**Exploring swing controls-**

**JLabel:**

JLabel is a class of java Swing. JLabel is used to display a short string or an image icon. JLabel can display text, image or both. JLabel is only a display of text or image and it cannot get focus. JLabel is inactive to input events such a mouse focus or keyboard focus.

**Class Constructors:**

1. **JLabel() :** creates a blank label with no text or image in it.
2. **JLabel(String s) :** creates a new label with the string specified.
3. **JLabel(Icon i) :** creates a new label with a image on it.

**Useful Methods: -**

1. **getIcon() :** returns the image that  the label displays
2. **setIcon(Icon i) :**Sets the icon that the label will display to image “i”
3. **getText() :** returns the text that the label will display
4. **setText(String s) :** Sets the text that the label will display to string ‘s’

**JTextField:**

The class JTextField is a component that allows editing of a single line of text. JTextField inherits the JTextComponent class and uses the interface SwingConstants.

**Class Constructors:**

1. **JTextField()** : constructor that creates a new TextField
2. **JTextField(int columns)** : constructor that creates a new empty TextField with specified number of columns.
3. **JTextField(String text)** : constructor that creates a new empty text field initialized with the given string.
4. **JTextField(String text, int columns)** : constructor that creates a new empty textField with the given string and a specified number of columns .

**Useful Methods:**

1. **setColumns(int n)** :set the number of columns of the text field.
2. **setFont(Font f)** : set the font of text displayed in text field.
3. **addActionListener(ActionListener l)** : set an ActionListener to the text field.
4. **int getColumns()** :get the number of columns in the textfield.

**JButton:**

The JButton class is used to create a labeled button that has platform independent implementation.

**Class Constructors:**

1. JButton() : It creates a button with no text and icon.
2. JButton(String s) : It creates a button with the specified text.
3. JButton(Icon i) : It creates a button with the specified icon object.

**Useful Methods:**

1. void setText(String s) : It is used to set specified text on button.
2. String getText() : It is used to return the text of the button.
3. void setEnabled(boolean b) : It is used to enable or disable the button.
4. void setIcon(Icon b) : It is used to set the specified Icon on the button.
5. void addActionListener(ActionListener a) : It is used to add the action listener to this object.

**JToggleButton:**

JToggleButton is used to create toggle button, it is two-states button to switch on or off.

**Class Constructors:**

1. **JToggleButton():** Creates an initially unselected toggle button without setting the text or image.
2. **JToggleButton(Action a):** Creates a toggle button where properties are taken from the Action supplied.
3. **JToggleButton(String text):** Creates an unselected toggle button with the specified text.
4. **JToggleButton(String text, boolean selected):** Creates a toggle button with the specified text and selection state.
5. **JToggleButton(String text, Icon icon):** Creates a toggle button that has the specified text and image, and that is initially unselected.

**Useful Methods:**

1. getAccessibleContext() : Gets the AccessibleContext associated with this JToggleButton.
2. getUIClassID() : Returns a string that specifies the name of the l&f class that renders this component.
3. paramString() : Returns a string representation of this JToggleButton.

**JCheckBox:**

JCheckBox can be selected or deselected. It displays it state to the user. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on ".

**Class Constructors:**

1. **JCheckBox()**: creates a new checkbox with no text or icon
2. **JCheckBox(String t)**:creates a new checkbox with the string specified
3. **JCheckBox(String text, boolean selected)** :creates a new checkbox with the string specified and the boolean value specifies whether it is selected or not.
4. **JCheckBox(String text, Icon icon)**:creates a new checkbox with the string and the icon specified.

**Methods to add Item Listener to checkbox:**

1. **addActionListener(ItemListener l)**: adds item listener to the component
2. **itemStateChanged(ItemEvent e)** : abstract function invoked when the state of the item to which listener is applied changes
3. **getItem()** : Returns the component-specific object associated with the item whose state changed
4. **getStateChange()** : Returns the new state of the item. The ItemEvent class defines two states: SELECTED and DESELECTED.
5. **getSource()** : Returns the component that fired the item event.

**Useful Methods:**

1. **setText(String s)** :sets the text of the checkbox to the given text
2. **setSelected(boolean b)**: sets the checkbox to selected if boolean value passed is true or vice versa
3. **getText()** : returns the text of the checkbox
4. **paramString()** : returns a string representation of this JCheckBox.

**JRadioButton:**

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz. It should be added in ButtonGroup to select one radio button only. We use “ButtonGroup” class to create a ButtonGroup and add radio button in a group.

**Class Constructors:**

1. JRadioButton() : Creates an unselected radio button with no text.
2. JRadioButton(String s) : Creates an unselected radio button with specified text.
3. JRadioButton(String s, boolean selected) : Creates a radio button with the specified text and selected status.

**Useful Methods:**

1. void setText(String s) : It is used to set specified text on button.
2. String getText() : It is used to return the text of the button.
3. void setEnabled(boolean b) : It is used to enable or disable the button.
4. void addActionListener(ActionListener a) : It is used to add the action listener to this object.

**JTabbedPane:**

It allows you to create a tabbed pane interface. It contains a collection of tabs. When you click on a tab, only data related to that tab will be displayed.

**Class Constructors:**

1. **JTabbedPane() :** Creates an empty TabbedPane with a default tab placement of JTabbedPane.Top.
2. **JTabbedPane(int tabPlacement) :** Creates an empty TabbedPane with a specified tab placement.

**Useful Methods:**

1. **addTab(String title, Component component) : C**reates a new tab with the given title and content.
2. **removeTabAt(int index) :** Removes the tab at the given index.
3. **getTabCount() :** Returns the number of tabs present in the JTabbedPane.

**JScrollPane:**

A JscrollPane is used to make scrollable view of a component. When screen size is limited, we use a scroll pane to display a large component or a component whose size can change dynamically.

**Class Constructors:**

1. JScrollPane() : It is a default constructor that creates an empty JScrollPane.
2. JScrollPane(Component comp) : This creates a JScrollPane with the specified view component as the scrollable content.
3. JScrollPane(LayoutManager layout) : This constructor creates a JScrollPane with the specified layout manager.
4. JScrollPane(int vertical, int horizontal) : This constructor creates an empty JScrollPane with the specified vertical and horizontal scrollbar.

**Useful Methods:**

1. void setVerticalScrollBarPolicy(int vertical) : Sets the vertical scrollbar policy
2. void setHorizontalScrollBarPolicy(int horizontal) : Sets the horizontal scrollbar policy
3. void setColumnHeaderView(Component comp) : sets the column header for the JScrollPane
4. void setRowHeaderView(Component comp) : sets the rowheader for the JScrollPane

**JList:**

The JList represents a list of text items. The list of text items can be set up so that the user can choose either one item or multiple items.

**Class Constructors:**

1. **JList()**: creates an empty blank list
2. JList(ary[] listData) : Creates a JList that displays the elements in the specified array.
3. JList(ListModel<ary> dataModel) : Creates a JList that displays elements from the specified, non-null, model.

**Useful Methods:**

1. **getSelectedIndex() :** Returns the index of selected item of the list.
2. **getSelectedValue() :** Returns the selected value of the element of the list.
3. **setSelectedIndex(int i) :** Sets the selected index of the list to ‘i’.
4. **setListData(ary[ ] listData) :** Changes the elements of the list to the elements of ‘listData’ .
5. **setVisibleRowCount(int v) :** Changes the visibleRowCount property.
6. **void addListSelectionListener(ListSelectionListener listener) :** It is used to add a listener to the list, to be notified each time a change to the selection occurs.
7. **ListModel getModel() :** It is used to return the data model that holds a list of items displayed by the JList component.

**JComboBox:**

JComboBox shows a popup menu that shows a list and the user can select a option from that specified list. JComboBox can be editable or read- only depending on the choice of the programmer.

**Class Constructor:**

1. **JComboBox()**: creates a new empty JComboBox .
2. **JComboBox(ComboBoxModel M)**: creates a new JComboBox with items from specified ComboBoxModel
3. **JComboBox(E [ ] i)** : creates a new JComboBox with items from specified array.
4. **JComboBox(Vector items)**: creates a new JComboBox with items from the specified vector.

**Useful Methods:**

1. **void addItem(Object anObject) :** It is used to add an item to the item list.
2. **removeItem(Object anObject)**: removes an item from the item list.
3. **void removeAllItems() :** It is used to remove all the items from the list.

**Swing Menus:**

JMenuBar, JMenu and JMenuItems are a part of Java Swing package. JMenuBar is an implementation of menu bar . the JMenuBar contains one or more JMenu objects, when the JMenu objects are selected they display a popup showing one or more JMenuItems . JMenu basically represents a menu.

**Class Constructors:**

1. **JMenuBar() :**Creates a new MenuBar.
2. **JMenu() :**Creates a new Menu with no text.
3. **JMenu(String name) :** Creates a new Menu with a specified name.

**Useful Methods:**

1. **add(JMenu c) :** Adds menu to the menu bar. Adds JMenu object to the Menu bar.
2. **add(Component c) :** Add component to the end of JMenu
3. **add(Component c, int index) :** Add component to the specified index of JMenu
4. **add(JMenuItem menuItem) :** Adds menu item to the end of the menu.
5. **add(String s) :**Creates a menu item with specified string and appends it to the end of menu.
6. **getItem(int index) :** Returns the specified menuitem at the given index

**Dialogs:**

A Dialog window is an independent sub window meant to carry temporary notice apart from the main Swing Application Window. The main purpose of the dialog is to add components to it.

**Class Constructors:**

1. **JDialog()**: creates an empty dialog without any title or any specified owner
2. **JDialog(Frame o)**:creates an empty dialog with a specified frame as its owner
3. **JDialog(Frame o, String s)** : creates an empty dialog with a specified frame as its owner   
   and a specified title
4. **JDialog(Window o)** : creates an empty dialog with a specified window as its owner
5. **JDialog(Window o, String t)** : creates an empty dialog with a specified window as its owner and specified title.
6. **JDialog(Dialog o)** :creates an empty dialog with a specified dialog as its owner
7. **JDialog(Dialog o, String s)** : creates an empty dialog with a specified dialog as its owner and specified title.

**Useful Methods:**

1. **setLayout(LayoutManager m)** : sets the layout of the dialog to specified layout manager
2. **setJMenuBar(JMenuBar m)**: sets the menubar of the dialog to specified menubar
3. **isVisible(boolean b)**: sets the visibility of the dialog, if value of the boolean is true then visible else invisible