



Chapter 3:

GUI Programming- A Review

Informatics Practices
Class XII (CBSE Board)

Revised as per
CBSE
Curriculum
2015

"Open Teaching-Learning Material"

Visit www.ip4you.blogspot.com for more....

Authored By:- Rajesh Kumar Mishra, PGT (Comp.Sc.)
Kendriya Vidyalaya Upper Camp, Dehradun (Uttarakhand)
e-mail : rkmalld@gmail.com

What is JAVA?



- ❑ JAVA is an Object Oriented programming language as well a platform.
 - ❑ By using JAVA, we can write Platform independent application programs, which can run on any type of OS and Hardware.
 - ❑ JAVA is designed to build Interactive, Dynamic and Secure applications on network computer system.
 - ❑ Java facilitates development of Multilingual applications because JAVA uses 2-Byte UNICODE character set, which supports almost all characters in almost all languages like English, Chinese, Arabic etc.
-

History of JAVA

JAVA was developed by **James Gosling** at **Sun Microsystems** under the **Green project** to write applications for electronic devices like TV-Set Top Box etc. The language was initially called **Oak** and later renamed with **Java**.



James Gosling

1991	James Gosling developed Oak to program consumer electronic devices
1995	Java Development Kit (JDK) 1.0 was released by the Sun Microsystems and JAVA used as a part of Netscape web browser to facilitate Internet Applications.
1998	Sun introduced "Open" source community and produces JDK 1.2 (Java 2) which was released as J2EE, J2SE, J2ME version.
2006	Sun declared Java as Free & Open Source Software (FOSS) under GNU-GPL and NetBeans IDE was released.
2010	Sun Microsystems was owned by Oracle Corporation and now Java Project is being governed by the Oracle.

Characteristics of JAVA

❑ Object Oriented Language

Java is Object Oriented Language (a real-world programming style)

❑ Open Source Product

It is Open Source i.e. freely available to all with no cost.

❑ Write Once Run Anywhere (WORA)

JAVA Program can be run on any type of H/W and OS platforms i.e. Java programs are platform independent.

❑ Light Weight Code

Big applications can be developed with small code.

❑ Security

JAVA Programs are safe and secure on Network.

❑ Interpreter & Compiler based Language

JAVA uses both Compiler and Interpreter (JVM) to produce portable and platform-independent object code.

❑ Built-in Graphics & Supports Multimedia

JAVA is equipped with Graphics feature. It is best for integration of Audio, Video and graphics & animation.

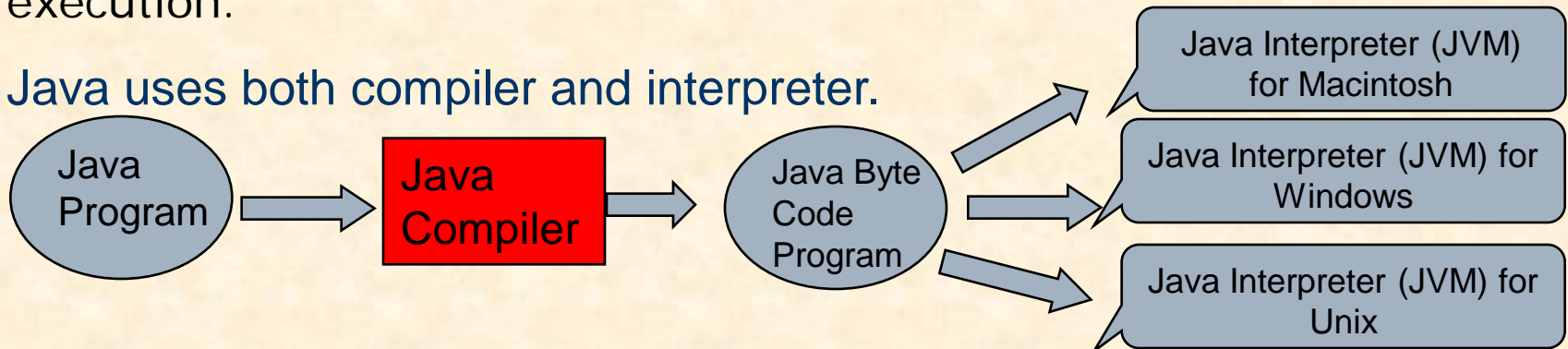
Why JAVA is Platform Independent?

A program written in HLL must be converted into its equivalent Machine code, so that computer can understand and execute. This conversion is known as Compilation. Generally, the converted machine code depends on the structure of H/w and OS platform. So that a Windows program will not work on UNIX, or LINUX or Mac platform etc. Since they are Platform dependent.

A program written in JAVA is platform-independent i.e. they are not affected with changing of OS. This magic is done by using Byte code. Byte code is independent of the computer system it has to run upon.

Java compiler produces **Byte code** instead of native executable code which is interpreted by Java Virtual Machine (**JVM**) at the time of execution.

Java uses both compiler and interpreter.



Basics of GUI Applications

□ How GUI application works ?

Graphical User Interface (GUI) based application contains Windows, Buttons, Text boxes, Dialogue boxes and Menus etc. known as GUI components. While using a GUI application, when user performs an action, an **Event** is generated which causes a **Message** sent to application to take action.

□ What is Event ?

An Events refers to the occurrence of an activity by user like click, double click etc.

□ What is Message ?

A Message is the information/request sent to the application when event occurs.

GUI in JAVA

- ❑ In JAVA, the GUI programming is done through **Swing API** (Application Programming Interface) which enables look-and feel (L&F) environment.
- ❑ Swing Controls are available with JAVA-SE as a part of Java Foundation Classes (JFC).
- ❑ A GUI application in JAVA contains **three** basic elements.-
 - 1. Graphical Component (Event Source):**

It is an object that defines a screen element such as Button, Text field, Menus etc. They are source of the Events. In GUI terminology, they are also known as **Widget** (Window Gadget). It can be **container** control or **child** control.
 - 2. Event:**

An Event (occurrence of an activity) is generated when user does something like mouse click, dragging, pressing a key on the keyboard etc.
 - 3. Event Handler Method:**

It contains method/functions which is attached to a component and executed in response to an event. In Java, Listener Interface stores all Event-response-methods or Event-Handler methods.

Basic Graphical Controls of JAVA(Swing controls)

The Java offers various Swing controls to facilitate development of GUI applications. Most commonly used controls are-

- **JFrame**: Used as a Basic Window or form.
- **JLabel**: Allows Non-editable text or icon to be displayed.
- **TextField**: Allows user input. It is editable through text box.
- **Button**: An action is generated when pushed.
- **CheckBox**: Allow user to select multiple choices.
- **RadioButton**: They are option button which can be turned on or off. These are suitable for single selection.
- **JList**: Gives a list of items from which user can select one or more items.
- **ComboBox**: It gives dropdown list of items with facility to add new items. It is combination of JList and TextField.
- **JPanel**: It is container controls which contains other controls using a frame.



A control which holds other component (child) controls, is called **Container Control** e.g. JFrame, JPanel, JDialog etc.

Using Swing Controls

The image shows a Java Swing window titled "Data Entry Form" with the following components and their corresponding Swing classes:

- jTextField**: Points to the "Name" text input field.
- jLabel**: Points to the window title "Data Entry Form".
- jRadioButton**: Points to the "Male" and "Female" radio buttons under the "Gender" label.
- jPanel**: Points to the "Properties do you have" section, which contains a group of checkboxes.
- jCheckBox**: Points to the "House", "Car", "Plot", and "Cash" checkboxes.
- jComboBox**: Points to the "City" dropdown menu, which currently shows "New Delhi".
- jList**: Points to the "Country visited" list, which contains "Austra", "Americ", "Russia", and "France".
- jTextArea**: Points to the "Remark" text area.
- jButton**: Points to the "Submit" and "Cancel" buttons.

Events Handling in JAVA GUI Application

An event is occurrence of some activities either initiated by user or by the system. In order to react, you need to implement some Event handling system in your Application. Three things are important in Even Handling-

- ❑ **Event Source:**

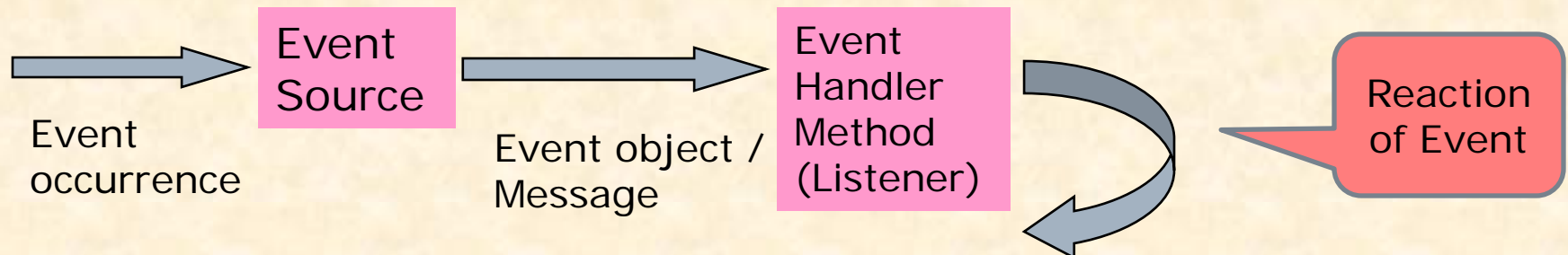
It is the GUI component that generates the event, e.g. Button.

- ❑ **Event Object or Message:**

It is created when event occurs. It contains all the information about the event which includes Source of event and type of event etc.

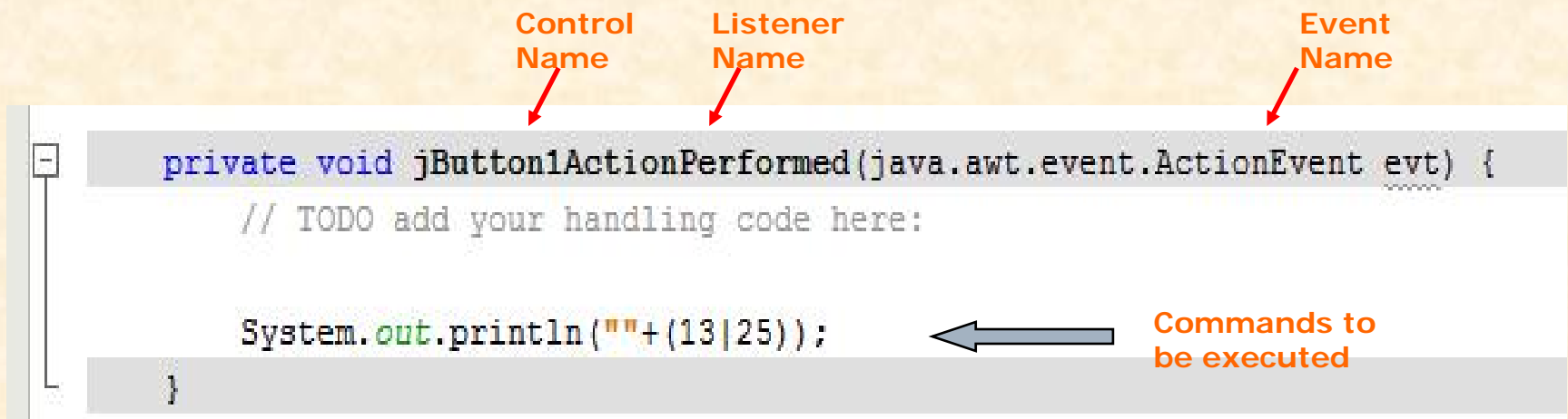
- ❑ **Event Handler (Event Listener) Method:**

It is implemented as in the form of code. It receives Event Message and handles events through Listener Interface.



How to use Event Handlers in NetBeans

- Right Click on desired control of the Form in Design view.
- Choose desired Event from the Context Menu and its Sub Menu.
- When you click on desired Event, it opens source Code editor with default Handler Name, where you can type TO DO code for the Handler. `ActionPerformed` is commonly used Listener.
- you will found a code window along with prototyped method to perform actions defined by you. You may write commands to be executed in **//TODO section**.



Working with Container Control- JFrame

- ❑ Every Swing Application must have at least one Top Level container (JFrame, JApplet, JDialog). A Top Level Container may have Mid-Level Container like Content Pane (JPanel, JMenuBar, JScrollBar etc.) or Components (JButton, JTextField etc.)
- ❑ A Frame (JFrame) is a Top Level (Desktop) container control having Title Border and other Properties.

Properties	Value	Description
title	Text	Sets the title (appears on the top) of the frame
cursor	Crosshair, East Resize, West Resize, Northwest Resize, Move, Hand, Wait, Default cursor	Specifies the type of mouse cursor when mouse moves on the frame.
Resizable	True /false	If checked, allows resizing of the frame
defaultCloseOperation	DO_NOTHING, HIDE, DISPOSE, EXIT_ON_CLOSE	Defines the action when close button is pressed.

Working with Panel- JPanel

- ❑ A Panel is container that holds other components displayed on the frame.
- ❑ To add Panel, just drag JPanel component on the frame and resize it.
- ❑ Drag other components (JButton, JTextFields etc.) from the Swing Control Box and drop it onto panel.

Properties	Value	Description
Background	Color	Sets the background color.
Border	No Border, Bevel Border, Compound Border, Empty order, Etched Border, Line Border, Titled Border etc.	Specifies the type of Border applies on the boundary of the panel.
Foreground	Color	Sets the foreground color.
ToolTipText	Text	Sets the text for tooltip.
MinimumSize	X, Y values	Defines the minimum width and height (x,y) in Twips(1/1440 inch)
MaximumSize	X, Y values	Defines the maximum(x,y) size.
PreferredSize	X, Y values	Defines the preferred (x,y) size.

Working with Push Buttons- JButton

- ❑ A button belongs to JButton class of Swing control API.
- ❑ It is mostly used action component, and can be used to trigger the associated events/methods in the application, when user clicks.

Properties	Value	Description
background	Color	Sets the background color. It works only when <u>contentAreaFilled</u> is set to True.
foreground	Color	Sets the foreground color.
toolTipText	Text	Sets the text for tool tip.
Text	Text	Caption of button.
mnemonic	Shortcut or Access key	Assign Shortcut key (Alt +key).
enabled	True/False	Determines whether Active or not.
font	Font name	Sets the font for the text of button.

Working with JButtons..

- Commonly used methods of JButton.

Method	Description
setText()	Sets the text displayed on the button. Ex. <code>JButton1.setText("You Clicked Me");</code>
getText()	Returns the text displayed on the button. Ex. <code>String result=JButton1.getText();</code> <code>JLabel1.setText(result);</code>
setIcon()	Sets the icon file to be displayed. Ex. <code>JButton1.setIcon(new ImageIcon</code> <code>("c:\\abc.png"));</code>
setEnabled(boolean)	Enables or disables the button. Ex. <code>JButton1.setEnabled(false);</code>
setVisible(boolean)	Makes the button visible or invisible. Ex. <code>JButton1.isVisible(false)</code>

Working with JLabel control

A Label control belongs to JLabel class and used to display non-editable text. The text to be displayed is controlled by `text` property (design time) and `setText()` method at run time. JLabel offers the following features-

- ❑ It can display Text or Image or both.
 - ❑ It may have bordered appearance.
 - ❑ Supports HTML for formatted text.
-

Commonly used Properties & Methods of JLabel

Properties	Value	Description
background	Color	Sets the background color. It works only when opaque is set to True.
foreground	Color	Sets the foreground color.
Text	Text	Sets the text to be displayed.
Font	Font name and size	Defines the font and size of text.
enabled	True/False	Determines whether Active or not
Icon	Image file to be displayed	Specifies the image file to be displayed.

Methods	Description
setText(String)	Sets the string of text to be displayed. Ex. jLabel1.setText("I am OK");
getText()	Returns the text displayed by the label. Ex. String st=jLabel1.getText();
setVisible(boolean)	Makes the Label visible or invisible. Ex. jLabel1.setVisible(false);

Displaying Image with JLabel

□ Setting up Image at Design Time :-

- Add JLabel control and click on ellipse (...) of **Icon property** in property window.
- In the dialogue box, select Image chooser option.
- Specify the path and file name in External Image option.
- Open source editor and go to top of the code and write-
`import javax.swing.ImageIcon;`

□ Setting up Image at Run time:-

- Import the javax library by placing following command at top of the code.
`import javax.swing.ImageIcon;`
 - Use the following command in a Event method where image to be displayed or changed.
`jLabel.setIcon(new ImageIcon("c:\\abc.png"))`
-

Working with JTextField control

A JTextField is a versatile control, used to get input from user or to display text. It is an object of JTextField class and allow the user to enter a **single line of text**.

JTextField offers the following features-

- ☐ You can insert and select text.
 - ☐ You can scroll the text, if not fit in visible area.
 - ☐ You can use selected text in other application using clipboard.
-

Commonly used Properties of JTextField

Properties	Value	Description
background	Color	Sets the background color.
foreground	Color	Sets the foreground color.
Text	Text	Sets the text to be displayed.
Font	Font name and size	Defines the font and size of text.
enabled	True/False	Determines whether Active or not
editable	True/False	Allow user to edit text, if set to true.

Methods	Description
setText(String)	Sets the string of text to be displayed. Ex. jTextField1.setText("I am OK");
getText()	Returns the text displayed by the label. Ex. String st=jTextField1.getText();
isEditable()	Returns the setting whether it is editable or not. Ex. Boolean b=jTextField1.isEditable();
isEnabled()	Returns the setting whether it is enabled or not. Ex. Boolean b=jTextField1.isEnabled();

Text interaction using JTextFields

In GUI application often we require to store the values of text fields to variable or vice-versa. Java offers three methods for this purpose-

❑ **getText():**

It returns the text stored in the text based GUI components like Text Field, Text Area, Button, Label, Check Box and Radio Button etc. in string type.

e.g. `String str1=jTextField1.getText();`

❑ **parse.....()**

This method converts textual data from GUI component into numeric type.

<code>Byte.parseByte(String s)</code>	– string into byte.
<code>Short.parseShort(String s)</code>	– string into short.
<code>Integer.parseInt(string s)</code>	– string into integer.
<code>Long.parseLong(string s)</code>	– string into long.
<code>Float.parseFloat(string s)</code>	– string into float.
<code>Double.parseDouble(string s)</code>	– string into double.

e.g. `int age=Integer.parseInt(jTextField1.getText());`

❑ **setText()**

This method stores string into GUI component.

e.g. `jTextField1.setText("Amitabh");`
`jLabel1.setText(""+payment);`

Alternatively it can be written as-
`jLabel1.setText(Integer.toString(payment));`

Working with jCheckBox control

A jCheckBox control belongs to JCheckBox class of Swing controls. It indicates whether a particular condition is on or off. You can use Check boxes to accept user's choice in terms of true/false or yes/no options.

Check Boxes may works independently to each other, so that any number of check boxes can be selected at the same time.

Some features of jCheckBox control's are-

- ❑ It can be used to input True/False or Yes/No typed input to the application.
 - ❑ Multiple check boxes can be selected at the same time.
-

Commonly used Properties of JCheckBox

Properties	Value	Description
background	Color	Sets the background color.
foreground	Color	Sets the foreground color.
Text	Text	Sets the text to be displayed.
Label	Text/Picture	Sets the text/Picture to be displayed.
Font	Font name and size	Defines the font and size of text.
enabled	True/False	Determines whether Active or not
mnemonic	Character	Specifies the shortcut (access) key
selected	True/false	Check box will be selected, if set to true. (default is false)
Button Group	Button Group name	Adds Check Boxes in a Group

Commonly used Methods of JCheckBox

Methods	Description
setText(String)	Sets the string of text to be displayed. Ex. jCheckBox1.setText("Computer");
getText()	Returns the text displayed by on the check box. Ex. String st=jCheckBox1.getText();
setEnabled()	Sets the check box enables, if true is given. Ex. jCheckBox1.setEnabled(true);
isEnabled()	Returns the state whether check box is enabled. Ex. Boolean st=jCheckBox1.isEnabled();
setSelected()	Sets the check box selected, if true is given. Ex. jCheckBox1.setSelected(true);
isSelected()	Returns the state whether check box is selected or not. Ex. If(jCheckBox1.isSelected()) { }

Working with JRadioButton control

A JRadioButton control belongs to JRadioButton class of Swing controls. It is used to get choices from the user. Generally, It is used as a grouped control, so that only one can be selected at a time.

By default, JButtons are independent control. To make them dependent, Radio Buttons should be kept in a **ButtonGroup** container control, so that only one button can be selected at a time in a group.

Some features of JRadioButton control's are-

- ❑ It can be used to get true/false or yes/no choice-typed input for the application.
 - ❑ They must be kept in a Button Group container control to form a group.
-

Commonly used Properties of JRadioButton

Properties	Value	Description
Background	Color	Sets the background color.
Foreground	Color	Sets the foreground color.
ButtonGroup	Name of control	Specifies the name of group to which Radio Button belongs.
Text	Text	Sets the text to be displayed.
Label	Text/Picture	Sets the text/Picture to be displayed.
Font	Font name and size	Defines the font and size of text.
Enabled	True/False	Determines whether Active or not
Mnemonic	Character	Specifies the shortcut (access) key
Selected	True/false	Button will be selected, if set to true. (default is false)

You must add a **ButtonGroup control** to the frame to group the Radio Buttons by using ButtonGroup property to facilitate only-one selection.

Commonly used Methods of JRadioButton

Methods	Description
setText(String)	Sets the string of text to be displayed. Ex. JRadioButton1.setText("Science");
getText()	Returns the text displayed on the Button. Ex. String st=JRadioButton1.getText();
setEnabled()	Sets the Radio Button enables, if true is given. Ex. JRadioButton1.setEnabled(true);
isEnabled()	Returns the state whether radio button is enabled. Ex. boolean st=JRadioButton1.isEnabled();
setSelected()	Sets the Radio Button selected, if true is given. Ex. JRadioButton1.setSelected(true);
isSelected()	Returns the state whether Radio Button is selected or not. Ex. if(JRadioButton1.isSelected()) {..... }

Working with JList control

A List (or List box) is box shaped control containing list of objects, from which single or multiple selection can be made. JList control offers the following features-

- ❑ A box shaped control capable to displaying a list of choices (Text or graphics/images)
 - ❑ It allows single or multiple selection of items using mouse.
 - ❑ Equipped with built-in scroll bar to view a large list.
 - ❑ `valueChanged()` method of ListSelection Listener is used to handle the JList events
 - ❑ After attaching a JList control, **Model** property is used to specify the list items at design time.
-

Commonly used Properties of JList

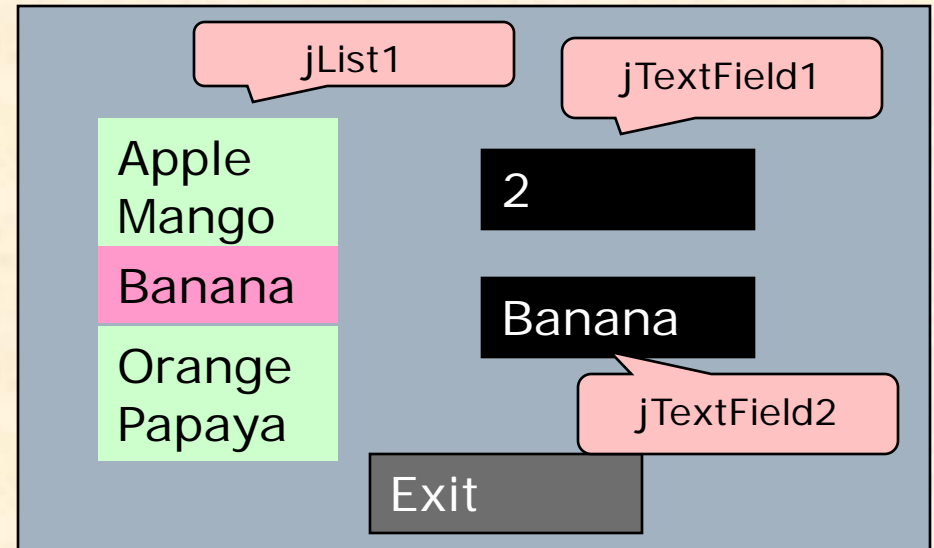
Properties	Value	Description
Background	Color	Sets the background color.
Foreground	Color	Sets the foreground color.
model	Items for Choice	Specifies the items to be displayed as a choice.
selectionMode	SINGLE SINGLE_INTERVAL MULTIPLE_INTERVAL	User may select single item. User may select Single range of items by holding SHIFT key. User may select Multiple range of Items by holding CTRL key
selectedIndex	Value	Specifies the index of Items to appear selected. (default is -1 since no items is selected.)
font	Font name	Specifies font's name and size etc.
enabled	True/False	Specifies that list will be active or not.

Commonly used Methods of JList

Methods	Description
clearSelection()	Clears the selection in the list. Ex. <code>JList1.clearSelection();</code>
getSelectedIndex()	Returns the index of selected items in single selection mode. Returns -1, if selection is not made. <code>int x= JList1.getSlectedIndex();</code> <code>if (JList1.getSelectedIndex()==1) { ... }</code>
getSelectedValue()	Returns the value or selected items. <code>String st= (String) JList1.getSlectedValue();</code> <code>if (JList1.getSelectedValue()=="Apple"){ ... }</code>
isSelectedIndex(int)	Returns True if given Index is selected. Ex: <code>if (JList1.isSlectedIndex(2)) { ... }</code>
Other methods like <code>isEnabled()</code> , <code>setVisible()</code> , <code>setEnabled()</code> etc. can be used with JList control.	

How to handle Selections in the JList

Suppose we want to display the index and value of selected items i.e. 2 and 'Banana' in Text Fields when user selects an item in the list.



Just Right Click on `JList` control and choose

Events->ListSelection->ValueChanged

Write the following code in //TO DO Section.....

```
int x = jList1.getSelectedIndex();  
String st = (String ) jList1.getSelectedValue();  
jTextField1.setText(""+x);  
jTextField2.setText(st);
```

Working with JComboBox control

A JComboBox (TextField + List Box) is a control which offers the list of choice which can be selected through a drop-down list.

By default it is an un-editable control, but we can make it editable by setting 'editable' property True.

`JComboBox1ActionPerformed(..)` method can be handled when user selects an item from the combo.

Difference Between List & Combo Box

- ❑ In Combo Box, user can select and edit an item but in List, Items can be selected and can not be edited.
- ❑ List does not offers Text Field where as Combo Box offers Text Field.
- ❑ Combo Box offers a Drop-down list, so that it takes less space in the frame, but List this feature is not available.
- ❑ List allows more than one items to be selected, but in Combo Box, only one item can be selected.

Commonly used Properties of JComboBox

Properties	Value	Description
Background	Color	Sets the background color.
Foreground	Color	Sets the foreground color.
model	Items for Choice	Specifies the items to be displayed as a choice.
selectedIndex	Value	Specifies the index of Items to appear selected (default is -1 since no items is selected.)
selectedItem	String/values	Specifies the indices of selected items.
font	Font name	Specifies font's name and size etc.
editable	True/False	If True, you can edit/type new value or choice in the Combo Box.
enabled	True/False	Specifies that list will be active or not.

Commonly used Methods of JComboBox

Methods	Description
<code>getSelectedIndex()</code>	Returns the index of selected items. <code>if (jComboBox1.getSelectedItem()==1) {...};</code>
<code>getSelectedItem()</code>	Returns the selected items. <code>if (jComboBox1.getSelectedItem()=="Mango") {..... ..};</code>
<code>isEditable()</code>	Returns True, if Combo box is editable.
<code>addItem(string)</code>	Adds an item to the choice list at the end. <code>Ex. jComboBox1.addItem("Banana");</code>
<code>getItemCount()</code>	Returns the number of items in the combo Box. <code>int x = jComboBox1.getItemCount();</code>
<code>getItemAt(int)</code>	Returns the items at specified index. <code>String st=jComboBox1.getItemAt(2);</code>
<code>removeAllItems()</code>	Removes all the items from combo.
<code>removeItemAt(index)</code>	Removes items for given index from the combo. <code>Ex. jComboBox1.removeItemAt(2);</code>

Working with JTextArea control

A JTextArea control is a multi-line text component, used to get input from user or to display text. It is an object of JTextArea class.

By default, it does not wrap (move next line) lines of text like word processor, if line goes beyond the boundary. Some features are-

- ☐ You can insert and select multiple line of text.
 - ☐ You can wrap text, if not fit in visible area.
 - ☐ You can use selected text in other application using clipboard.
-

Commonly used Properties of JTextArea

Properties	Value	Description
Background	Color	Sets the background color.
Foreground	Color	Sets the foreground color.
LineWrap	True/ false	Defines Wrapping feature enable/disable
Rows	number	Set the number of rows of in text area
Columns	number	Sets the number of columns
Text	Text	Sets the text to be displayed.
Font	Font name and size	Defines the font and size of text.
Enabled	True/False	Determines whether Active or not
Editable	True/False	Allow user to edit text, if set to true.

Commonly used Methods of JTextArea

Methods	Description
setText()	Sets the string of text to be displayed. Ex. <code>TextArea1.setText("I am OK");</code>
getText()	Returns the text displayed by the label.
setEditable()	Sets the TextArea editable.
isEditable()	Returns the setting whether it is editable or not. Ex. <code>boolean b=TextArea.isEditable();</code>
append()	Adds specified text in the text area. Ex: <code>TextArea1.append("How are you");</code>
Insert(string, int)	Inserts specified text at given position. Use 0 to insert at top. Ex. <code>TextArea1.insert("Amit",1);</code>
setLineWrap()	Enables or disables line wrap feature. Ex. <code>TextArea1.setLineWrap(true);</code>
isEnabled()	Returns the status whether it is enabled or not. Ex. <code>boolean b=TextArea.isEnabled();</code>

Working with JPasswordField Control

- ❑ A JPasswordField is a type of Text field that shows Encrypted text (actual text is not shown for security purpose) like '*' or any other specified character as you type.
- ❑ The character displayed in place of typed character is called echo Character, which is controlled by echoChar property.

Properties	Value	Description
Background	Color	Sets the background color.
Foreground	Color	Sets the foreground color.
Text	Text	Sets the text to be displayed.
Font	Font and size	Defines the font and size of text.
enabled	True/False	Determines whether Active or not
editable	True/False	Allow user to edit text, if set to true.
echoChar	Character	Specifies the character to be displayed in place of typed character.

Commonly used Methods of JPasswordField

Methods	Description
setEchoChar(char)	Sets the echo character. Ex. <code>JPasswordField1.setEchoChar('#');</code>
getPassword() *	Returns the text displayed by the password field. String <code>pwd= new String (JPasswordField1.getPassword());</code>

* getPassword() method returns a character array, not a string. To store it in a string variable you need to use constructor of string.

Comparing Strings in JAVA

In Java two strings can not be compared directly, by using == operator.

Two methods `equals()` and `compareTo()` are used for this purpose.

`equals()` returns TRUE/FALSE but `compareTo()` returns 0 if both are equal otherwise non-zero value is returned.

Ex. `if (pwd.equals("abc")) {.....} else {.....}`

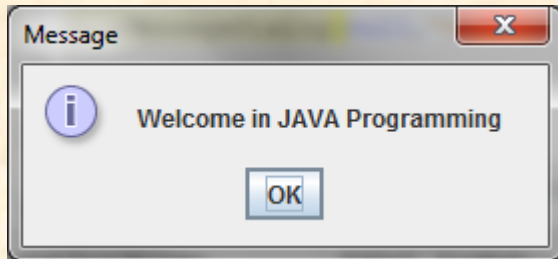
JOptionPane : Built-In Dialog of JAVA

JOptionPane allows to create pop-up window or dialog with predefined style.

Java provides following types of Dialogs which can be used as per requirement.

Dialog Types	Method	Description
Message Dialog	showMessageDialog()	Used to inform user by displaying a message. It includes OK button only.
Input Dialog	showInputDialog()	Used to get user input using Text Field.
Confirm Dialog	showConfirmDialog()	Used to ask a user to confirm some information with Yes/No or OK/Cancel buttons.

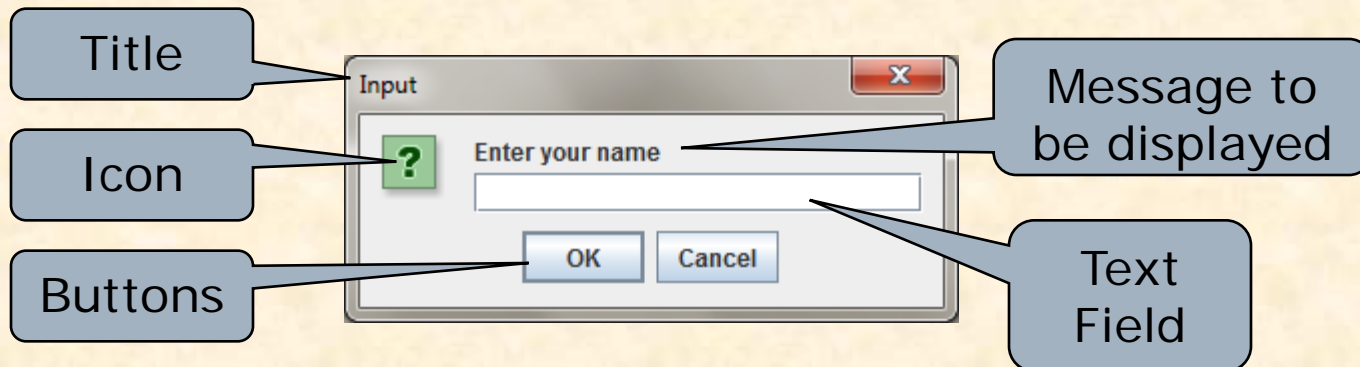
JOptionPane Dialog Types:



Message Dialog



Confirm Dialog



Input Dialog

Working with JOptionPane

To use the JOptionPane dialog control in the application, you must import the following class(s).

```
import javax.swing.JOptionPane;
```

In general, the following syntax of methods along with optional parameters can be used-

JOptionPane.show.....([frame name],<"Message"> [,<"Title">]);

❑ **Frame Name:** Generally **null** is used to indicate current frame.

❑ **Message:** User given string to convey the message.

❑ **Title:** Text to be displayed as Title on the Title bar.

Example(s):

```
JOptionPane.showMessageDialog(null,"JAVA Welcomes You");
```

```
JOptionPane.showMessageDialog (null,"My Name is "+name);
```

```
String n= JOptionPane.showInputDialog (null,"Enter your Name ?");
```

```
String n= JOptionPane.showInputDialog (null,"Enter your Name ?", "Input Name");
```

```
int ch=JOptionPane.showConfirmDialog(null,"Want to exit ?");
```

```
int ch=JOptionPane.showConfirmDialog(null,"Want to exit ?", "Confirm ?");
```

Return Value of JOptionPane dialog

❑ Value returned by Input Dialog:

ShowInputDialog() returns a **string** value which can be directly assigned on a string type variable. You may use **parse...()** method to convert it into other data types like **int** or **float** etc. for further use in the application.

```
String n=JOptionPane.showInputDialog(null, "Enter Name? ");
```

```
private void jButton1ActionPerformed(...) {  
    // Program to calculate area of circle using Input dialog  
    String str=JOptionPane.showInputDialog(null, "Enter radius  
of circle ?");  
    int radius=Integer.parseInt(str);  
    float area=(22/7)*(radius*radius);  
    JOptionPane.showMessageDialog(null, "Area of circle="+area);  
}
```

Return Value of JOptionPane dialog

❑ Value returned by Confirm Dialog:

Sometimes it is required to check the response of user i.e. which button is pressed, so that application can respond accordingly.

.showConfirmDialog() returns **int** type value which can be compared with the following constants to determine the status of button pressed by the user.

Returned Value	Indication
JOptionPane.YES_OPTION	YES button is pressed
JOptionPane.OK_OPTION	OK button is pressed
JOptionPane.NO_OPTION	NO button is pressed
JOptionPane.CANCEL_OPTION	CANCEL button is pressed
JOptionPane.CLOSE_OPTION	User closed the dialog using X button

```
int ans=JOptionPane.showConfirmDialog(null, "Want to exit ?");  
If (ans==JOptionPane.YES_OPTION)  
    System.exit(0);
```

JDialog – A Simple General purpose dialog

- ❑ The JDialog is a swing window dialog equipped with Minimise, Maximise and Close functionality.
- ❑ JDialog creates a standard pop-up window for displaying desired information related to your application.
- ❑ Steps to Attach a JDialog Control-
 1. Design an application as required and drag Dialog control from swing palette and drop it into the application.
 2. To open Dialog frame, double click on jDialog node under other component in Inspector window.
 3. Attach message button and title in Dialog control as per requirement.
 4. Attach the following **ActionPerformed** event handler of attached button.

jDialog1.dispose();

5. Now switch to Frame by double clicking on JFrame node in Inspector window.
6. Attach the following code on button from where you want to run the dialog.

jDialog.setVisible(true);

Displaying Text/Values in JAVA GUIs

In GUI application often we require to display an information as a message or value stored in a variable. The following three ways are used –

❑ Using **JTextField**s or **JLabel** Controls-

```
jTextField1.setText("Area =" + a);  
jLabel1.setText("Hello Java");
```

❑ Using **Dialog Box**-

```
JOptionPane.showMessageDialog(null, "Hello.. ");  
JOptionPane.showMessageDialog(null, "Area=" + a);
```

❑ Displaying result at output window using **System.out.print()** method-

e.g.

```
System.out.print("Hello...");  
System.out.println("Area=" + a);
```

JAVA Tokens

- ❑ The smallest individual unit in a program is known as Token. It may any word, symbols or punctuation mark etc.
 - ❑ Following types of tokens used in Java-
 - ❖ Keywords
 - ❖ Identifiers
 - ❖ Literals
 - ❖ Punctuators (; [] etc)
 - ❖ Operators (+, -, /, *, =, == etc.)
-

Keywords in Java

- ❑ Keywords are the reserve words that have a special meaning to the compiler.
 - ❑ Key words can't be used as identifiers or variable name etc.
 - ❑ Commonly used key words are-
char, long, for, case, if, double, int, short, void, main, while , new etc.
-

Identifiers in Java

- ❑ Identifiers are fundamental building block of program and used as names given to variables, objects, classes and functions etc.
 - ❑ The following rules must be followed while using identifiers.
 - Identifiers may have alphabets, digits and dollar (\$), underscore (_) sign.
 - They must not be Java keywords.
 - They must not begin with digit.
 - They can be of any length.
 - They are Case Sensitive ie. Age is different from age.
 - ❑ Example of Valid identifiers-
MyFile, Date9_7_7, z2t09, A_2_Z, \$1_to_100, _chk etc.
 - ❑ Example of Invalid identifiers-
Date-RAC, 29abc, My.File, break, for
-

Literals in Java

- ❑ Literals or constants are data items that have fixed data value.
 - ❑ Java allows several types of literals like-
 - Integer Literals
 - Floating Literals
 - Boolean Literals
 - Character Literals
 - String Literals
 - The null literals
-

Integer Literals

- ❑ An integer constant or literals must have at least one +/- digit without decimal point.
 - ❑ Java allows three types of integer literals -
 - Decimal Integer Literals (Base 10)
e.g. 1234, 41, +97, -17 etc.
 - Octal Integer Literals (Base 8)
e.g. 010, 014 (Octal must start with 0)
 - Hexadecimal Integer Literals (Base 16)
e.g. 0xC, 0xab (Hex numbers must start with 0x)
 - ❑ L or U suffix can be used to represent long and unsigned literals respectively.
-

Floating / Real Literals

- A real literals are fractional numbers having at least one digit before and after decimal point with + or – sign.

The following are valid real numbers-

2.0, 17.5, -13.0. -0.00626

The following are invalid real numbers-

7, 7. , +17/2 and 17,250.26 etc.

- A real literals may be represented in Exponent form having Mantissa and exponent with base 10 (E). Mantissa may be a proper real numbers while exponent must be integer.

The following are valid real in exponent form-

152E05, 1.52E07, 0.152E08, -0.12E-3, 1.5E+8

The following are invalid real exponent numbers-

172.E5, 1.7E, 0.17E2.3, 17,22E05, .25E-7

Other Literals

- ❑ The Boolean Literals represents either TRUE or FALSE. It always Boolean type.
- ❑ A null literals indicates nothing. It always null type.
- ❑ Character Literals must contain one character and must enclosed in single quotation mark.

e.g. 'a', '%' , '9' , '\\' etc.

Java allows some non-graphic characters (which can not be typed directly through keyboard) by using Escape sequence (\) .

The following Escape characters are commonly used in Java.

\a (alert),	\b (backspace),	\f (Form feed),
\n (new line),	\r (return key),	\t (Horizontal tab),
\v (vertical tab),	\\ (back slash),	\' (single quote) ,
\\" (double quote) ,	\? (question mark),	\0 (null) etc.

- ❑ String Literals is a sequence of zero or more characters enclosed in double quotes. E.g. "abs", "amit" , "1234" , "12 A" etc.
-

Data types in JAVA

Data types are means to identify the type of data and associated operations of handling it.

Java offers two types of data types.

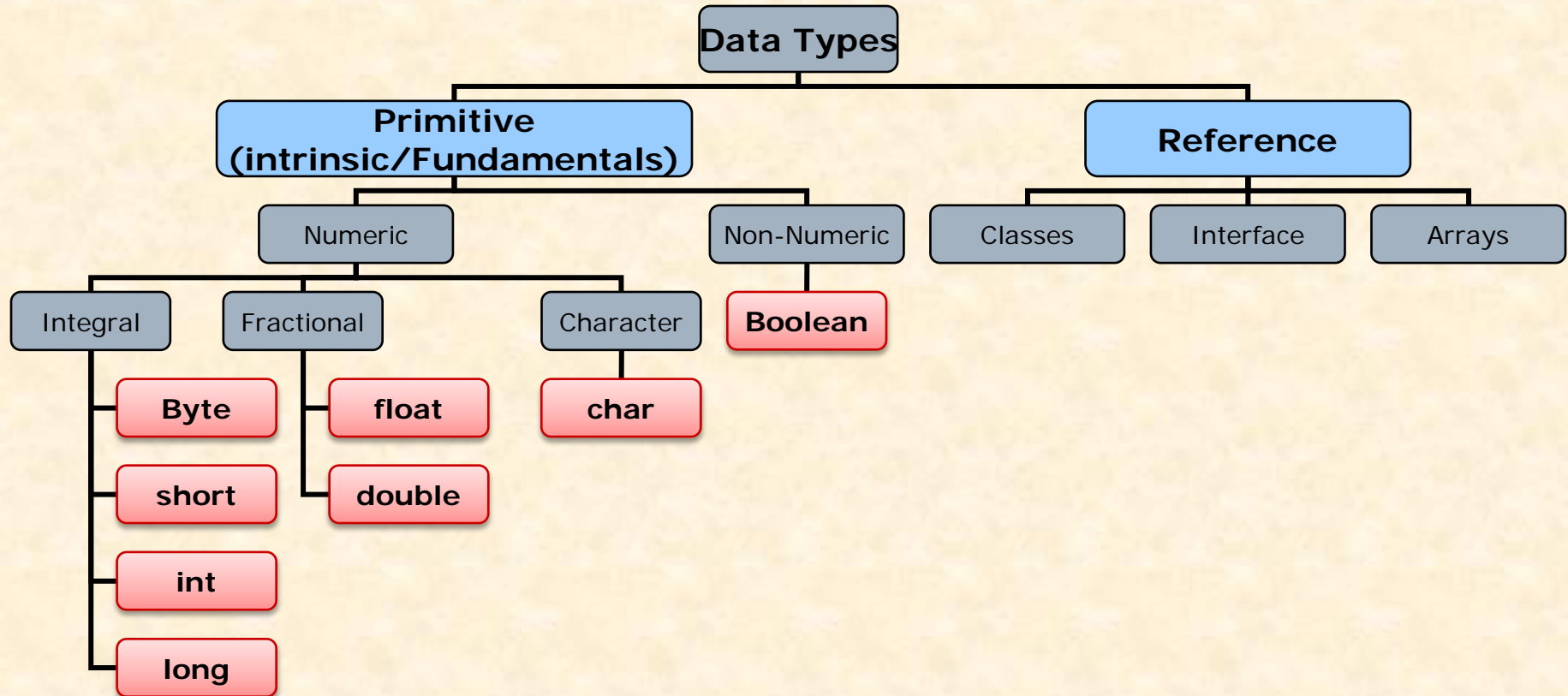
□ Primitive:

These are in-built data types offered by the compiler. Java supports 8 primitive data types e.g. byte, short, int, long, float, double, char, boolean.

□ Reference:

These are constructed by using primitive data types, as per user need. Reference data types store the memory address of an object. Class, Interface and Array are the example of Reference Data types.

Data Types in Java



String Data type is also used in Java as Reference data type

Primitive Data types

Type	Size	Description	Range
byte	1 Byte	Byte integer	-128 to +127
short	2 Byte	Short integer	-32768 to +32767
int	4 Byte	integer	-2^{31} to $2^{31}-1$
long	8 Byte	Long integer	-2^{63} to $2^{63}-1$
float	4 Byte	Single precision floating point (up to 6 digit)	-3.4×10^{-38} to $+3.4 \times 10^{38}$
double	8 Byte	Double precision floating (up to 15 digit)	-1.8×10^{-308} to $+1.8 \times 10^{308}$
char	2 Byte	Single character	0 to 65536
Boolean	1 Byte	Logical Boolean values	True or False

- **L** suffix can used to indicate the value as long.
 - By default Java assume frictional value as double, **F** and **D** suffix can be used with number to indicate float and double values respectively.
-

Working with Variables

- ❑ A variable is named memory location, which holds a data value of a particular data type.
 - ❑ **Declaration and Initialization of variable-**
<data type> <variable Name>;
 int age;
 double amount;
 double price=214.70, discount =0.12;
 String name="Amitabh"
 long x=25L;
 byte a=3;
 float x= a+b;
 - ❑ By default all Numeric variables initialized with 0, and character and reference variable with null, boolean with false, if it is not initialized.
 - ❑ The keyword **final** can be used with variable declaration to indicate **constant**.
E.g. **final double SERVICE_TAX=0.020;**
-

Operators in Java

- ❑ The operators are symbols or words, which perform specified operation on its operands.
 - ❑ Operators may Unary, Binary and Ternary as per number of operands it requires.
 - ❑ Java offers the following types of Operators:-
 - ❖ Arithmetic Operator
 - ❖ Increment/Decrement Operator
 - ❖ Relational or Comparison Operators
 - ❖ Logical Operators
 - ❖ Assignment Operators
 - ❖ Other Operators.
-

1. Arithmetic Operators

+	Unary plus	Represents positive values.	<code>int a = +25;</code>
-	Unary minus	Represents negative values.	<code>int a = -25;</code>
+	Addition	Adds two values	<code>int x = a + b;</code>
-	Subtraction	Subtract second operands from first.	<code>int x = a - b;</code>
*	Multiplication	Multiplies two values	<code>int x = a * b;</code>
/	Division	Divides first operand by second	<code>int x = a / b;</code>
%	Modulus (remainder)	Finds remainder after division.	<code>int x = a % b;</code>
+	Concatenate or String addition	Adds two strings	<code>"ab" + "cd" ⇨ "abcd"</code> <code>"25" + "12" ⇨ "2512"</code> <code>"" + 5 ⇨ "5"</code> <code>"" + 5 + "xyz" ⇨ "5xyz"</code>

2. Increment & Decrement Operator

- Java supports `++` and `--` operator which adds or subtract 1 from its operand. i.e.

`a=a+1` equivalent to `++a` or `a++`

`a=a-1` equivalent to `--a` or `a--`

- `++` or `--` operator may be used in **Pre** or **Post** form.

`++a` or `--a` (increase/decrease before use)

`a++` or `a--` (increase/decrease after use)

Ex. Find value of P? (initially `n=8` and `p=4`)

`p=p*--n; ⇒ 28`

`p=p*n--; ⇒ 32`

Ex. Evaluate `x=++y + 2y` if `y=6`.

$$= 7 + 14 = 21$$

3.Relational Operator

- ❑ Relational operators returns true or false as per the relation between operands.
- ❑ Java offers the following six relational operators.

<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal to

Relational operators solved from left to right.

Ex: $3 >= 3$ true

$3 != 3$ false

$a == b$ true if a and b have the same value.

$a < b < c ==> (a < b) < c$ true if a is smallest.

4.Logical Operator

- ❑ Logical operators returns true or false as per the condition of operands. These are used to design more complex conditions.
- ❑ Java offers the following six (5 binary and 1 unary) logical operators.

Operator	Name	use	Returns true if
&&	And	$x \& y$	X and y both true
	Or	$x y$	Either x or y is true
!	Not	$!x$	X is false
&	Bitwise and	$x \& y$	X and y both true
	Bitwise or	$x y$	Either x or y is true
^	Exclusive or	$x \wedge y$	If x and y are different

Ex: $5 > 8 || 5 < 2$ (false)

$6 < 9 \&\& 4 > 2$ (true)

$!(5 != 0)$ (false)

$1 == 0 || 0 > 1$ (false)

$6 == 3 \&\& 4 == 4$ (false)

$!(5 > 9)$ (true)

5. Assignment Operator

- In Java = operator is known as Assignment operator, it assigns right hand value to left hand variables.

Ex: `int x=5;`

`z= x+y;`

- Java offers some special shortened Assignment operators, which are used to assign values on a variable.

Operator	use	Equivalent to
<code>+=</code>	<code>X+=y</code>	<code>X=X+Y</code>
<code>-=</code>	<code>X-=y</code>	<code>X=X-Y</code>
<code>*=</code>	<code>X*=y</code>	<code>X=X*Y</code>
<code>/=</code>	<code>x/=y</code>	<code>X=X/Y</code>
<code>%=</code>	<code>X%=y</code>	<code>X=X%Y</code>

Ex: `x-=10` => `x=x-10`

`x%=y` => `x=x%y`

6. Other Operators

- ❑ In Java some other operators are also used for various operations. Some operators are-

Operator	Equivalent to
? :	Shortcut of If condition (ternary operator) <Condition> ? <true action> : <false action>
[]	Used to declare array or access array element
.	Used to form qualified name (refer)
(type)	Converts values as per given data type
new	Creates a new object
instanceof	Determines whether the object is an instance of a class.

Ex. result = marks >= 50 ? 'P' : 'F'

6 > 4 ? 9 : 7 evaluates 9 because 6 > 4 is true.

Operator's Precedence

Operator's precedence determines the order in which expressions are evaluated. There is certain rules called Precedence for evaluating a complex expression.

e.g. $y=6+4/2$ (why 8 not 5 ?)

Operators	Remark	Associativity
. [] ()	() used to make a group, [] used for array and . Is used to access member of object	L to R
++ -- !	Increment/Decrement Operator, Not Operator Returns true or false based on operands	R to L
New, (type)	New is used to create object and (type) is used to convert data into other types.	R to L
* / %	Multiplication, division and modulus	L to R
+ -	Addition and Subtraction	R to L
== !=	Equality and not equality	L to R
& , ^,	Bitwise And, Bitwise Exclusive Or , Bitwise or	L to R
&&	Logical And	L to R
	Logical or	L to R
? :	Shortcut of IF	R to L
=, +=, -=, *=, /=, % =	Various Assignment operators	R to L

Expression in Java

- ❑ An expression is a valid combination of operators, constants and variable and keywords i.e. combination of Java tokens.
- ❑ In java, **three** types of expressions are used.
- ❑ **Arithmetic Expression**

Arithmetic expression may contain one or more numeric variables, literals and operators. Two operands or operators should not occur in continuation.

e.g. $x + *y$ and $q(a + b - z/4)$ are invalid expressions.

Pure expression: when all operands are of same type.

Mixed expressions: when operands are of different data types.

- ❑ **Logical Expression**

Logical or Boolean expression may have two or more simple expressions joined with relational or logical operators.

e.g. ♦ $x > y$ ♦ $(y + z) >= (x / z)$ ♦ $x || y \&\& z$

- ❑ **Compound Expression**

It is combination of two or more simple expressions.

e.g. ♦ $(a + b) / (c + d)$ ♦ $(a > b) || (b < c)$



Data Type Conversion in JAVA

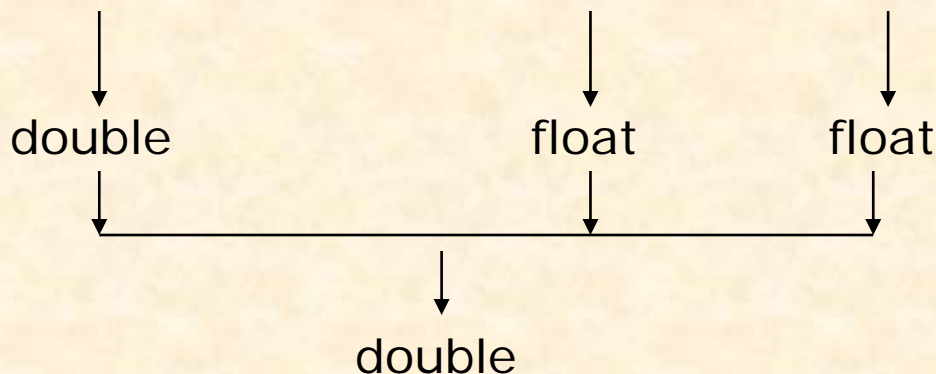
- ❑ The process of converting one predefined type into another is called type conversion.
 - ❑ In mixed expression, various types of constant and variables are converted into same type before evaluation.
 - ❑ Java facilitates two types of conversion.
 - ✓ Implicit type conversion
 - ✓ Explicit type conversion
-

Implicit Type Conversion

- It is performed by the compiler, when different data types are intermixed in an expression. It is also called **Coercion**.
- In Implicit conversion, all operands are promoted (Coercion) up to the **largest data** type in the expression.

Ex. Consider the given expression, where f is float, d is double and I is integer data type.

Result= (f * d) - (f + i) + (f / i)



Explicit Conversion in JAVA

- ❑ An explicit conversion is user defined that forces to convert an operand to a specific data type by (type) cast.

e.g. (float) (x/2) suppose x is integer.

The result of x/2 is converted in float otherwise it will give integer result.

- ❑ In pure expression the resultant is given as expression's data type.

e.g. 100/11 will give 9 not 9.999 (since both are integer)

- ❑ In mixed expression the implicit conversion is applied (largest type promotion)

e.g. int a, mb=2, k=4 then evaluate

$$a = mb * 3/4 + k/4 + 8 - mb + 5/8$$

$$= 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8$$

$$= 6 / 4 + 1 + 8 - 2 + 5 / 8$$

$$= 1 + 1 + 8 - 2 + 0 \quad (6/4 \text{ will give } 1)$$

$$= 8$$

JAVA Statements

- ❑ A statement in Java is a complete unit of execution. It may consists of Expression, Declaration, Control flow statements and must be ended with semicolon (;)
- ❑ Statements forms a block enclosed within { }. Even a block may have no statement (empty).

E.g. If(a>b)
 {

 }

❑ Just Think..... Why?

System.out.print('h'+'a') will give 169

System.out.print(" "+'h'+'a') will give ha

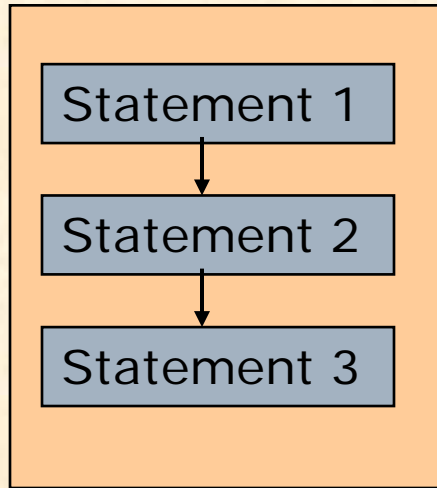
System.out.print("2+2=")+2+2) will give 2+2=22

System.out.print("2+2=")+(2+2)) will give 2+2=4

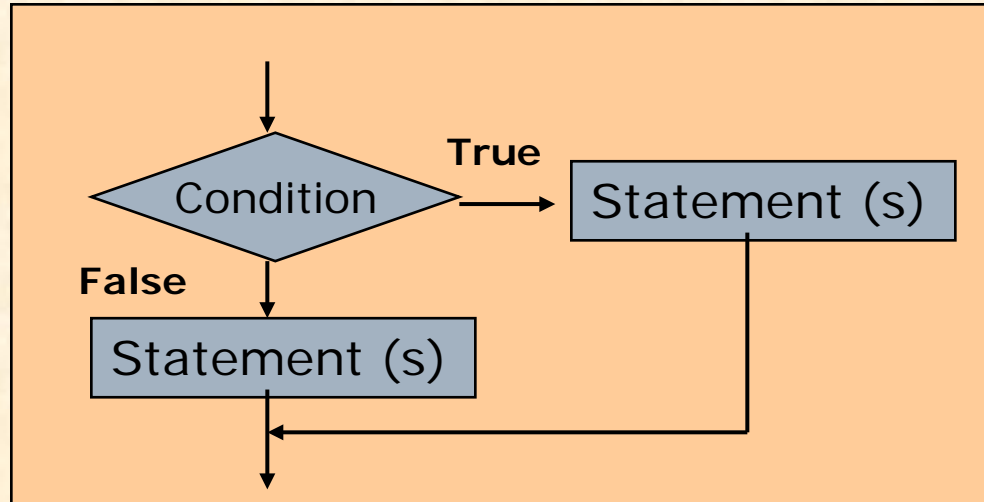
Programming Constructs in JAVA

- ❑ In general a program is executed from begin to end. But sometimes it required to execute the program selectively or repeatedly as per defined condition. Such constructs are called control statements.
 - ❑ The programming constructs in Java, are categorized into -
 - **Sequence:**
Statements are executed in top-down sequence.
 - **Selection (conditional):**
Execution of statement depends on the condition, whether it is True or False.
(Ex. if.., if...else, switch constructs)
 - **Iteration (Looping):**
Statement is executed multiple times (repetition) till the defined condition True or False.
(Ex. for.. , while... , do..while loop constructs)
-

Diagrammatic Representation

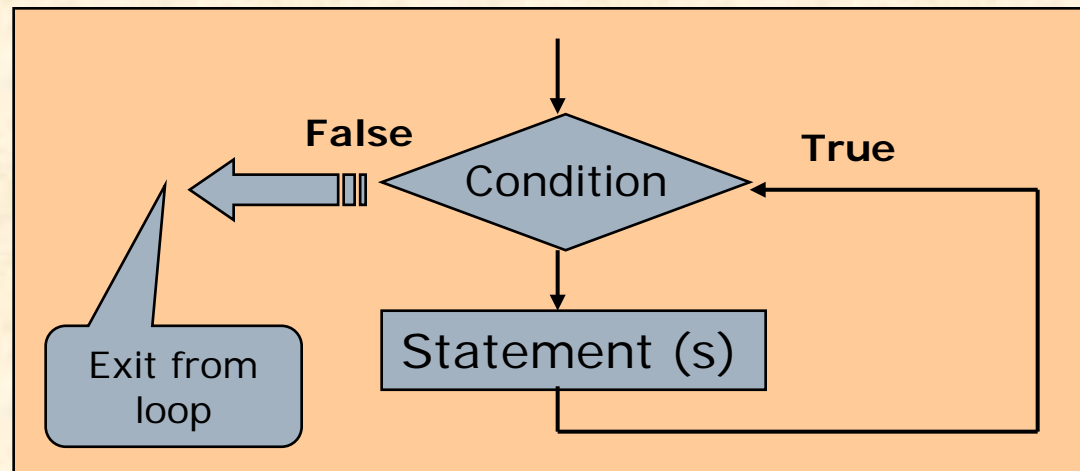


Sequence construct



Selection construct

Iteration Construct



Selection statement (if..)

- ❑ The if... statement is used to test a condition. If defined condition is true the statements are executed otherwise they are ignored.
- ❑ The condition may be simple or complex (using &&, || etc.) and must be enclosed in ().
- ❑ Expression defined as condition must be evaluated as True or False.

Syntax

```
if (condition)
{
    statement 1 ;
    .....
}
```

```
if ( num>0) {
    JLabel1.setText("Number is positive");
}
```

```
if ( ch>='0' && ch<='9' ) {
    JLabel1.setText("It is digit");
}
```

In case of single statement in if... the use of { } is optional.

Selection statement (if..else..)

- ❑ The if...else.. also tests condition. If defined condition is true the statements in **True block** are executed otherwise **else block** is executed.

```
if (condition)
{
    statement 1 ;
    .....
}
else
{
    statement 2;
    .....
}
```

```
if ( num>0) {
    jLabel1.setText("Number is positive");
}
else
{
    jLabel1.setText("Number is zero or negative");
}
```

```
if ( age >= 18)
    jLabel1.setText("Eligible to Vote");
else
    jLabel1.setText("Not eligible to Vote");
```



In case of single statement `{ }` is optional.

Nested if...

- ❑ An if... or if..else.. may have another if.. Statement in its true block or in false block. It is known as Nesting of if (placing an if inside another if).

```
if (condition1){  
    if(condition 2)  
    { ..... }  
    else  
    { ..... }  
}  
else{  
    if(condition 3)  
    { ..... }  
    else  
    { ..... }  
}
```

```
if ( num>0)  
{  
    JLabel1.setText("Number is positive");  
}  
else  
{ if (num<0)  
    JLabel1.setText("Number is negative");  
  else  
    JLabel1.setText("Number is zero");  
}
```



Nested if.. block

If...else...If ladder

- When a else block contains another if.. and this nested else block may have another if and so on. This nesting is often known as if..else..if ladder or staircase.

```
if (condition1)
    statement 1;
else
    if (condition 2)
        statement 2;
    else
        if (condition 3)
            statement 3;
        else
            if(condition 4)
                statement 4;
            .....
            .....
        else
            statement n;
```

```
if (day==1)
    JLabel.setText("Sunday");
else
    if (day==2)
        JLabel.setText("Monday");
    else
        if (day==3)
            JLabel.setText("Tuesday");
        else
            if(day==4)
                JLabel.setText("Wednesday ");
            else
                if(day==5)
                    JLabel.setText("Thrusday");
                else
                    if(day==6)
                        JLabel.setText("Friday");
                    else
                        JLabel.setText("Saturday");
```

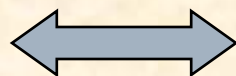
Conditional operator and if.. statement

- ❑ The ? : (conditional operator) may be used as alternative to if..else.. statement in Java.
- ❑ In contrast to if..., ?: is more concise and compact code it is less functional than 'if'.
- ❑ ?: produces an expression so that a single value may be incorporated whereas if.. is more flexible, whereas you may use multiple statements, expressions and assignments.
- ❑ When ?: is used as nested form, it becomes more complex and difficult to understand.

Syntax

```
if ( a>b)  
    c=a;  
else  
    c=b;
```

Expression 1 ? Expression 2: expression 3;




```
C = a>b ? a : b ;
```


The switch statement

- ❑ Multi-branching selection can be made in Java by using switch statement.
- ❑ It test successively, the value of an expression (short, int, long or char type), when match is found, the statements associated with constant is executed.

```
switch (<expression>
{ case <const 1> : statement (s);
                        break;
  case <const 2> : statement (s);
                        break;
  case <const 2> : statement (s);
                        break;
  .....
  [default : statement (s);]
}
```

- 
1. No two identical constant can be used.
 2. Default.. is optional and may be anywhere in switch block, if used.

```
switch (day)
{ case 1 : Dow="Sunday";
      break;
  case 2 : Dow="Monday";
      break;
  case 3 : Dow="Tuesday";
      break;
  case 4 : Dow="Wednesday";
      break;
  case 5 : Dow="Thursday";
      break;
  case 6 : Dow="Friday";
      break;
  case 7 : Dow="Saturday";
      break;
  default : Dow="Wrong Input";
}
jLabel.setText("Weak day"+Dow);
```

Switch and if..else statement

The switch and if..else both are used to make a selection construct in Java, but there are some differences.

- ❑ Switch can test only equality whereas if.. Can evaluate any type of relational or logical expression.
- ❑ In switch a single value or constant can be tested but in if.. more versatile expression can be tested.
- ❑ The switch statement can handle only byte, short, int or char variable but If.. can test more data type like float, double or string etc.

```
if ( condition 1)
{
    switch (exp1)
    {
        .....
        .....
    }
}
else
{
    .....;
}
```

```
switch (exp 1)
{
    case <cont>:switch (exp2)
    {
        .....
        .....
    }
    .....
}
```

Nesting of switch with switch or if.. may be used....

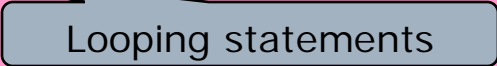
Iteration (looping) statements

- ❑ Iteration or looping allow a set of instructions to be executed repeatedly until a certain condition is true or false.
 - ❑ As per placing of condition to be tested, a loop may be **Entry-controlled** or **Exit-controlled** loop. In Entry controlled loop, a condition is tested (pre test) before executing the statements. Whereas in Exit-controlled statements are executed first then condition is tested (post test), which ensures at least on time execution of statements.
 - ❑ As per number of execution, a loop may be **Counter-controlled** or **Sentinel** loop. Counter controlled loops are executed fixed number of times, but sentinel loop may be stopped any time by giving its sentinel value. i.e. number of execution can not be forecasted.
 - ❑ A body of loop contains a block, having statements to be executed.
-

The for .. loop

In simple use, a for.. Loop is Entry-controlled and counter controlled loop having fixed number of iteration.

```
for (initialization exp (s) ; condition ; update exp (s) )  
{ .....  
  .....  
}
```



A blue rounded rectangle labeled "Looping statements" has a line pointing to the body of the for loop (the lines between the curly braces).

```
for (int i=1; i<=10 ; i++ ) {  
    System.out.println (""+i);  
}
```

```
//loop to find even nos. up to 50  
for (int i=0; i<=50 ; i=i+2)  
    System.out.println (""+i);
```

```
//loop to find factorial  
int fact=1,num=5;  
for (int i=1; i<=num ; i++)  
    fact=fact * i;  
System.out.println ("factorial="+fact);
```

```
//loop to get sum of first 10 nos.  
int sum=0;  
for (int i=1; i<=10 ; i++ ) {  
    sum=sum+ i;  
}  
System.out.println (""+sum);
```

Variations in for.. loop

❑ Multiple initialization and update expression

A for.. Loop may contain multiple initialization and/or update expressions separated by (,)

```
for (i=0,sum=0;i<=n; sum++, i++)
```

❑ Optional Expressions

In for loop initialization, condition and update section may be omitted. Note that (;) must be present.

```
for ( ; i<=n ; )
```

❑ Infinite loop

For.. Loop may be endless (infinite) when defined condition is always true or it is omitted.

```
for ( ; ; )
```

❑ Empty loop

for.. Loop may not contain any statement in looping body i.e. Empty loop. If (;) placed after for.. Loop then empty loop is created.

```
for (int i=1; i<=300 ; i++) ;
```

❑ Variable declared inside for.. Loop can't accessed outside the loop. Since it is out of scope.

The while.. loop

In simple use, a while .. Loop is Entry-controlled and counter controlled or sentinel looping statement, which executes statements until defined condition is true.

```
while (condition)
```

```
{ .....
```

```
.....
```

```
}
```



Looping statements

```
int i=1;
while ( i<=10) {
    i=i+1;
    System.out.println (" "+i);
}
```

```
//while loop to find factorial
```

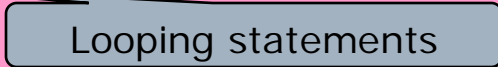
```
int fact=1,num=5,i=1;
while (i<=num)
{ fact=fact * i;
  i++;
}
System.out.println ("factorial="+fact);
```

A while loop also may be empty or infinite

The do..while loop

Unlike for.. and while.. Loop, do..while loop is Exit-controlled and counter controlled or sentinel looping statement, which executes statements until defined condition is true. It always executes at least once.

```
do
{ .....
  .....
} while (condition);
```



```
// do.. Loop to print A – Z letters
char ch ='A';
do {
    System.out.println (" "+i);
    ch++;
} while (ch<='Z');
```

```
//do.. loop fixed time
int i=1;
do
{ System.out.println (" "+i);
  i++;
} while (i<=10);
```

A do...while loop also may be empty or infinite

Which loop is better ?


- ❑ Java offers three looping statements i.e. for.., while.. and do..while. There are some situation where one loop is more appropriate than others.
 - ❑ The for loop is best suited where number of execution is known in advance. (fixed execution)
 - ❑ The while and do.. are more suitable in the situations where it is not known that when loop will terminate. (unpredictable times of execution).
 - ❑ The do.. Loop ensures at least one time of execution since it is Exit-controlled loop.
-

Jump statements in Java


- ❑ Java offers three jump statements (return, break and continue), which transfers the control else where unconditionally.
 - ❑ The **return** statement can be used any where in the program. It transfers the control to calling module or Operating System. However Java provides System.exit() method to stop the execution of program.
 - ❑ The **break** is used with for.., while, do.. and switch statements which enables the program to skip over some part of the code and places control just after the nearest closing of block. It is used to terminate the loop.
 - ❑ The **continue** statement is used within looping statement (not with switch) and works like break i.e. it also skips the statements. Unlike break, it forces the next iteration of loop by skipping the in between code and continues the loop.
-

Break and Continue the loop


```
While (condition 1)
{ statement 1;
  if (condition 2)
    break;
  .....
  statement 2;
}
Statement 3;
```




```
for (ini;cond;update)
{ statement 1;
  if (condition)
    break;
  .....
  statement 2;
}
Statement 3;
```




```
Do
{ statement 1;
  if (condition)
    break;
  .....
  statement 2;
} While (test condition)
Statement 3;
```




```
While (condition 1)
{ statement 1;
  if (condition 2)
    continue;
  .....
  statement 2;
}
Statement 3;
```



```
for (ini;cond;update)
{ statement 1;
  if (condition)
    continue;
  .....
  statement 2;
}
Statement 3;
```



```
Do
{ statement 1;
  if (condition)
    continue;
  .....
  statement 2;
} While (test condition)
Statement 3;
```



Scope of a variable

- ❑ In Java, a variable can be declared anywhere in the program but before using them.
- ❑ The area of program within which a variable is accessible, known as its scope.
- ❑ A variable can be accessed within the block where it is declared.

