OK button, the background color of the application window changes using the showDialog() method:



```java
import javax.swing.*;
import java.awt.event.*;
import java.awt.BorderLayout;
import java.awt.Color;

class _6_ColorChooserDemo extends JFrame{
    Color color = Color.BLACK;
    _6_ColorChooserDemo(){

        JPanel p = new JPanel();
        p.setBackground(color);
        JButton b = new JButton("Choose color");

        b.addActionListener(new ActionListener() {
           public void actionPerformed(ActionEvent e){
                color = JColorChooser.showDialog(_6_ColorChooserDemo.this, "Choose a color", color);
                if(color == null){
                    color = Color.BLACK;
                }
                p.setBackground(color);
           }
        });

        add(p,BorderLayout.CENTER);
        add(b,BorderLayout.SOUTH);

        setSize(400,400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public static void main(String[] args) {
        new _6_ColorChooserDemo();
    }
}
```

File Choosers

- Swing provides a JFileChooser class that allows us to display a file dialog box using showOpenDialog() method to display a dialog box for opening a file
- showSaveDialog() method is used to display a dialog box for saving a file
- The button for accepting a file is then automatically labled open or save
- File Chooser steps:
  - Make a JFileChooser object eg.

    JFileChooser chooser = new JFileChooser();

  - Set the directory by calling the setCurrentDirectory() method eg:

    chooser.setCurrentDirectory(new File("file_name");

  - Show the dialog box by calling:

    int result = chooser.showOpenDialog(parent);

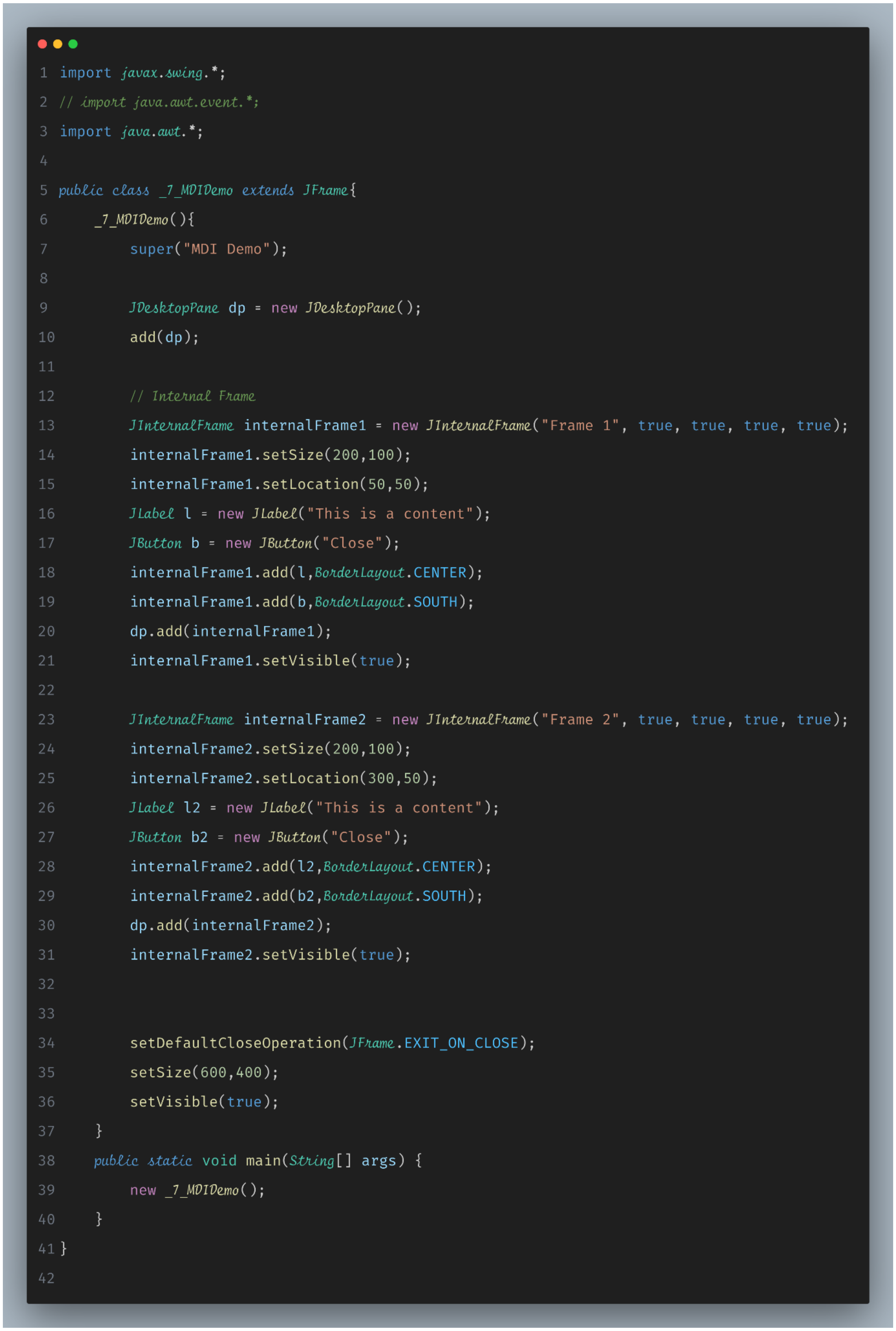
    int result = chooser.showSaveDialog(parent);

  - We can also call the showDialog() method and pass an explicit text to approve button:

    int reult = chooser.showDialog(parent,"select");

Desktop panes and Internal Frames

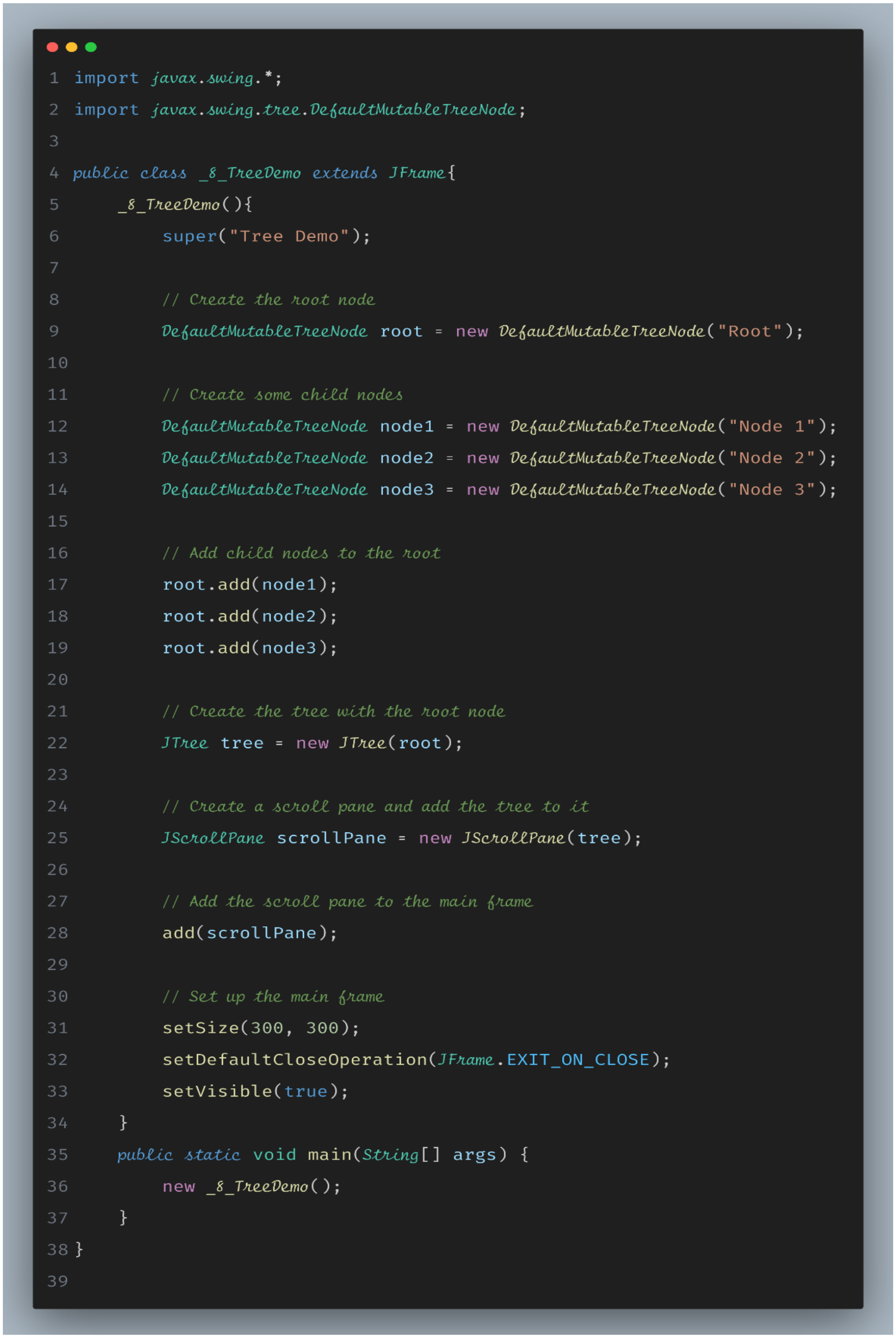
- JDesktopPane and JInternalFrame are classes in the Java Swing library that provide a framework for creating multiple document interface (MDI) applications
- MDI applications allow you to create a main window that contains multiple sub-windows, each representing a separate document or view
- JDesktopPane is a container class that acts as a desktop in an MDI application.
- It is used to hold and manage instances of JInternalFrame
- JInternalFrame represents an internal frame that can be contained within a JDesktopPane.
- It is essentially a lightweight, independent, and resizable window
- Here's a simple example of using JDesktopPane and JInternalFrame:



```java
import javax.swing.*;
// import java.awt.event.*;
import java.awt.*;

public class _7_MDIDemo extends JFrame{
    _7_MDIDemo(){
        super("MDI Demo");

        JDesktopPane dp = new JDesktopPane();
        add(dp);

        // Internal Frame
        JInternalFrame internalFrame1 = new JInternalFrame("Frame 1", true, true, true, true);
        internalFrame1.setSize(200,100);
        internalFrame1.setLocation(50,50);
        JLabel l = new JLabel("This is a content");
        JButton b = new JButton("Close");
        internalFrame1.add(l,BorderLayout.CENTER);
        internalFrame1.add(b,BorderLayout.SOUTH);
        dp.add(internalFrame1);
        internalFrame1.setVisible(true);

        JInternalFrame internalFrame2 = new JInternalFrame("Frame 2", true, true, true, true);
        internalFrame2.setSize(200,100);
        internalFrame2.setLocation(300,50);
        JLabel l2 = new JLabel("This is a content");
        JButton b2 = new JButton("Close");
        internalFrame2.add(l2,BorderLayout.CENTER);
        internalFrame2.add(b2,BorderLayout.SOUTH);
        dp.add(internalFrame2);
        internalFrame2.setVisible(true);


        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(600,400);
        setVisible(true);
    }
    public static void main(String[] args) {
        new _7_MDIDemo();
    }
}

```

Trees

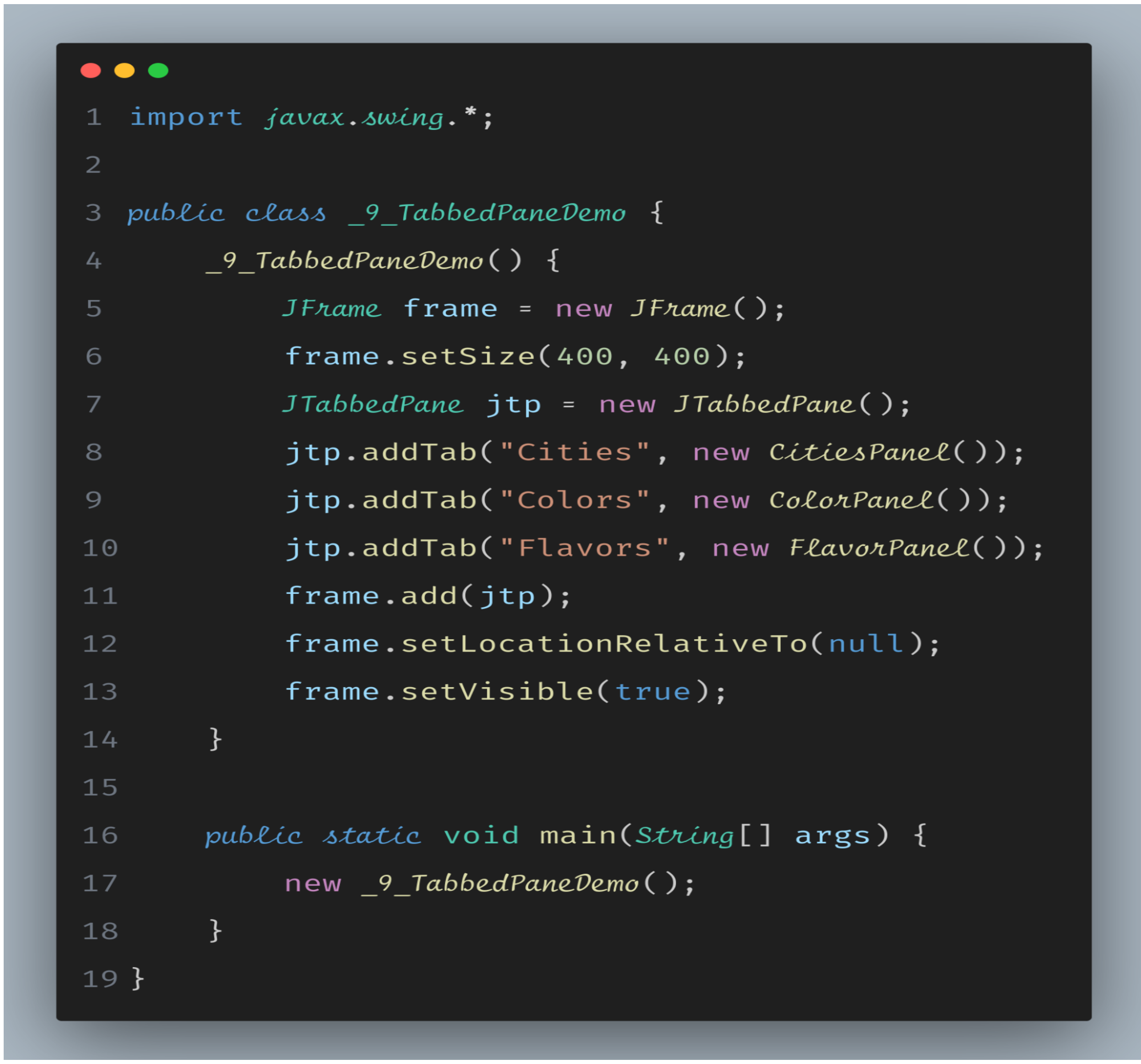
- In Swing, a JTree is a component that displays a tree-like structure, where each node may have children nodes
- It's a versatile component often used to represent hierarchical data
    - We create a DefaultMutableTreeNode for the root and some child nodes.
    - We add the child nodes to the root.
    - We create a JTree with the root node.
    - We create a JScrollPane to allow scrolling if the tree becomes too large.
    - We add the JTree to the JScrollPane and the JScrollPane to the main frame.
- Here's a simple example of using a JTree in a Swing application:



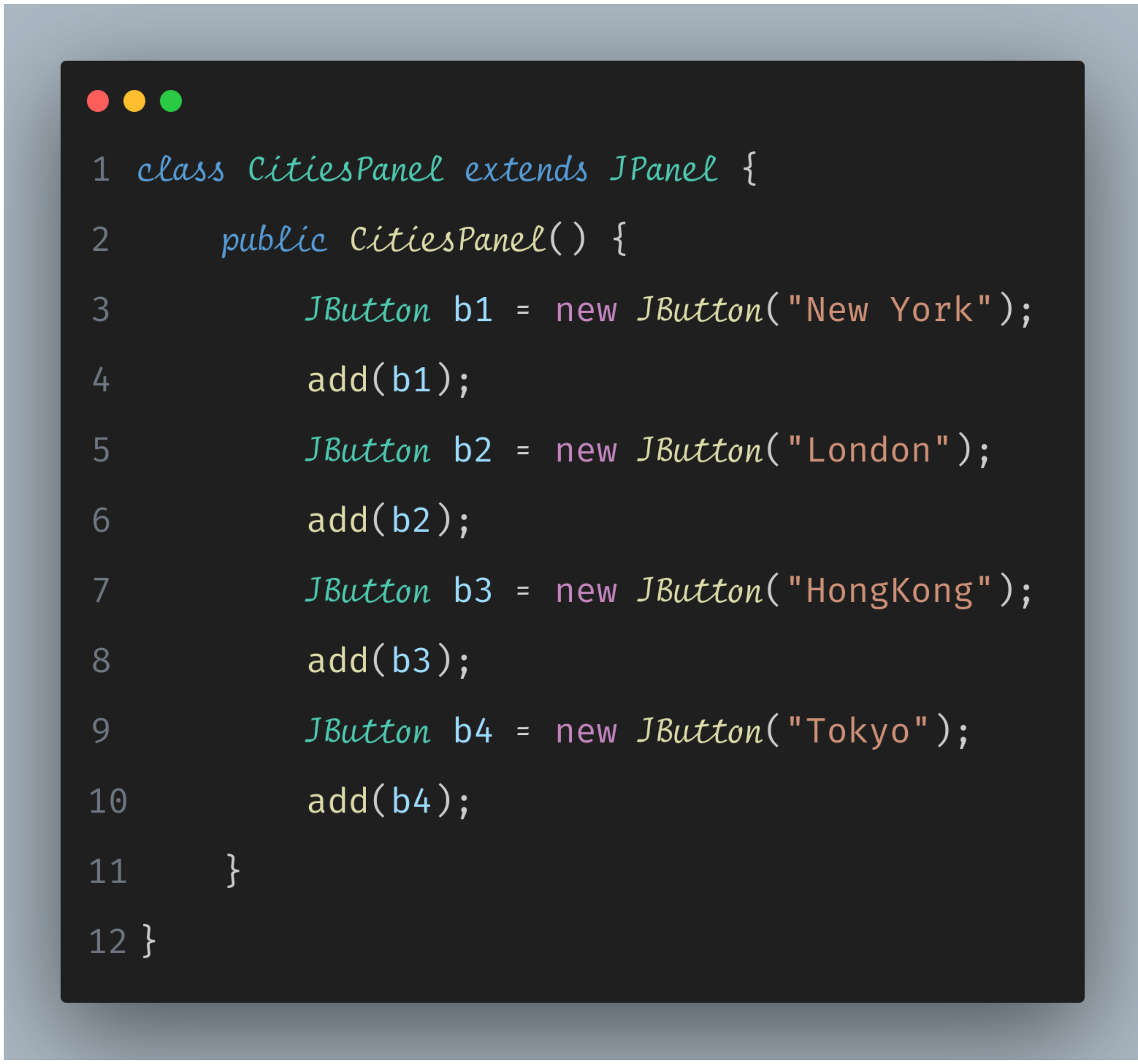
```java
import javax.swing.*;
import javax.swing.tree.DefaultMutableTreeNode;

public class _8_TreeDemo extends JFrame{
    _8_TreeDemo(){
        super("Tree Demo");

        // Create the root node
        DefaultMutableTreeNode root = new DefaultMutableTreeNode("Root");

        // Create some child nodes
        DefaultMutableTreeNode node1 = new DefaultMutableTreeNode("Node 1");
        DefaultMutableTreeNode node2 = new DefaultMutableTreeNode("Node 2");
        DefaultMutableTreeNode node3 = new DefaultMutableTreeNode("Node 3");

        // Add child nodes to the root
        root.add(node1);
        root.add(node2);
        root.add(node3);

        // Create the tree with the root node
        JTree tree = new JTree(root);

        // Create a scroll pane and add the tree to it
        JScrollPane scrollPane = new JScrollPane(tree);

        // Add the scroll pane to the main frame
        add(scrollPane);

        // Set up the main frame
        setSize(300, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public static void main(String[] args) {
        new _8_TreeDemo();
    }
}

```

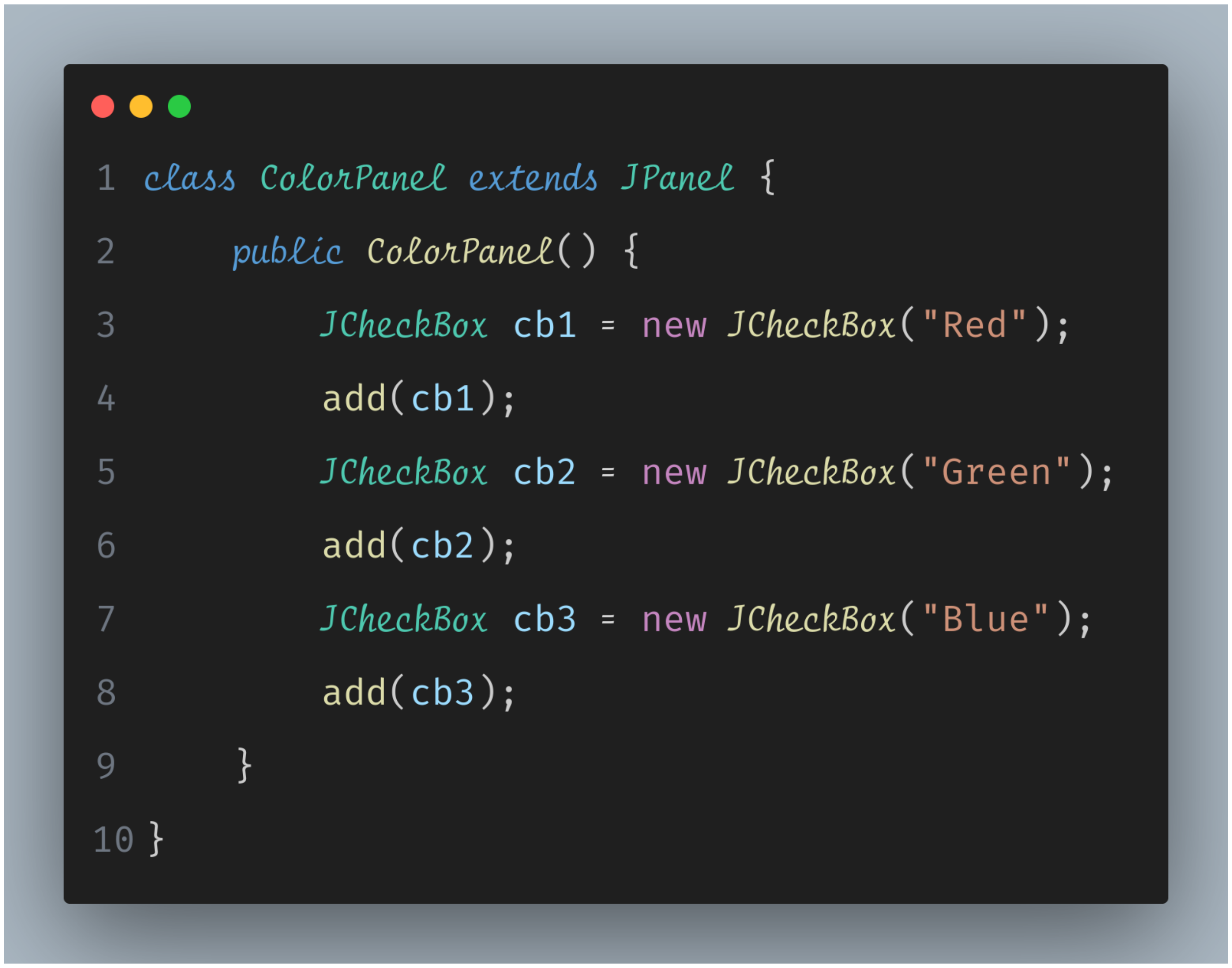
JTabbedPane

- JTabbedPane is a Swing component in Java that allows you to create a tabbed pane interface. It enables you to organize the content of your GUI into tabs, where each tab represents a different view or component.
- Users can switch between tabs to access different sets of information or functionality.
- Steps:
  - Create an instance of JTabbedPane.
  - Add each tab by calling addTab( ).
  - Add the tabbed pane to the content pane.
- Here is an example of JTabbed Pane:

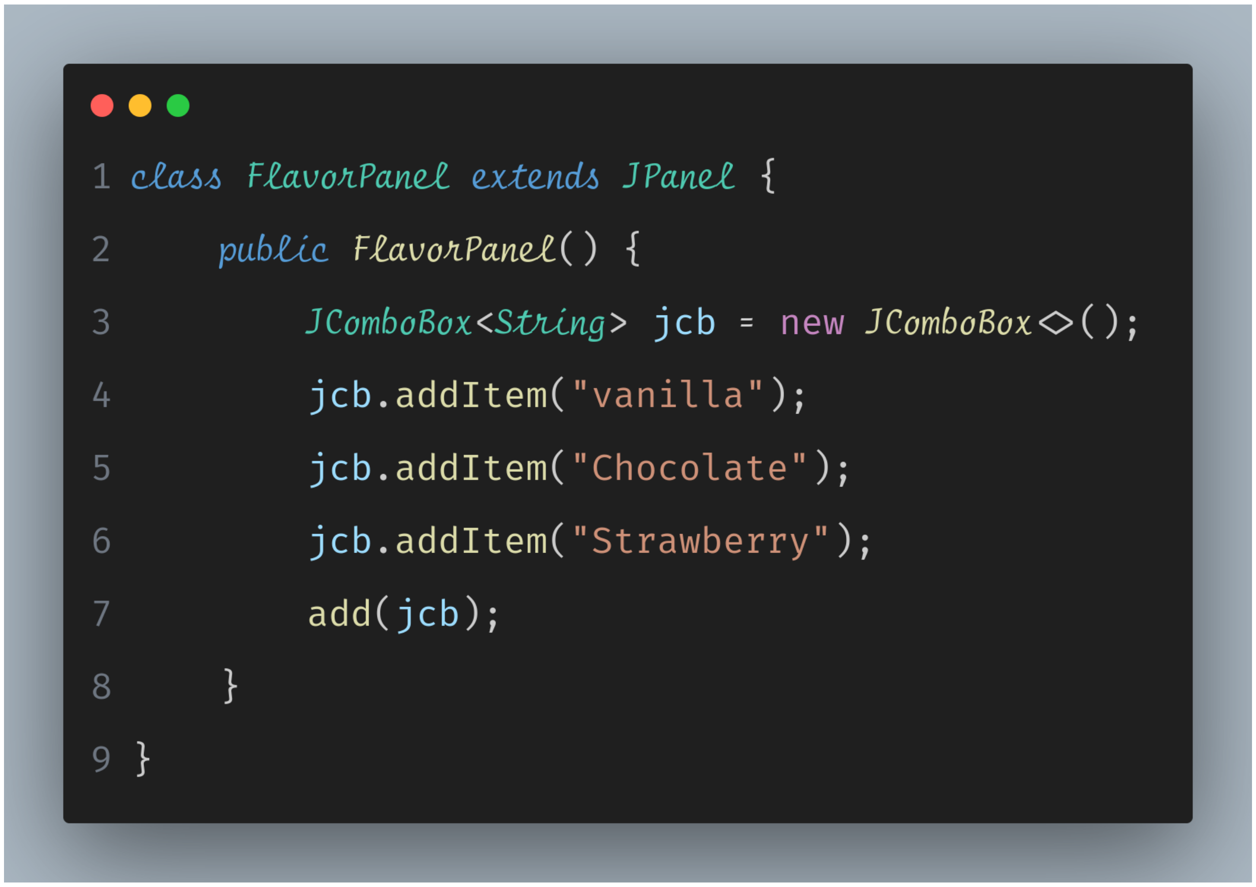


```java
import javax.swing.*;

public class _9_TabbedPaneDemo {
    _9_TabbedPaneDemo() {
        JFrame frame = new JFrame();
        frame.setSize(400, 400);
        JTabbedPane jtp = new JTabbedPane();
        jtp.addTab("Cities", new CitiesPanel());
        jtp.addTab("Colors", new ColorPanel());
        jtp.addTab("Flavors", new FlavorPanel());
        frame.add(jtp);
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        new _9_TabbedPaneDemo();
    }
}
```


```java
class CitiesPanel extends JPanel {
    public CitiesPanel() {
        JButton b1 = new JButton("New York");
        add(b1);
        JButton b2 = new JButton("London");
        add(b2);
        JButton b3 = new JButton("HongKong");
        add(b3);
        JButton b4 = new JButton("Tokyo");
        add(b4);
    }
}
```



```java
class ColorPanel extends JPanel {
    public ColorPanel() {
        JCheckBox cb1 = new JCheckBox("Red");
        add(cb1);
        JCheckBox cb2 = new JCheckBox("Green");
        add(cb2);
        JCheckBox cb3 = new JCheckBox("Blue");
        add(cb3);
    }
}
```



```
class FlavorPanel extends JPanel {
    public FlavorPanel() {
        JComboBox<String> jcb = new JComboBox<>();
        jcb.addItem("vanilla");
        jcb.addItem("Chocolate");
        jcb.addItem("Strawberry");
        add(jcb);
    }
}
```