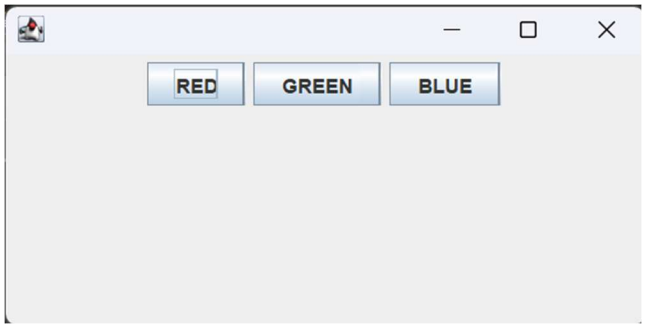
Example Manage more than one multibutten

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
        // 1. Class implements ActionListener

public class MultBut  implements ActionListener{

        JFrame frame=new JFrame();
        JButton btn1 = new JButton("RED");
        JButton btn2 = new JButton("GREEN");
        JButton btn3 = new JButton("BLUE");

        public MultBut (){ //Constructor
        frame.setSize(400,200);
        frame.setLayout(new FlowLayout());
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
        frame.add(btn1);
        frame.add(btn2);
        frame.add(btn3);
// 2. Register 'this' class as the listener
        btn1.addActionListener(this);
        btn2.addActionListener(this);
        btn3.addActionListener(this);    }
  // 3. Define what happens on click

        public void actionPerformed(ActionEvent e){
        // the ActionListener monitoring 3 buttons.
        // getSource() use for button is being pressed
        if (btn1== e.getSource()){
        frame.getContentPane().setBackground(Color.red);}

        if (btn2== e.getSource()) {
        frame.getContentPane().setBackground(Color.green);}

        if (btn3== e.getSource()) {
        frame.getContentPane().setBackground(Color.blue);}    }

        public static void main(String args[]){
                MultBut  mb= new MultBut ();  }}
```



TextField

TextField is a component that allows the **user to enter a line of unformatted text**, it does not allow multi-line to input it only allows the user to enter a single line of text. The text can then be used as per requirement.

Create a text field with some initial text:

```
JTextField textField = new JTextField("This is a text");
```

Create a text field with a specified number of columns:

```
JTextField textField = new JTextField(20);
```

Create a text field with both initial text and number of columns:

```
JTextField textField = new JTextField("This is a text", 20);
```

Create a default and empty text field then set the text and the number of columns later:

```
JTextField textField = new JTextField();
textField.setText("This is a text");
textField.setColumns(20);
```

Adding the text field to a container

A JTextField can be added to any container like JFrame or JPanel

```
frame.add(textField);
panel.add(textField);
frame.add(textField, BorderLayout.CENTER);
panel.add(textField, gridbagConstraints);
```

Getting or setting content of the text field

- Getting all content:

String content = textField.getText();

The **getText()** method in Java is primarily used to **retrieve the current text content** from various text-based components or elements, most notably in Java Swing GUI applications and Selenium web automation. The **method returns the text as a String**.

Hints

In Java Swing, you can use a **lambda expression** (e ->) with **addActionListener** as a concise way to implement the **ActionListener interface**, as it is a functional interface with a single method, **actionPerformed(ActionEvent e)**.

The general syntax for a single-statement lambda is:

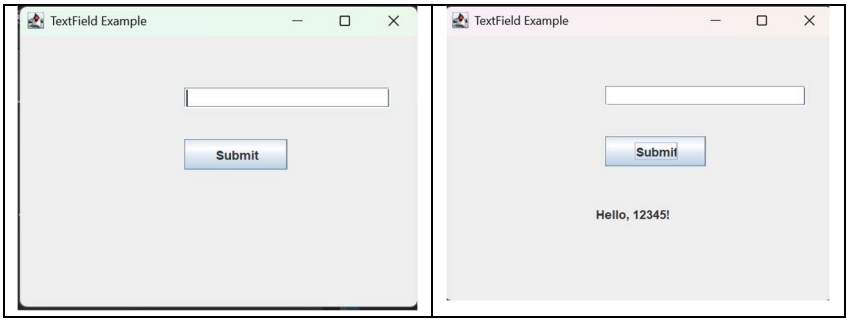
component.addActionListener(e -> { /* code to execute */ });

Example

```
import javax.swing.*;
  public class SwingEx  {
     public static void main(String[] args) {

         JFrame f = new JFrame("TextField Example");
         f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         JButton submitButton = new JButton("Submit");
         submitButton.setBounds(160, 100, 100, 30);

         JTextField userNameField = new JTextField(20);
         userNameField.setBounds(160, 50, 200, 20);

         JLabel greetingLabel = new JLabel();
         greetingLabel.setBounds(150, 170, 400, 14);
         // Add the components to the frame

         f.add(userNameField);
         f.add(greetingLabel);
         f.add(submitButton);
```

```
// public void addListener() {
submitButton.addActionListener(e -> {
// Get the text from the field
String userName = userNameField.getText();

// Set the text of the label
greetingLabel.setText("Hello, " + userName + "!");

// Optional: clear the text field after retrieval
userNameField.setText("");});

f.setSize(400, 300);
f.setLayout(null);
f.setVisible(true); } }
```

Output



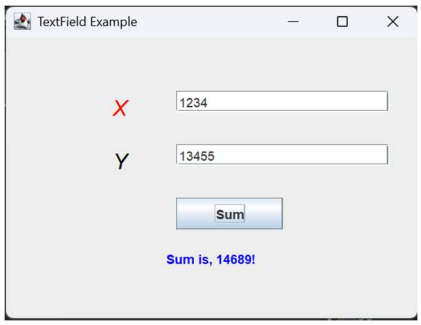
Example: Sum of two integer numbers

```
import javax.swing.*;

public class SwingEx  {
  public static void main(String[] args) {

        JFrame f = new JFrame("TextField Example");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JButton submitButton = new JButton("Sum");
        submitButton.setBounds(160, 150, 100, 30);

        JTextField userNameField = new JTextField(20);
        userNameField.setBounds(160, 50, 200, 20);
```

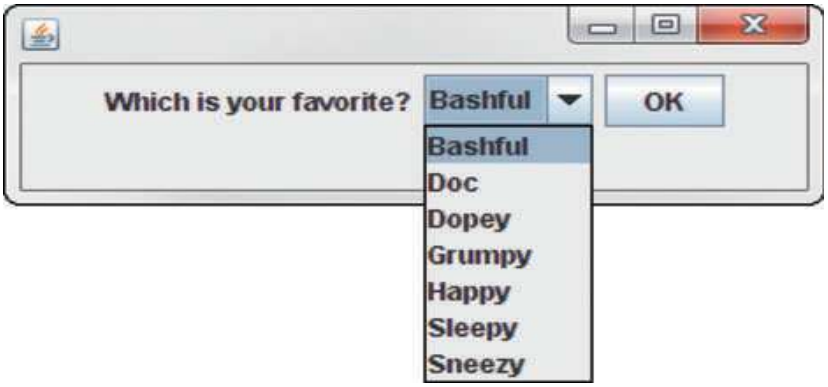
```
JTextField userNameField1 = new JTextField(20);
userNameField1.setBounds(160, 100, 200, 20);

JLabel greetingLabel = new JLabel();
greetingLabel.setBounds(150, 200, 400, 14);
greetingLabel.setForeground(Color.blue);//Font color

JLabel l1 = new JLabel("X");
l1.setBounds(50, 50, 100, 30);
l1.setForeground(Color.red);//Font color

JLabel l2 = new JLabel("Y");
l2.setBounds(50, 100, 100, 30);
l2.setForeground(Color.black);//Font color
Font fo= new Font("SansSerif", Font.ITALIC, 20);
l1.setFont(fo);
l2.setFont(fo);

 // Add the components to the frame
f.add(userNameField);
f.add(userNameField1);
f.add(greetingLabel);
f.add(submitButton);
f.add(l1);
f.add(l2);


// public void addListener() {
  submitButton.addActionListener(e -> {
  // Get the text from the field
        String userName = userNameField.getText();
        String userName1= userNameField1.getText();
        int number1 = Integer.parseInt(userName);
        int number2 = Integer.parseInt(userName1);
        int sum= number1 +number2;

         // Set the text of the label
         greetingLabel.setText("Sum is, " + sum + "!");
```

```
// Optional: clear the text field after retrieval
userNameField.setText("");
userNameField1.setText("");});
f.setSize(400, 300);
f.setLayout(null);
f.setVisible(true); } }
```

Output



combo box



In Java, a combo box is a graphical user interface (GUI) component that **combines a text field and a drop-down list**, implemented as the **JComboBox class** in Swing. It allows users to select an item from a predefined list or, if the component is editable, type in a value.

Creating a JComboBox (Swing)

You can create a JComboBox using various constructors, most commonly with an **array or Vector of items**, or as an empty box to which you add items dynamically.

The items can be any Java object; the **combo box will use their toString() method** to display the text.

Constructors of the JComboBox are:

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

You use the JComboBox class to create combo boxes. Creating a combo box is easy. You have three constructors to choose from, the first of which creates an empty combo box:

JComboBox combo1 = new JComboBox();
Then you can use the **addItem** to add items to the combo box:
1.

```
combo1.addItem("Bashful");
combo1.addItem("Doc");
combo1.addItem("Dopey");
combo1.addItem("Grumpy");
combo1.addItem("Happy");
combo1.addItem("Sleepy");
combo1.addItem("Sneezy");
```

Alternatively, you can create a combo box and initialize its contents from an array, as in this example:

2.

```
String[] theSeven = {"Bashful", "Doc", "Dopey", "Grumpy", "Happy", "Sleepy",
    "Sneezy"};

JComboBox combo1 = new JComboBox(theSeven);
```

Listeners

If your code is not sending **setSelectedItem()** messages to a combo box, then you can handle its responses with either an **action listener** or **an item listener**.

```
comboBox.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        code to implement responses to item selection.
    }
});
```

or

```
comboBox.addItemListener(new ItemListener() {
    public void itemStateChanged(ItemEvent e) {
        code to implement responses to item selection.
    }
});
```

A **JComboBox<String> petList**. To create and use a **JComboBox named petList** in Java Swing, you need to instantiate the class, optionally provide it with data (such as a String array), and add it to a container.

Empty Constructor: Creates a JComboBox (**new JComboBox<>**) with a default, empty data model. Items can be added later using the **addItem() method**.

The **frame.pack()** method in Java Swing **automatically sizes a JFrame so that all its components are at or above their preferred sizes**.

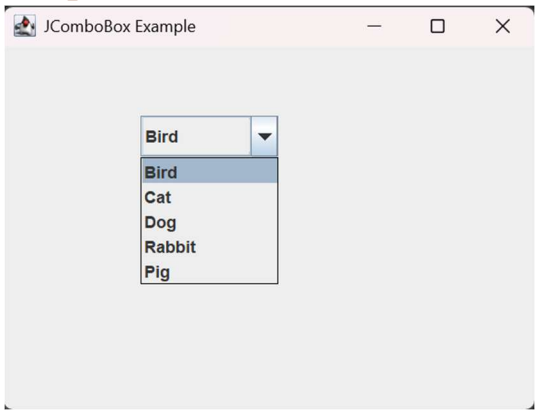
getSelectedItem(): Returns an Object, which is why a (String) cast is necessary if the items are Strings.

Example: Creating a JComboBox with an array of Strings:

```
import javax.swing.*;
import java.awt.event.*;
public class ComboBoxDemo {
public static void main(String[] args) {
```

```
JFrame frame = new JFrame("JComboBox Example");
frame.setSize(300, 200);
frame.setLayout(null); // Must use null layout
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

String[] petStrings = { "Bird", "Cat", "Dog", "Rabbit", "Pig" };

 JLabel greetingLabel = new JLabel();
 greetingLabel.setBounds(80, 100, 100, 50);
 greetingLabel.setForeground(Color.BLUE);//Font color

 // Create the combo box
JComboBox<String> petList = new JComboBox<>(petStrings);
petList.setBounds(100, 50, 100, 30);
petList.setSelectedIndex(0); // Select the first item by default
// Add an action listener to handle selections
petList.addActionListener(new ActionListener() {

  public void actionPerformed(ActionEvent e) {
     String selectedPet = (String) petList.getSelectedItem(); // Get selected item
      //System.out.println("Selected: " + selectedPet);
      greetingLabel.setText("Hello, " + selectedPet + "!");  } });
       frame.add(petList);
       frame.add(greetingLabel);
       //frame.pack();
       frame.setVisible(true);  }}
```

Output



setSelectedItem()

To use setSelectedItem(), you **pass** the **exact object** that is already present in the combo box's data model as an argument.

Syntax:

void setSelectedItem(Object anObject)

- This method sets the selected item in the combo box display area to the provided object.
- **Key Behavior**: The method relies on the **equals() method** of the objects in the combo box model to find a match.
    - **If a matching object is found**, the **display area updates** to show that object as selected.
    - **If no matching object** is on the list, the **selection will not change** (for uneditable combo boxes).

Example

Here is a complete example of creating a **JComboBox** and using **setSelectedItem()**:

```
import javax.swing.*;

public class ComboBoxExample {

   public static void main(String[] args) {

      JFrame frame = new JFrame("JComboBox setSelectedItem Example");

      frame.setSize(300,250);

      frame.setLayout(null); // Must use null layout


      // 1. Create an array of items (Strings in this case)

      String[] items = {"Apple", "Banana", "Cherry", "Date"};


      // 2. Create the JComboBox with the items

      JComboBox<String> comboBox = new JComboBox<>(items);


      // 3. Set a specific item as selected programmatically
```

```
// The argument must be an object present in the 'items' array.
comboBox.setSelectedItem("Banana");
 or
// Alternative: Use setSelectedIndex(int index) if you know the index
// comboBox.setSelectedIndex(2); // Selects "Cherry"

// Add components to the frame (standard Swing practice)
frame.add(comboBox);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
//frame.pack(); // Adjusts window size to fit components
frame.setVisible(true);    }}
```

CheckBox

Checkboxes are used as a GUI representation of a boolean value

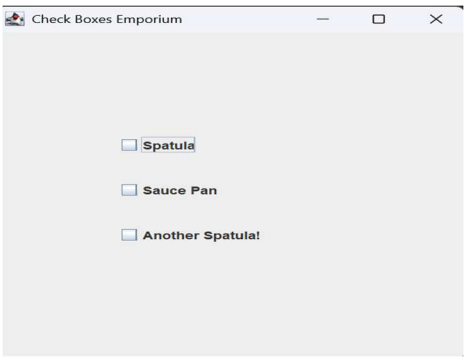
- They can either be checked or unchecked.
- Swing Provides a **JCheckBox** class
- A JCheckBox object has an isSelected() method that returns **true/false** depending on whether the box is checked or not.

Example

```
public class ChecBox {

    public static void main(String[] args) {
    JFrame f = new JFrame("Check Boxes Emporium");
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JCheckBox spatula = new JCheckBox("Spatula");
    spatula.setBounds(100,100,500,50);
    JCheckBox saucepan = new JCheckBox("Sauce Pan");
    saucepan.setBounds(100,150,500,50);
    JCheckBox spatula2 = new JCheckBox("Another Spatula!");
    spatula2.setBounds(100,200,500,50);
```

```
f.add(spatula);
f.add(saucepan);
f.add(spatula2);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);} }
```

Output

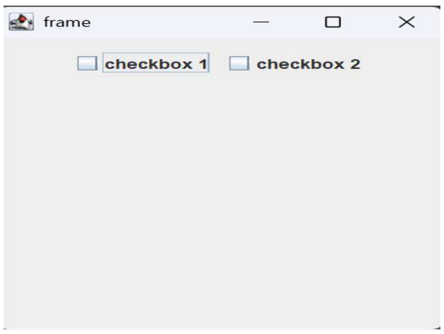


Example

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

class Checkbox1 {

  public static void main(String[] args) {

    // create a new frame
    JFrame f = new JFrame("frame");

    // set layout of frame
    f.setLayout(new FlowLayout());

    // create checkbox
    JCheckBox c1 = new JCheckBox("checkbox 1");
    JCheckBox c2 = new JCheckBox("checkbox 2");

    // create a new panel
    JPanel p = new JPanel();

    // add checkbox to panel
      p.add(c1);
      p.add(c2);
```

```
    // add panel to frame
      f.add(p);

      f.setSize(300, 300);
      f.setVisible(true);   }}
```

Output



Implement the checkbox

- The **itemStateChanged** method is part of the Java AWT **ItemListener interface** and is invoked automatically when an item's selection state changes in a component like a JComboBox, JCheckBox, or Choice list. The signature method is **void itemStateChanged(ItemEvent e)**.

- The **getStateChange() method** is typically called inside the **itemStateChanged(ItemEvent e) method** of a class that implements the ItemListener interface. This allows developers to handle the specific action (selection or deselection) that triggered the event.

- ItemEvent.SELECTED is a **static final integer constant** in Java's java.awt.event.ItemEvent class used to indicate that an item in an ItemSelectable component (like a Checkbox, Choice list, or JComboBox) has been **selected** opposite of ItemEvent.DESELECTED

Example

```
import javax.swing.*;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
public class ImplemChbox {
```

```
public static void main(String[] args) {
    JFrame frame = new JFrame("Checkbox Example");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(300, 300);
    frame.setVisible(true);
    JCheckBox myCheckBox = new JCheckBox("I agree to the terms");
```

Or

```
    JCheckBox myCheckBox = new JCheckBox("I agree to the terms",true);
    myCheckBox.setBounds(100,100,500,50);

    // implementation
    myCheckBox.addItemListener(new ItemListener() {
      public void itemStateChanged(ItemEvent e) {
        if (e.getStateChange() == ItemEvent.SELECTED) {
          System.out.println("Checkbox is selected.");}
        else {
          System.out.println("Checkbox is deselected.");}  }   });

    frame.add(myCheckBox);
    frame.setLayout(null); }}
```

Output

Checkbox is selected