

**SOFTWARE
DEVELOPMENT**

SOFTWARE TESTING

IT-IMS

**ENGINEERING
SERVICES**

BFSI

SOFT SKILLS

APPLY

RULE

SIMPLIFY

CONCEPT

EXAMPLE



Copyrights of SEED Infotech Ltd. | Official Curriculum

Copyrights of SEED Infotech Ltd. | Official Curriculum

Client-Server Application Development Using Java

Student Book and Lab Manual

Module Code : [M-ILT-Jav-00064-J2-I-En]

Copyright @ Avani Publications.

Author : Mr. Saket Kale

This edition has been printed and published in house by Avani Publications.

This book including interior design, cover design and icons may not be duplicated/reproduced or transmitted in anyway without the express written consent of the publisher, except in the form of brief excerpts quotations for the purpose of review. The information contained herein is for the personal use of the reader and may not be incorporated in any commercial programs, other books, databases or any kind of software without written consent of the publisher. Making copies of this book or any portion thereof for any purpose other than your own is a violation of copyright laws.

Limits of Liability/Disclaimer of Warranty : The author and publisher have used their best efforts in preparing this book. Avani Publications make no representation or warranties with respect to the accuracy or completeness of the contents of this book, and specifically disclaim any implied warranties of merchantability or fitness for any particular purpose. There are no warranties, which extend beyond the descriptions contained in this paragraph. No warranty may be created or extended by sales representatives or written sales materials. The accuracy and completeness of the information provided herein and the opinions stated herein are not guaranteed or warranted to produce any particular results and the advice and strategies contained herein may not be suitable for every individual. Author or Avani Publications shall not be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential or other damages.

Trademarks : All brand names and product names used in this book are trademarks, registered trademarks or trade names of their respective holders. Avani Publications is not associated with any product or vendor mentioned in this book.

Note: All API references are taken from Java documentation. (<http://docs.oracle.com/javase/7/docs/api/>)

Print Edition : July 2015

Issue No./ Date : 01 / Jan. 12, 2012 **Revision No. & Date :** 01 / July 02, 2012

Avani Publications

15A-1/2/4, Anandmayee Apartment, Off Karve Road, Pune 411004

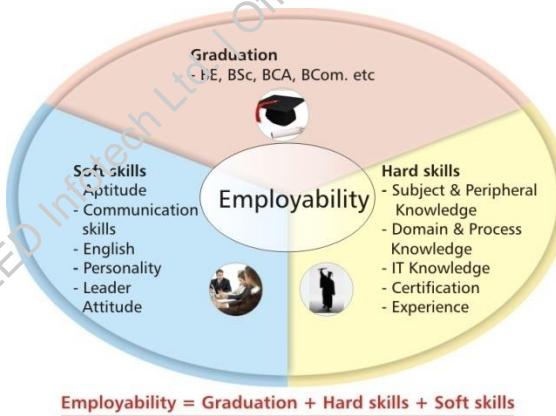
▶ ▶ ▶ **Contents**

Sr. No.	Chapter Name	Page No.
0	Enhancing Employability	iv
1.	Java Swing (Basic Controls)	1
2.	Java Swing(Menu, Dialog, GridBag)	53
3.	Java Swing(Containers, MVC)	100
4.	Java Multithreading Basics	142
5.	Java Thread Communication	176
6.	Java Applets	210
7.	Input-Output(File IO)	233
8.	Java Sockets	278
9.	Java Database Connectivity(JDBC)	314
10.	Appendix A	355
11.	Appendix B	359
12.	Lab Manual	364
13.	Appendix C	395
14.	Appendix D	406

Enhancing Employability

Employability depends on the knowledge, skills and abilities that individuals possess, the way they use these to solve problems and contribute to the growth of the employing organization and the society. Employability skills are those skills that are necessary for getting, keeping and doing well in a job.

Employability can be defined as an equation.



Promise of this book is to help strengthen your “Java programming skill” which can be classified under **Hard Skills** of employability equation.

Why learn ‘Java’ programming?

- a) Java is a simple, platform independent and robust object-oriented programming language.
- b) Java applications are used in various sectors including banking, insurance, retail, media, education, manufacturing etc.
- c) E-commerce, Gaming, Mobile, Embedded, Media and many more types of applications are being developed using Java.
- d) The enterprise applications that use technologies like application servers, business process management (BPM), Portal Servers are either created using Java or can be extended using Java.
- e) Fast growing mobile application development platform Android uses Java.
- f) Many software test automation tools use Java for scripting.
- g) There is continuous demand for good Java professionals in software industry in large numbers.

Role of this book

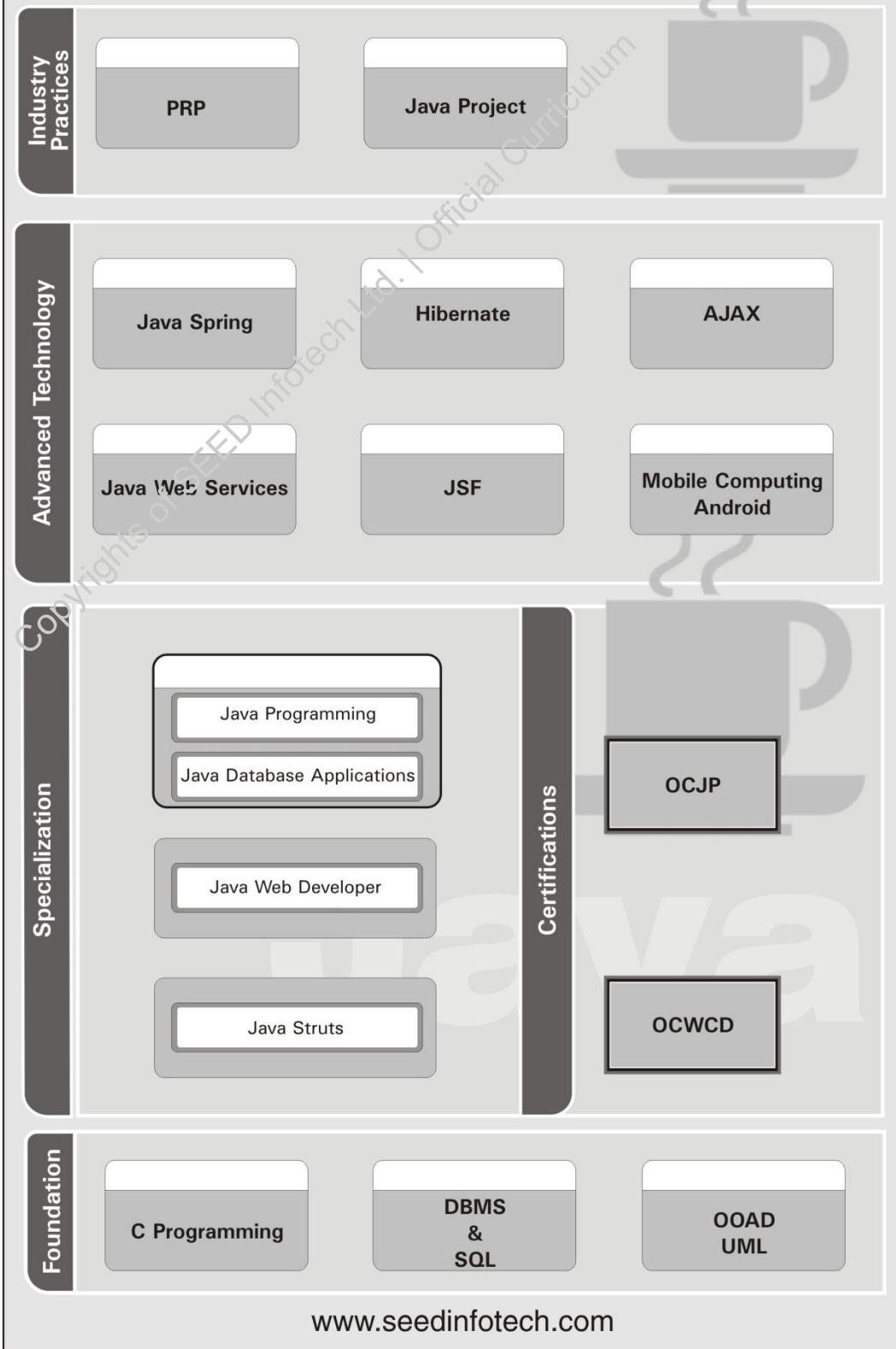
This book is a step by step guide to master Java programming language skills and would aid and reinforce the learning in the classroom. To master any programming language one needs hands-on practice along with clarity of concepts. This student book combined with lab manual is designed to serve the purpose.

The smart tips embedded in the form of Tech App, Interview Tip, Additional Reading, Best Practices, etc. would help increase your curiosity and also help you to become expert with knowledge of peripheral concepts.

We strive hard to make technical contents of this book completely error free. However some mistakes might have slipped our attention. We request the reader to send us any such errors you sight which will help us in improving this book further. Your issues or suggestions are welcome on email at

product-issues@seedinfotech.com

Pathway of Java Expert



“Beyond Obvious” Icons

In the student book, we have included special icons (Beyond Obvious Icons) in the form of footnotes that are interspersed in the study material to give you the precise context of the concept you are learning. As you get accustomed to this way of learning, you will enjoy the fun of it which will make this learning highly productive!



This icon indicates a particular best practice which is followed while developing the applications, in design, in coding etc. Knowledge of best practices makes one a good developer or designer.



This icon indicates the points which are important from your technical interview. Before interview, you may visit these small tips as quick revision pointers.



This icon indicates the features which are new and available in new version JDK 7.



This icon suggests that it is good for the student to go through this additional reading material to bring more clarity to the concepts.



This icon indicates that this is a group discussion or group exercise. This is added for collaborative learning and problem solving. In the job environment one needs to take part in such discussions to arrive at the solution.



This icon indicates that more details are provided with respect to application of the technology/tool/concept which you are learning. This is application of technology learned to solve the real world problem.



This icon indicates a important note or tip from the application design perspective.



This icon indicates quiz to be solved in the classroom. This is for reinforcement of what you have learnt by challenging you through the questions.

Chapter - 1

Java Swing (Basic Controls)



This chapter covers GUI applications using AWT and Swings API. It includes classes like Frame, Window, and Panel. It also contains basic components like Button, TextField etc., respective events of these components, Layouts, LayoutManager.

Objectives

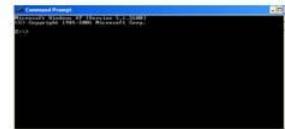
After completion of this chapter you will be able to:

- State what is graphical user interface (GUI).
- Define event driven programming.
- List Java related and other technologies commonly used for GUI creation.
- Name components and libraries in Java Swing and state their application.
- State advantages of Java Swing over old APIs like AWT.
- List steps for building a GUI application using Java Swing.
- Construct an application UI using basic controls like button, text field, combo box, radio button, password field, check box, list etc.

- Construct a UI using containers like JFrame, Panel, and Window etc.
- Develop an application to perform simple event handling like button click.
- Identify events, associated listeners and callback methods.
- Arrange the components using simple layouts like Border, Grid and Flow layout.
- Develop an interactive GUI based application using event handling mechanism.

Why Graphical User Interface (GUI) Programming

- Command-Line
 - Text input
 - Cumbersome to use
- Graphical User Interface
 - Interesting
 - More user-friendly
 - Easy to learn
 - More intuitive



User interface is a part of the application or boundary where consider “human interact with computer.

Command-Line Interface

There is a simple user interface like a command prompt which is used in DOS or UNIX. In this type of interface, the user is prompted to type an instruction or command on the screen through the keyboard. This is cumbersome, since the instruction has to be accurate – there should be no spelling mistakes or syntax errors. So it is necessary for the user to be well acquainted with the instructions that are to be given. Neither is it interesting nor user friendly. Everything is black and white.

Graphical User Interface

A graphical user interface is easier to learn, interesting to use and pleasant to the eye. In this type of interface, the actions to be performed are denoted by using small colorful graphics or pictures. That is why it is called a graphical user interface. Here, the focus is on user actions. The user can interact by using the

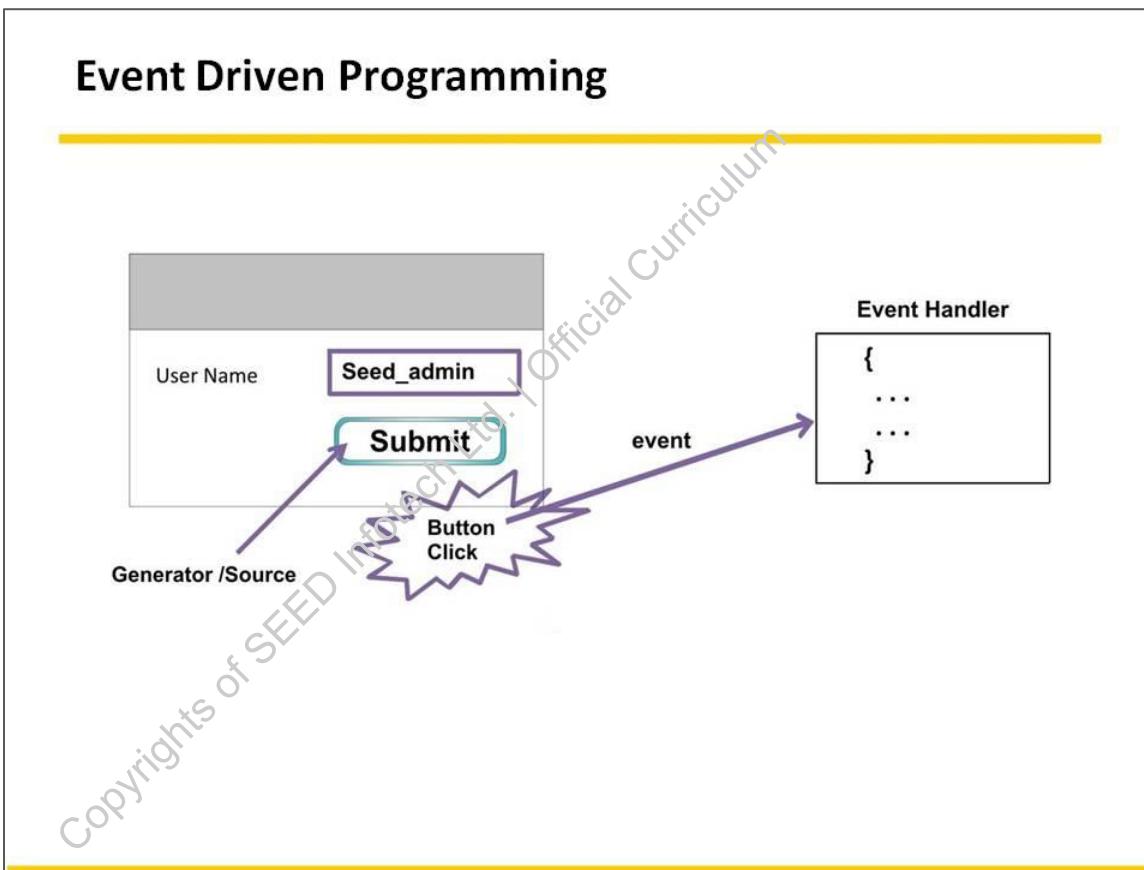
mouse to select the action to be performed by clicking on a particular graphic. So, it gives him a feeling of controlling the application and also responsiveness. For obvious reasons such interface is more user-friendly (technically speaking for high usability). The user need not remember any command syntax and there is no chance of making a spelling mistake. For example, if the user wants to print a file, all he needs to do is to click on a small graphic depicting a printer.



Tech App

There is an emerging discipline called usability engineering which focuses on science of how to make the user interfaces more user friendly. Software Testing also contains a specialized testing called usability testing. For more information on this discipline refer

http://en.wikipedia.org/wiki/Usability_testing



The normal sequential execution flow of an application is good enough for applications that are straight-forward and which follow a pre-defined execution sequence. These are typically called as batch jobs. Applications which rely on user interaction for execution may not be easily created using the classical sequential programming model. GUI needs a different kind of programming called as Event Driven Programming.

An event is any action or incident of user's interest. Based on the event, user expects a response from the system. For example, when the door bell button is pressed, a bell rings and somebody comes to open the door. Here, the ringing of the bell is an event and response from the system is opening the door.

Similarly, the GUI consists of various components like a button, a mouse pointer and the user interacts by clicking a button, moving a mouse, and so on. Such activity performed by the user is an event and the component is called event generator or event source. User expects that system should respond to such event. This response needs to be pre-programmed in the system and is called event

handler. Business logic methods are invoked from these handlers. When one wants to learn event-driven programming one should get accustomed to this terminology and should be able to create small code snippets as part of event handlers. This is very different from the sequential programming which is known to all.

GUI Frameworks

- Java Frameworks
 - AWT/Swing
 - JavaFX
 - Flex
- Non-Java Frameworks
 - .NET WinForms
 - Visual Basic
 - Oracle Forms



To build graphical user interface, rich set of components and container and event handling mechanisms are required. There are readymade sets of such libraries available in various languages and platforms. These are called GUI frameworks. These frameworks help one build the rich graphical user interface applications easily and faster. Following is a set of some popular Java and non-Java GUI frameworks.



Additional Reading

AWT / Swing: Abstract Window Toolkit (AWT)	<p>AWT / Swing: Abstract Window Toolkit (AWT) or Swing is libraries provided by Java for building GUI Application using URL http://download.oracle.com/javase/tutorial/uiswing/</p>
JavaFX	<p>A set of libraries provided by OpenSource community for building Rich Internet Applications using Java URL: http://download.oracle.com/javafx/1.3/tutorials/core/</p>
Flex	<p>A set of libraries used to integrate Adobe Flash as a front-end (GUI) with Java as a backend (Handler) URL : http://www.adobe.com/products/flex.html</p>
.NET Winforms	<p>A set of generic APIs used by .NET languages (C#, VB.NET) to build Rich UI application in .NET domain URL : http://www.dotnetspider.com/tutorials/DotNet-Tutorial-275.aspx</p>
Visual Basic	<p>A programming language which helps a developer build enterprise applications using Event Driven Mechanism URL : http://www.vbtutor.net/vb2010/index.html</p>
Oracle Forms	<p>URL : http://www.slideshare.net/magupta26/oracle-forms-tutorial</p>

Why Swing?

- Removes platform dependency prevalent in AWT
 - No need for platform specific peer classes
- Lightweight in comparison to AWT
- Eases programming efforts
 - undo – redo support built-in for text components
 - 2D graphics rendering simpler
 - Pluggable (customizable) look and feel
- Swing components implement Model View Control design pattern for flexible UI rendering

Java Swing is not a replacement for Abstract Window Toolkit (AWT). It is an enhancement over the existing APIs of AWT. AWT and Swing are similar, with most of AWT components having corresponding components in Swing. It is recommended though that AWT should be restricted only to the event handling mechanism and the UI component creation and rendering should be done only using Swing.

Following are the compelling reasons to use Java Swing:

- Rich set of components easing the developer effort
 - undo - redo support built-in for text components
 - 2D graphics rendering simpler
 - Pluggable (customizable) look and feel
 - Model-View-Controller(MVC) design pattern for flexible UI rendering
- Few dependencies on the underlying platform which is prevalent in AWT
- Consistent user experience across various platforms

Java Swing Packages

Following packages are useful for developing rich user interfaces in Java using Swing APIs.

Package name	Description
javax.swing	Provides a set of "lightweight" (written in Java with no native code) components that, to the maximum degree possible, work the same on all platforms.
javax.swing.event	Provides support for events fired by Swing components.
javax.swing.plaf	Provides one interface and many abstract classes that Swing uses to provide its pluggable look and feel capabilities.
javax.swing.table	Provides classes and interfaces for dealing with JTable.
javax.swing.text	Provides classes and interfaces that deal with editable and non-editable text components.
javax.swing.tree	Provides classes and interfaces for dealing with JTree.
javax.swing.undo	Allows developers to provide support for undo/redo in applications such as text editors.
java.awt	Provides the base toolkit classes for interacting with the OS also provides classes for rendering 2d graphics.
java.awt.event	Provides bases classes for event delegation model.

java.awt.Graphics2D	This Graphics2D class extends the Graphics class to provide more sophisticated control over geometry, coordinate transformations, color management, and text layout. This is the fundamental class for rendering 2-dimensional shapes, text and images on the Java(tm) platform.
java.awt.graphics	Provides the basis for co-ordinate system for building canvas.

Building a GUI application

1. Identify user requirements for GUI
2. Map the requirements to Swing components(wire frames)
3. Group UI components
4. Define UI structure
5. Select suitable events to handle
6. Link events to execution logic

The process of development of graphical user interface is a part of making software.

Problem Statement

User should enter his username and password on the form. Password should be validated and the message should be shown to the user accordingly.

Step 1: Requirements Analysis

- User needs to input two string-based values.
- User will need to be notified about what he/she is entering.
- User will need to signal data entry complete.
- User may need to clear the data-entry.
- User may need to cancel the application.



Here a very simplified approach is taken for brevity. In a real software development scenario a technique called Use Case modeling is used.

http://en.wikipedia.org/wiki/Use_case_diagram

Step 2: Component Selection

Creation of the UI mockups or wireframes is done based on the input from the first step.

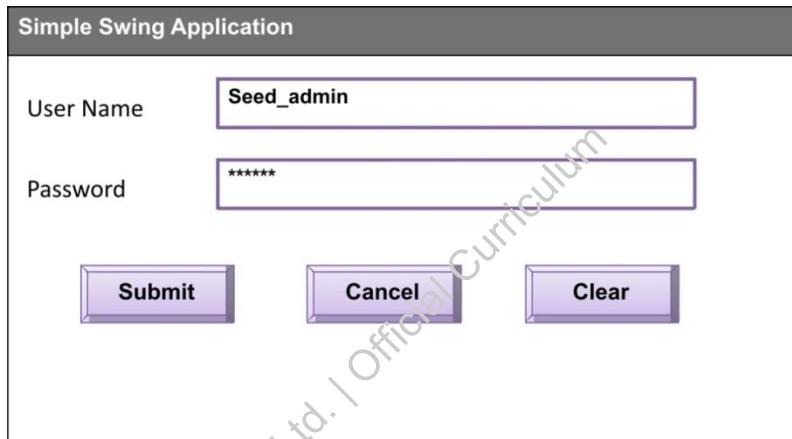
Requirement	Information	Component
Data Entry	UserName	JTextField
Data Entry	Password	JPasswordField
Information Rendering	Captions	JLabel
Performing Action	Submit, Cancel, Clear	JButton
Group Components	Container	JFrame, JPanel
Event Detector	Listener	ActionListener
Event Handler	Bridging Code	actionPerformed
UI Structure	Layout	FlowLayout

Step 3: Container Selection

A container needs to be chosen to hold the components together; in this case JFrame (details will be explored later) will be used.

Step 4: Applying Layout

To ensure that the User Interface is readable and not displayed haphazardly, a specific layout needs to be utilized. As the application is a simple one without many components simple layout strategy as provided by FlowLayout can be used.



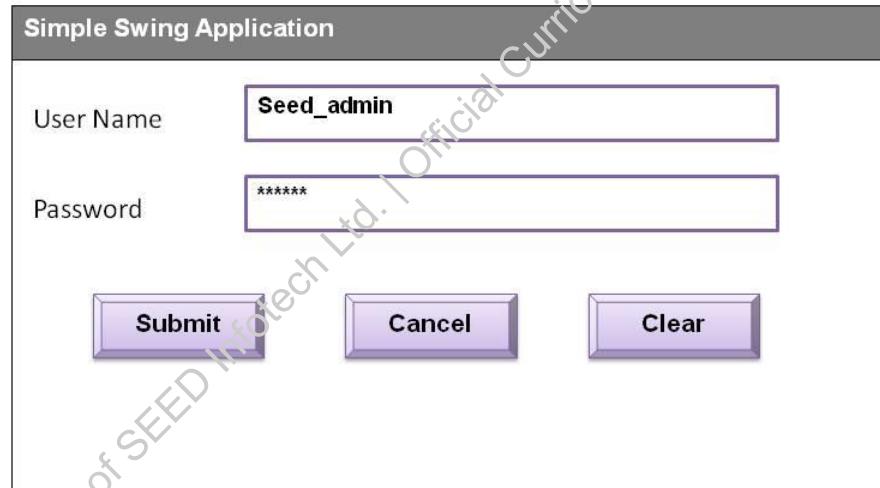
Step 5: Event Identification

The business logic needs to be triggered when the button is clicked, hence out of all the events available, the event corresponding to the button click needs to be trapped. Clicking on a button is equivalent to taking an action. In Swing it is mapped to a listener called as `ActionListener` which is responsible for identifying the source of the event and receiving the metadata associated with the event.

Step 6: Event Handling

`ActionListener` has a method called as `actionPerformed` which facilitates linking of the event source with the corresponding business logic.

A Swing Application



```
package com.seed.demos;
//import statements
public class SwingApp {
    public static void main(String[] args) { Container
        final JFrame frame = new JFrame
("JPasswordField Demo");
        JLabel lblUser = new JLabel("User Name:");
        final JTextField tfUser = new JTextField (20);
        lblUser.setLabelFor (tfUser);
        JLabel lblPassword = new JLabel ("Password :");
        final JPasswordField pfPassword = new
JPasswordField (20);
        lblPassword.setLabelFor (pfPassword);
        JButton btnCancel = new JButton ("Cancel");
        JButton btnClear = new JButton ("Clear");
        JButton btnLogin = new JButton ("Submit")
```

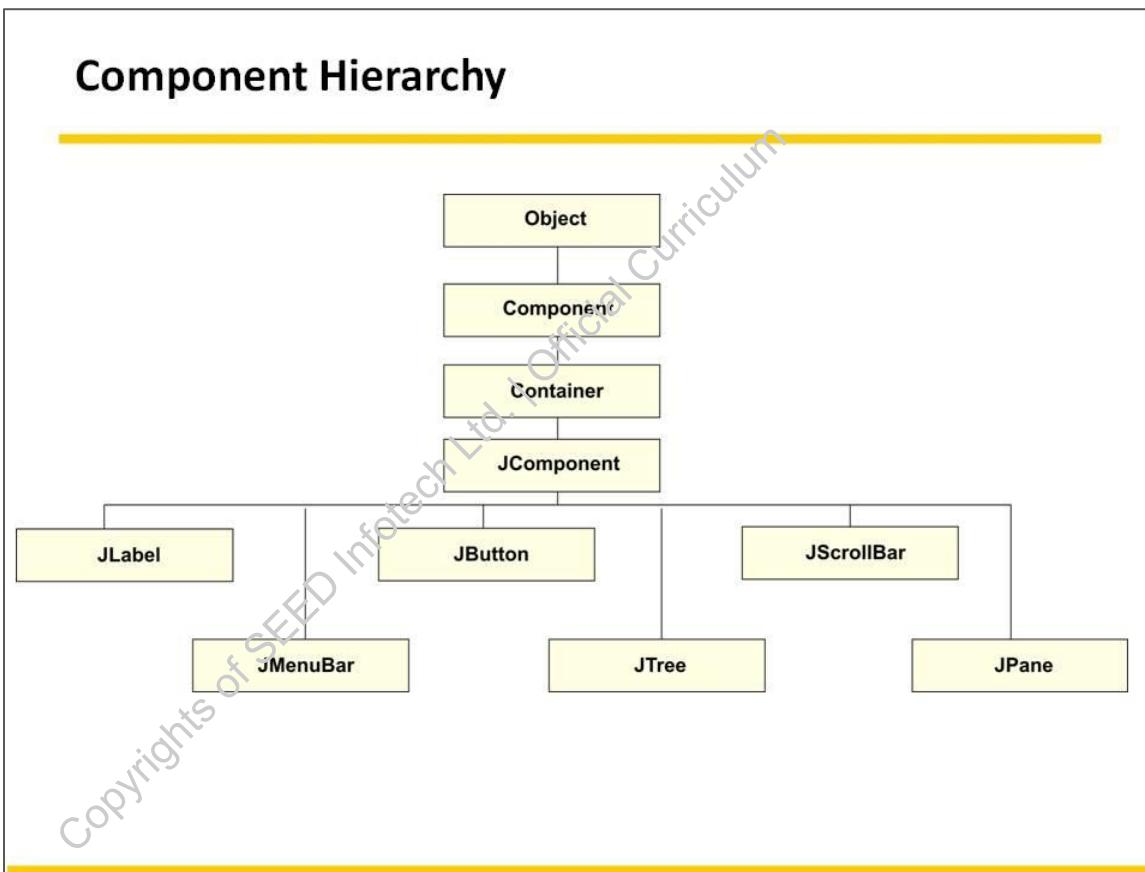
Connect
component to
event handler

```

        btnCancel.addActionListener (new
PerformActions());
        btnLogin.addActionListener (new
PerformActions());
        btnCancel.addActionListener (new
ActionListener() {
            public void actionPerformed (ActionEvent
arg0)
            {
                tfUser.setText ("");
                pfPassword.setText ("");
            }
        });
        JPanel panel = new JPanel (),
panel.setLayout (new FlowLayout ());
panel.add (lblUser);
panel.add(tfUser);
panel.add (lblPassword);
panel.add (pfPassword);
panel.add (btnLogin);
panel.add (btnCancel);
panel.add (btnClear);
frame.setDefaultCloseOperation
(JFrame.EXIT_ON_CLOSE);
frame.setSize (500, 300);
frame.getContentPane ().add (panel);
frame.setVisible (true);
}
}
class PerformActions implements ActionListener
{
    public void actionPerformed (ActionEvent arg0) {
        String caption=arg0.getActionCommand ();
        if (caption.equals ("Submit"))
        {
            System.out.println ("Will be performing
database operations later");
        }
    }
}

```

```
    }
    else if (caption.equals ("Cancel"))
    {
        System.out.println ("Will need to stop
execution and return to base");
    }
}
```



The basic requirement from a GUI is to allow the user to interact with the system. In simpler words, the system should allow the user to provide data to the system and get output from the system

For doing this kind of unglamorous work, a set of simple display components are available. Each of these components is designed to perform simple tasks that are frequently required by the user.

JButton



Following are some of the important methods of JButton class:

Method	Description
JButton (Action) JButton (String, Icon)	Creates a JButton instance, initializing it to have the

<code>JButton(String) JButton(Icon) JButton()</code>	specified text/image/action.
<code>void setText(String) String getText()</code>	Sets or gets the text displayed by the button.
<code>void setIcon(Icon) Icon getIcon()</code>	Sets or gets the image displayed by the button when the button isnot selected or pressed.
<code>void setHorizontalAlignment(int) void setVerticalAlignment(int) int getHorizontalAlignment() int getVerticalAlignment()</code>	Set or get where in the button its contents should be placed. The AbstractButton class allows any one of the following values for horizontal alignment: RIGHT, LEFT, CENTER (the default), LEADING, and TRAILING. For vertical alignment: TOP, CENTER (the default), and BOTTOM.
<code>void setHorizontalTextPosition(int) void setVerticalTextPosition(int) int getHorizontalTextPosition() int getVerticalTextPosition()</code>	Set or get where the button's text should be placed, relative to the button's image. The AbstractButton class allows any one of the following values for horizontal position: LEFT, CENTER, RIGHT, LEADING, and TRAILING (the default). For vertical position: TOP, CENTER (the default), and BOTTOM.
<code>void setMarginInsets() Insets getMargin()</code>	Sets or get the number of pixels between the button's border and its contents.
<code>void setMnemonic(int)</code>	Set or get the keyboard

char getMnemonic()	alternative to clicking the button. One form of the setMnemonic method accepts a character argument; however, the Swing team recommends that should be an int argument instead, specifying a KeyEvent.VK_X constant.
void addActionListener(ActionListener) ActionListener removeActionListener()	Adds or removes an object that listens for action events fired by the button.
void addItemListener(ItemListener) ItemListener removeItemListener()	Adds or removes an object that listens for item events fired by the button.
void setSelected(boolean) boolean isSelected()	Sets or gets whether the button is selected. Makes sense only for buttons that have on/off state, such as check boxes.
void doClick() void doClick(int)	Programmatically perform a "click". The optional argument specifies the amount of time (in milliseconds) that the button should look pressed.
void setMultiClickThreshhold(long) long getMultiClickThreshhold()	Sets or gets the amount of time (in milliseconds) required between mouse press events for the button to generate corresponding action events.

JTextField

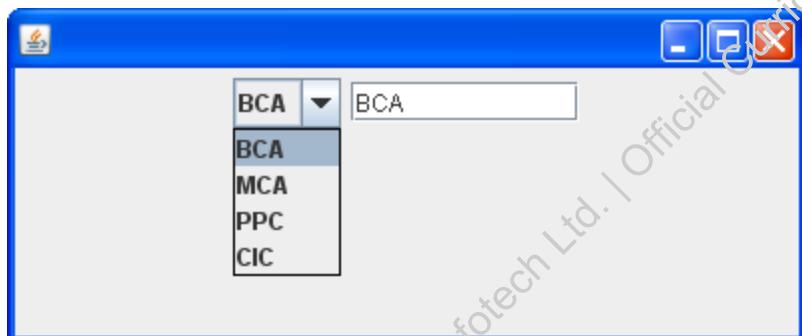
value will be enter here

Following are some of the important methods of JTextField class:

Method	Description
JTextField() JTextField(String) JTextField(String, int) JTextField(int)	Creates a text field. When present, the int argument specifies the desired width in columns. The String argument contains the field's initial text.
Void setText(String) String getText()	Sets or obtains the text displayed by the text field.
void setEditable(boolean) boolean isEditable()	Sets or indicates whether the user can edit the text in the text field.
void setColumns(int); int getColumns()	Sets or obtains the number of columns displayed by the text field. This is really just a hint for computing the preferred width of the field.
void setHorizontalAlignment(int); int getHorizontalAlignment()	Sets or obtains how the text is aligned horizontally within its area. You can use JTextField.LEADING, JTextField.CENTER, and JTextField.TRAILING for arguments.
void addActionListener(ActionListener) void removeActionListener(ActionListener)	Adds or removes an action listener.

r)	
void selectAll()	Selects all characters in the text field.

JComboBox



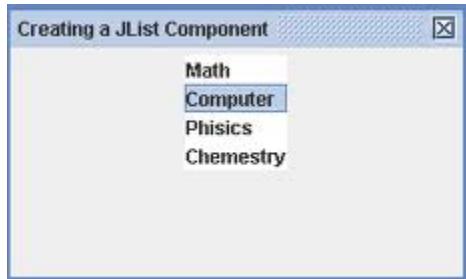
Following are some of the important methods of **JComboBox** class:

Method	Description
<code>JComboBox ()</code> <code>JComboBox (ComboBoxModel)</code> <code>JComboBox (Object [])</code> <code>JComboBox (Vector)</code>	Creates a combo box with the specified items in its menu. A combo box created with the default constructor has no items in the menu initially. Each of the other constructors initializes the menu from its argument: a model object, an array of objects, or a Vector of objects.
<code>void addItem (Object)</code> <code>void insertItemAt (Object, int)</code>	Adds or inserts the specified object into the combo box. The insert method places the specified object at the specified index, thus inserting it before the object currently at that index. These methods require that the combo box's data model be an instance of MutableComboBoxModel.
<code>Object getItemAt (int)</code> <code>Object getSelectedItem ()</code>	Gets an item from the menu of the combo box.

<pre>void removeAllItems() void removeItemAt(int) void removeItem(Objec t)</pre>	<p>Removes one or more items from the menu of the combo box . These methods require that the combo box's data model be an instance of MutableComboBoxModel.</p>
<pre>int getItemCount()</pre>	<p>Gets the number of items in the menu of the combo box.</p>
<pre>void setModel(ComboBox xModel) ComboBoxModel getModel()</pre>	<p>Sets or get the data model that provides the items in the menu of the combo box.</p>
<pre>void setAction(Action) Action getAction()</pre>	<p>Sets or gets the Action associated with the menu of the combo box.</p>
<pre>void addActionListene r(ActionListener)</pre>	<p>Adds an action listener to the combo box. The listener's actionPerformed method is called when the user selects an item from the menu of the combo box or, in an editable combo box, when the user presses Enter.</p>
<pre>void addItemListener(ItemListener)</pre>	<p>Adds an item listener to the combo box. The listener's itemStateChanged method is called when the selection state of any of the combo box's items change.</p>
<pre>void setEditable(bool ean)</pre>	<p>Sets or gets whether the user can type in the combo box.</p>

boolean isEditable()	
-------------------------	--

JList



Following are some of the important methods of `JList` class:

Method	Description
<code>JList(ListModel)</code> <code>JList(Object[])</code> <code>JList(Vector)</code> <code>JList()</code>	Creates a list with the initial list items specified. The second and third constructors implicitly create an immutable <code>ListModel</code> ; should not be modified subsequently the passed-in array or <code>Vector</code> .
<code>void setModel(ListModel)</code> <code>ListModel getModel()</code>	Sets or gets the model that contains the contents of the list.
<code>void setListData(Object[])</code> <code>void setListData(Vector)</code>	Sets the items in the list. These methods implicitly create an immutable <code>ListModel</code> .
<code>void setVisibleRowCount(int)</code>	Sets or gets the <code>visibleRowCount</code> property. For a VERTICAL layout orientation, this sets or gets the preferred number of rows

<pre>int getVisibleRowCount()</pre>	<p>to display without requiring scrolling. For the HORIZONTAL_WRAP or VERTICAL_WRAP layout orientations, it defines how the cells wrap.</p>
<pre>void setLayoutOrientation(int) int getLayoutOrientation()</pre>	<p>Sets or gets the way list cells are laid out. The possible layout formats are specified by the JList-defined values VERTICAL (a single column of cells; the default), HORIZONTAL_WRAP ("newspaper" style with the content flowing horizontally then vertically), and VERTICAL_WRAP ("newspaper" style with the content flowing vertically then horizontally).</p>
<pre>int getFirstVisibleIndex() int getLastVisibleIndex()</pre>	<p>Gets the index of the first or last visible item.</p>
<pre>int ensureIndexIsVisible(int)</pre>	<p>Scrolls so that the specified index is visible within the viewport that this list is in.</p>
<pre>void addListSelectionListener(ListSelectionListener)</pre>	<p>Registers to receive notification of selection changes.</p>
<pre>Void setSelectedIndex(int) void setSelectedIndices(int[])</pre>	<p>Sets the current selection as indicated. Uses setSelectionMode to set what ranges of selections are acceptable. The boolean argument specifies whether the list should attempt to scroll itself so that the selected item is visible.</p>

<pre>void setSelectedValue (Object, boolean) void setSelectionInte rval(int, int)</pre>	
<pre>int getAnchorSelecti onIndex() int getLeadSelection Index() int getSelectedIndex () int getMinSelectionI ndex() int getMaxSelectionI ndex() int[] getSelectedIndic es() Object getSelectedValue () Object[] getSelectedValue s()</pre>	<p>Gets information about the current selection as indicated.</p>
<pre>void setSelectionMode (int) int getSelectionMode</pre>	<p>Sets or gets the selection mode. Acceptable values are: SINGLE_SELECTION, SINGLE_INTERVAL_SELECTION, or MULTIPLE_INTERVAL_SELECTION (the default),</p>

()	which are defined in ListSelectionModel.
void clearSelection() boolean isSelectionEmpty() ()	Sets or gets whether any items are selected.
boolean isSelectedIndex(int)	Determines whether the specified index is selected.
int getNextMatch(String, int, javax.swing.text. .Position.Bias)	Given the starting index, searches through the list for an item that starts with the specified string and returns that index (or -1 if the string is not found). The third argument, which specifies the search direction, can be either Position.Bias.Forward or Position.Bias.Backward.
void setDragEnabled(boolean) boolean getDragEnabled()	Sets or gets the property that determines whether automatic drag handling is enabled.

JCheckBox



Following are some of the important methods of JCheckBox class:

Method	Description
JCheckBox (Action)	Create a JCheckBox instance. The string argument specifies the text, if
JCheckBox (String)	

```
JCheckBox(String,boolean)
JCheckBox(Icon)
JCheckBox(Icon,boolean)
JCheckBox(String,Icon)
JCheckBox(String,Icon,boolean)
JCheckBox()
```

any, that the check box should display. Similarly, the Icon argument specifies the image that should be used instead of the default check box image. Specifying the boolean argument as true initializes the check box to be selected. If the boolean argument is absent or false, then the check box is initially unselected.

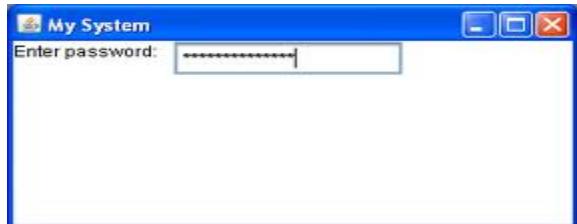
JRadioButton



Following are some of the important methods of JRadioButton class:

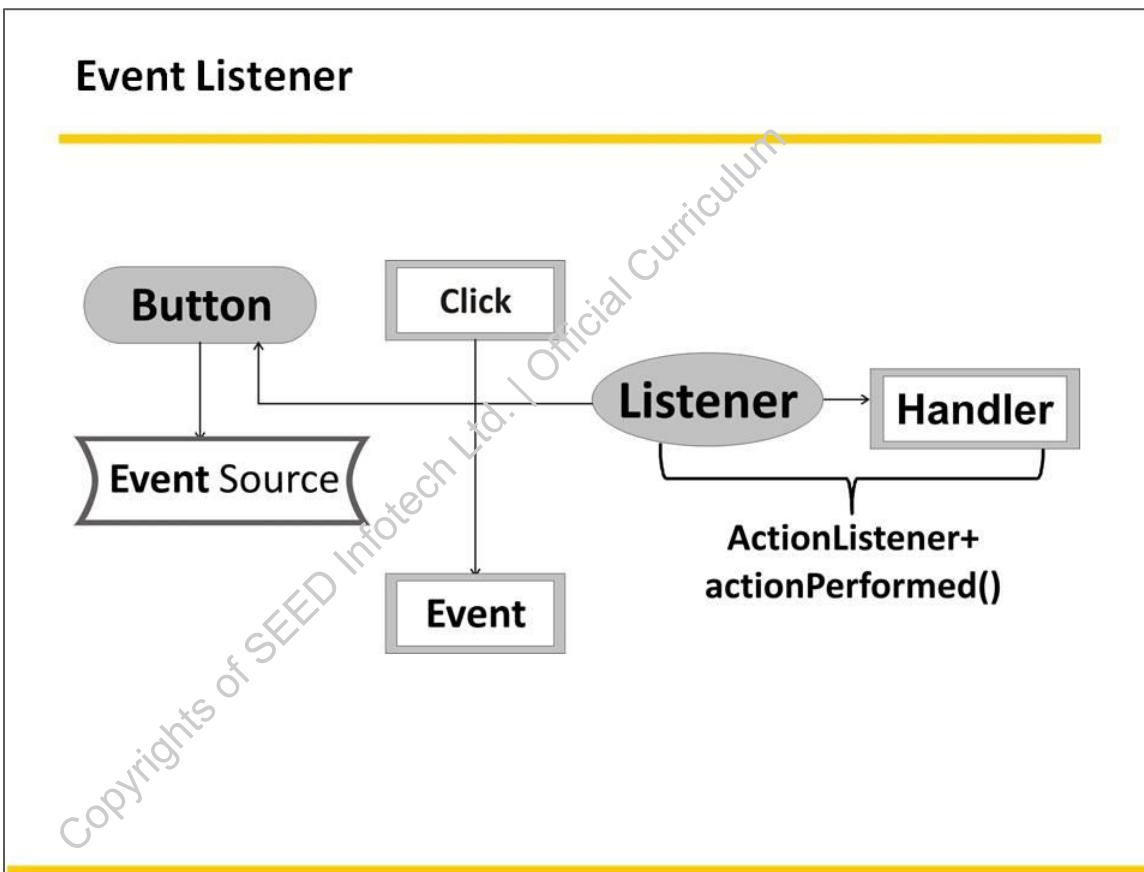
Method	Description
<pre>JRadioButton(Action)) JRadioButton(String)) JRadioButton(String, ,boolean) JRadioButton(Icon) JRadioButton(Icon, boolean) JRadioButton(String, ,Icon) JRadioButton(String, ,Icon, boolean) JRadioButton()</pre>	<p>Create a JRadioButton instance. The string argument specifies the text, if any, that the radio button should display. Similarly, the Icon argument specifies the image that should be used instead of the look and feel's default radio button image. Specifying the boolean argument as true initializes the radio button to be selected, subject to the approval of the ButtonGroup object. If the boolean argument is absent or false, then the radio button is initially unselected.</p>

JPasswordField



Following are some of the important methods of JPasswordField class:

Method	Description
<code>JPasswordField()</code> <code>JPasswordField(String)</code> <code>JPasswordField(String, int)</code> <code>JPasswordField(int)</code> <code>JPasswordField(Document, String, int)</code>	Creates a password field. When present, the int argument specifies the desired width in columns. The String argument contains the field's initial text. The Document argument provides a custom model for the field.
<code>char[] getPassword()</code>	Returns the password as an array of characters.
<code>void addActionListener(ActionListener)</code> <code>void removeActionListener(ActionListener)</code>	Adds or removes an action listener.
<code>void selectAll()</code>	Selects all characters in the password field.



Code can be written to respond to the events generated by the components. When user does some action with the component like button click, an event is generated.

This event is passed to the event handler which has the logic to respond to the event. In physical terms, a component (say Button) and a method which would respond to the button click (say validation) have been created. It has to be checked how to connect these two pieces together.

Summarizing the entire process of event delegation model in Swing:

1. There is a component on the screen (GUI), which receives the user's attention.
2. User performs some action (Event) on it.
3. The system encapsulates the meta-data associated with the component (e.g., the name/label of the component) and generates an object called as an Event (which contains the metadata mentioned). This data in turn is passed on to the listener which has been registered with the component.

4. The listener delegates the data to the event handler for further processing.
5. The handler picks up the necessary inputs from the data passed to it and invokes the business logic execution by passing it.

When button is clicked, it creates and ActionEvent object and invokes actionPerformed (ActionEvent) method of ActionListener interface. Now, all that is needed is to write code inside the actionPerformed method by implementing the ActionListener interface. Event source needs a reference to the object of event handler so that it can call its method.

Event Listeners

Following table lists the various event listeners.

Listener	Purpose	Example
ActionListener	Listens for a component specific event	Button click
ChangeListener	Listens for state change of component	Data entry in a text field
MouseListener	Listens for Mouse click, release	Mouse click
MouseMotionListener	Listens for mouse movement	Mouse moving on the screen
KeyListener	Listens for keyboard activity	Key press with keyboard
WindowListener	Listens for Window handling	Window maximize, minimize, close
ItemListener	Listens for state change of an item	Checkbox on/off, List item selection
AdjustmentListener	Listens for scrollbar movement	Scrollbar being dragged

WindowFocusListener	Listens for window activation	Making a window active
---------------------	-------------------------------	------------------------

Ready-reckoner for Events and Listeners

Event	Listener	Action/Method	Component
Click	ActionListener	actionPerformed	Button, CheckBox, RadioButton, TextField, PasswordField
DoubleClick	ActionListener	actionPerformed	List
Keyboard	KeyListener	keyTyped keyPressed keyReleased	Any UI Component
Focus	FocusListener	focusGained focusLost	Any UI Component
Mouse Button Press Mouse Button Release	MouseListener	mousePressed mouseReleased	Any UI Component/Container
Mouse moved Mouse Click+Move	MouseMotion Listener	mouseMoved mouseDragged	Any UI Component/Container
minimized	WindowLister	windowIconified	Frame, Dialog

maximized	ner	ied windowDeiconified	
Activated Deactivated	WindowListener	windowActivated windowDeactivated	Frame, Dialog
Item selection	ItemListener	itemStateChanged	List, CheckBox, RadioButton, ComboBox
state changed	ChangeListener	stateChanged	TextField, PasswordField

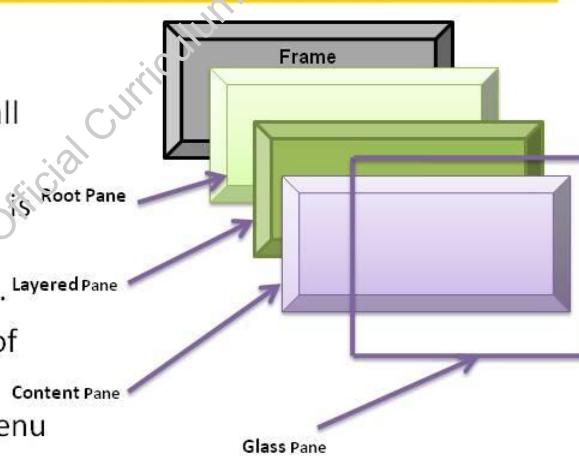
The table displayed above should be used as a ready-reckoner for identifying the event with user action. The first column identifies the user action on the screen, the second column identifies the listener associated with that event, third column specifies the method that needs to be implemented, and the last column maps the component with the event of the first column.

For example, a user types some data in a textbox and presses enter key:

1. Identify the component from last column
2. Identify the action performed
3. Identify the listener and the method to be implemented.

Basic Containers

- Root Pane: A lightweight container which is on top of all other containers.
- Layered Pane: A layered pane is of extreme importance when there are multiple dimensions.
- Content Pane: The container of the root pane's visible components, excluding the menu bar.
- Glass Pane: The glass pane is like a sheet of glass put on top of the actual content of the window.



Containers are responsible for keeping the components together as a single group. This would facilitate management of all the components together as a group for resizing, repositioning and creating a meaningful user interface with collection of components. Some of the basic containers are discussed below.

A root pane is always available, regardless of whether it is needed or not. As soon as a window in any form is created a root pane is also created. It is actually made up of four distinct parts.

Glass Pane

The glass pane is like a sheet of glass put on top of the actual content of the window. Normally a glass pane is hidden, but it can be made opaque by rendering by using the `paintComponent` method. As it lies on top of all the layers in the display, it can act as an interceptor for any input event generated on the screen.

Layered Pane

A layered pane is of utmost importance when multiple dimensions are needed to place the component, to give an illusion of depth. Content Pane is one such default instance of a layered pane.

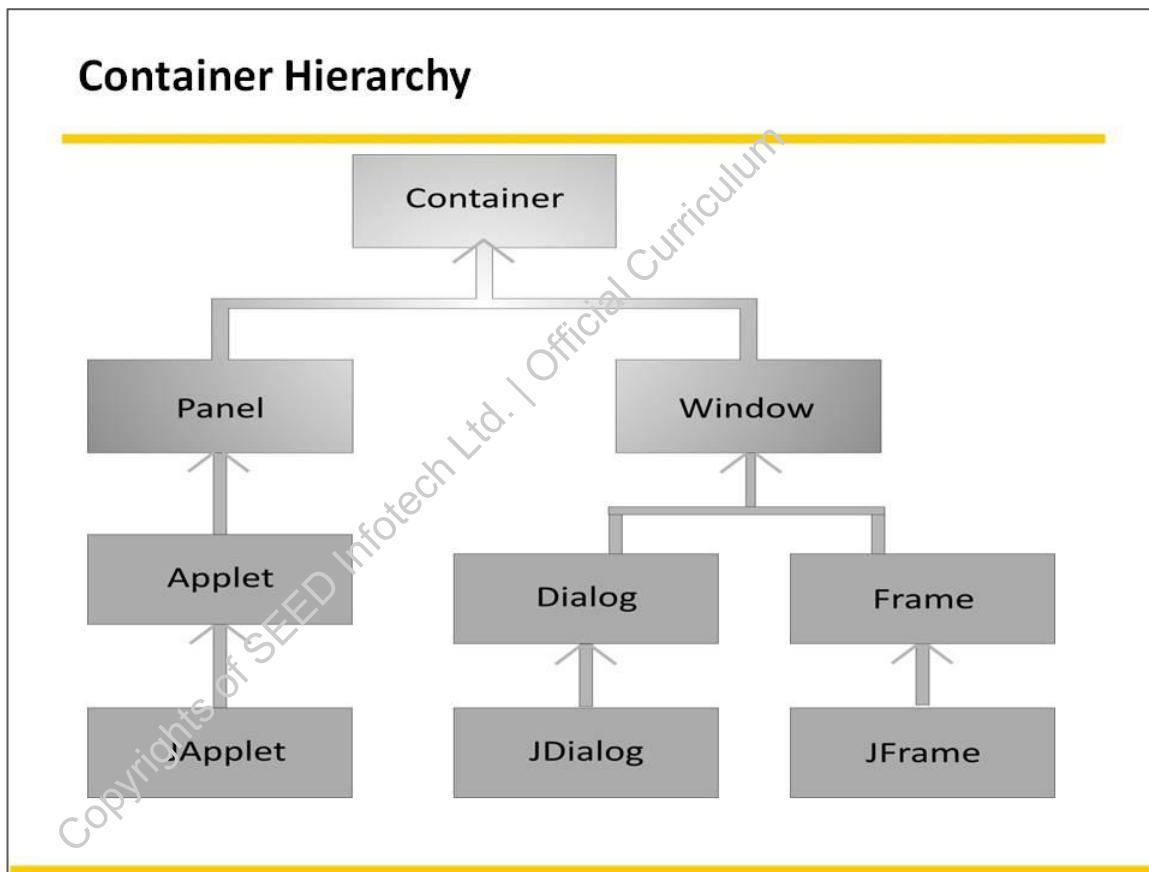
Content Pane

It is the container of the visible components of the rootpane, excluding the menu bar.

Root pane

A JRootpane is made up of a glassPane, an optional menuBar, and a contentPane. (The JLayeredPane manages the menuBar and the contentPane.) The glassPane is placed on top of everything.

The Root pane manages the content pane and the menu bar, along with a couple of other containers.



Container: the Parent

An AWT container object is a component that can contain other AWT components. Components added to a container are stacked in a list. The order of the list defines the components' front-to-back stacking order within the container. If no index is specified when adding a component to a container, it will be added to the end of the list (and hence to the bottom of the stacking order).

Container class cannot be initialized directly. To initialize it, its subclasses, such as Panel, Frame, or Dialog are used. Once a container is created, its LayoutManager can be set with `setLayout()`, and components can be added to it with `add()`, and removed from them with `remove()`. `getComponents()` returns an array of the components contained in a container.

Some of the important methods of Container class are as follows:

Method	Description
void addImpl(Component comp, Object constraints, int index)	Adds the specified component to the container at the specified index.
void setLayout(LayoutManager mgr)	Sets the layout manager for the container.
Component add(Component comp)	Appends the specified component to the end of the container.
Component[] getComponents();	Gets all the components in the container
void applyComponentOrientation(ComponentOrientation o)	Sets the ComponentOrientation property of this container and all components contained within it.
LayoutManager getLayout()	Gets the layout manager for the container.
int remove(int index)	Removes the component, specified by index, from the container.
void validate()	Validates the container and all of its subcomponents.

JFrame

A Frame is a top-level container with associated meta-data like, a title and a border. The size of the frame includes any area designated for the border. The border size is included in the overall size of the frame. This results in reducing the rendering area for the components by the size of the border.

A frame, implemented as an instance of the `JFrame` class, is a window that has decorations such as a border, a title, and supports button components that close or minimize/restore the window. Applications with a GUI usually work with at least one frame.

The impact of the frame can be further enhanced by using built-in decorations available. *The implementation of providing decorations will be taken up later in the discussion.* The developer can also associate an image as an icon for the window.

JFrame API

Following are some of the important methods of `JFrame`:

Method	Description
<code>JFrame ()</code> <code>JFrame (String)</code>	Creates a frame that is initially invisible. The <code>String</code> argument provides a title for the frame. To make the frame visible, invoke <code>setVisible (true)</code> on it.
<code>void setDefaultCloseOperation (int)</code> <code>int getDefaultCloseOperation ()</code>	Sets or gets the operation that occurs when the user pushes the close button on the frame. Possible choices are: <code>DO NOTHING ON CLOSE</code> <code>HIDE ON CLOSE</code> <code>DISPOSE ON CLOSE</code> <code>EXIT ON CLOSE</code> The first three constants are defined in the <code>WindowConstants</code> interface, which <code>JFrame</code> implements. The <code>EXIT_ON_CLOSE</code> constant is defined in the <code>JFrame</code> class.
<code>void setIconImage (Image)</code> <code>Image getIconImage ()</code>	Sets or gets the icon that represents the frame. Note that the argument is a <code>java.awt.Image</code> object, not a <code>javax.swing.ImageIcon</code> (or any other

	<code>javax.swing.Icon implementation).</code>
<code>void setTitle(String) String getTitle()</code>	Sets or gets the frame title.
<code>void setUndecorated(boolean) boolean isUndecorated()</code>	Set or get whether the frame should be decorated. Works only if the frame is not yet displayable (has not been packed or shown). Typically used with full-screen exclusive mode or to enable custom window decorations.
<code>static void setDefaultLookAndFeelDecorated(boolean) static boolean isDefaultLookAndFeelDecorated()</code>	Determine whether subsequently created JFrames should have their Window decorations (such as borders, and widgets for closing the window) provided by the current look-and-feel. Note that this is only a hint, as some look and feels may not support this feature.
<code>void setSize(int, int) void setSize(Dimension) Dimension getSize()</code>	Sets or gets the total size of the window. The integer arguments to <code>setSize</code> specify the width and height, respectively.
<code>void setBounds(int, int, int, int) void setBounds(Rectangle) Rectangle getBounds()</code>	Sets or gets the size and position of the window. For the integer version of <code>setBounds</code> , the window's upper left corner is at the x, y location specified by the first two arguments, and has the width and height specified by the last two arguments.
<code>void</code>	Sets or gets the location of the upper left

setLocation(int, int) Point getLocation())	corner of the window. The parameters are the x and y values, respectively.
void setLocationRelativeTo Component)	Positions the window so that it is centered over the specified component. If the argument is null, the window is centered onscreen. To properly center the window, invoke this method after the window size has been set.
void setLayout (LayoutManager mgr)	Sets the layout manager for the frame.

Code Example

Following code snippet explains the simple methods of a JFrame:

```
package com.seed.project.JFrameProject;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.WindowConstants; Creating instance  
of JFrame

public class SimpleFrame {

    public static void main(String args) Adding label component to JFrame
        JFrame frame = new JFrame("Swing Frame"); //Title
        of the window

        JLabel label = new JLabel("This is a Swing frame",
        JLabel. CENTER);

        frame. add(label);
}
```

```

        frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
        frame.setSize(350, 200); // width=350, height=200
        frame.setVisible(true); // Display the frame
    }

}

```

JPanel

The JPanel class provides general-purpose containers for lightweight components. By default, panels do not add colors to anything except their own background.

In many types of look and feel, panels are transparent by default. Opaque panels work well as content panes and can help with painting efficiently. A transparent panel draws no background.

JPanel API

Following are some the important methods of JPanel:

Method	Description
JPanel()	Creates a new JPanel with a double buffer and a flow layout.
JPanel(LayoutManager layout)	Create a new buffered JPanel with the specified layout manager.
JPanel(LayoutManager layout, boolean isDoubleBuffered)	Creates a new JPanel with the specified layout manager and buffering strategy.
AccessibleContext getAccessibleContext()	Gets the AccessibleContext associated with the JPanel.

Code Example

Following code snippet explains use of simple methods of Panel.

```
package com.seed.project.panelProject;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
// DemoDrawing
public class DemoDrawing extends JFrame {

    DrawingArea left = new DrawingArea();
    DrawingArea right = new DrawingArea();
    int flip = 0;

    DemoDrawing() {
        left.setBackground(Color.white);
        right.setBackground(Color.black);

        JButton b = new JButton("Clik Here");
        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // When button is pressed, change
                colors
                if (flip == 1) {
                    left.setMyColor(Color.red);
                    right.setMyColor(Color.blue);
                    flip = 0;
                }else{
                    left.setMyColor(Color.blue);
                    right.setMyColor(Color.red);
                    flip = 1;
                }
            }
        });
    }

    // Add components to layout
}
```

```
Container content = this.getContentPane();
content.setLayout(new FlowLayout());
content.add(b);
content.add(left);
content.add(right);
this.setTitle("Demo Drawing by using JPanel");
this.pack();
}
public static void main(String[] args) {
    JFrame windo = new DemoDrawing();

windo.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    windo.setVisible(true);
}
// DrawingArea
class DrawingArea extends JPanel {
    //fields
    Color drawColor;

    public DrawingArea() {
        drawColor = Color.green;
        setPreferredSize(new Dimension(100,100));
    }
    //paintComponent
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(drawColor);
        int w = getWidth();
        int h = getHeight();
        g.drawLine(0, 0, w, h); // upper left to lower
right.
        g.drawLine(0, h, w, 0); // lower left to upper
right.
    }
    // setMyColor
    public void setMyColor(Color x) {
```

```

        drawColor = x;
        repaint(); // color changed, must repaint
    }
}

```

Window

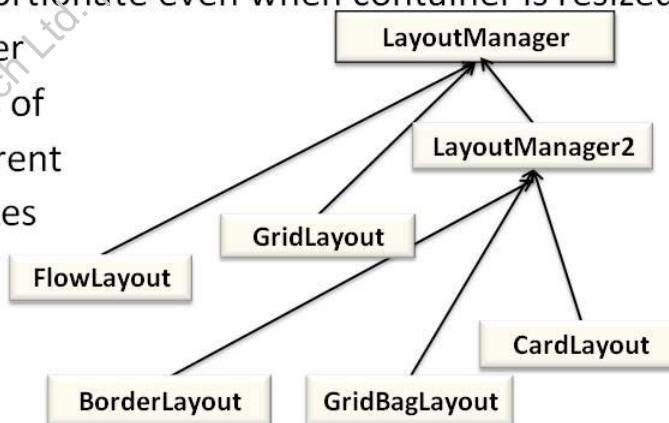
A Window object is a top-level window with no borders and no menubar. The default layout for a window is BorderLayout. A window must have a frame, dialog, or another window defined as its owner when it is constructed. To arrange a window in an application the physical screen coordinates are required . The origin of the virtual-coordinate system is at the upper left-hand corner of the primary physical screen.

Window API :Following are some of the important methods of Window:

Method	Description
Window(Frame owner)	Constructs a new invisible window with the specified Frame as its owner.
Window(Window owner)	Constructs a new invisible window with the specified Window as its owner.
Window(Window owner, GraphicsConfiguration gc)	Constructs a new invisible window with the specified window as its owner and a graphics configuration of a screen device.
void dispose()	Releases all of the native screen resources used by the Window, its subcomponents, and all of its owned children.
Window[] getOwnedWindows()	Returns an array containing all the windows the window currently owns.
void setBounds(int x, int y, int width, int height)	Moves and resizes this component.

Layout Management

- Consists of Layout and LayoutManager classes
- Components' size and placement while rendering is controlled by layout
- Keeps the UI proportionate even when container is resized
- Frees the developer from complexities of rendering on different resolutions and sizes



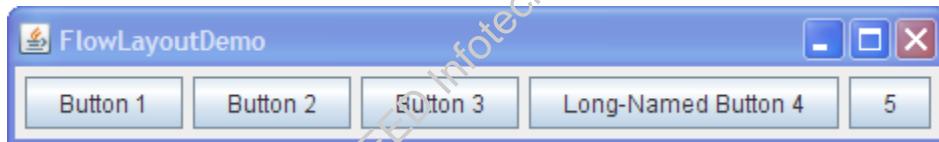
A Layout is a specific structure that the application will follow for building complex User Interfaces. The individual components are added to the interface in a sequential order and their placing on the screen is governed according to a layout. In many GUI building tools, component size and position need to be specified by the designer. When the UI look and feel is designed for a particular operating system and/or a device size, this is an appropriate way. When the UI can be viewed on any operating system and with any size, the fixed size UI does not look very professional. Java handles this issue with more elegant solution called Layout and LayoutManager.

Layout class provides certain structural arrangement of the UI components. Specialty of using these layouts is the proportionate sizes of components are maintained even when the container is resized. This keeps the overall components relative position and size intact without disturbing the structure of the UI when rendered at different resolutions. All Layouts implement an interface called LayoutManager or its versions to implement this behavior. This ensures that on any device, at any display resolution the UI would look similar. Programmer is free

from the actual complexities of rendering the components according to the display size. Hence, choice of Layout becomes a very integral decision when finalizing the look-and-feel of the application. There are many pre-built layouts available in JDK. If one is not happy with anyone of these, user defined classes can be created by implementing the `LayoutManager` interface.

FlowLayout

`FlowLayout` simply lays the components in a single row, starting a new row if its container is not wide enough to accommodate the next component added. It is a default layout manager for `JPanel`.



FlowLayout API

Following are some important methods of `FlowLayout`:

Method	Description
<code>FlowLayout()</code>	Constructs a new <code>FlowLayout</code> object with a centered alignment and horizontal and vertical gaps with the default size of 5 pixels.
<code>FlowLayout(int align)</code>	Creates a new flow layout manager with the indicated alignment and horizontal and vertical gaps with the default size of 5 pixels. The alignment argument can be <code>FlowLayout.LEADING</code> , <code>FlowLayout.CENTER</code> or <code>FlowLayout.TRAILING</code> . When the <code>FlowLayout</code> object controls a container with a left-to-right component orientation (the default), the <code>LEADING</code> value specifies the components to be

	left-aligned and the TRAILING value specifies the components to be right-aligned.
FlowLayout (int align, int hgap, int vgap)	Creates a new flow layout manager with the indicated alignment and the indicated horizontal and vertical gaps. The hgap and vgap arguments specify the number of pixels to put between components.
void addLayoutComponent(String name, Component comp)	Adds the specified component to the layout.

Code Example

Following code is for the understanding the concept of FlowLayout:

```
package com.seed.project.layoutProject;
import java.awt.*;
class DemoFlow extends JFrame
{ JButton b1, b2, b3, b4;
public DemoFlow () {
    Container cp=getContentPane ();
    b1=new JButton ("Button b1");
    cp.add (b1);
    b2=new JButton ("Button b2");
    cp.add (b2);
    b3=new JButton ("Button b3");
    cp.add (b3);
    b4=new JButton ("Button b4");
    cp.add (b4);
    cp.setLayout (new FlowLayout (FlowLayout.
CENTER));
}
```

```
public static void main (String [] args)
{
    DemoFlow d=new DemoFlow ();
    d.setSize (300,300);
    d.setVisible (true);
}
}
```

BorderLayout

BorderLayout divides the container into five areas called West, East, North, South, and Center.

It is default layout for ContentPane.



BorderLayout API

Following are some important methods of BorderLayout.

Method	Description
BorderLayout ()	Creates border layout. It is the default constructor.
BorderLayout (int horizontalGap, int verticalGap)	Defines a border layout with specified gaps between components.
setHgap (int)	Sets the horizontal gap between components.

setVgap(int)	Sets the vertical gap between components.
void addLayoutComponent(Component comp, Object constraints)	Adds the specified component to the layout, using the specified constraint object.

Code Example

Following code is for the understanding the concept of BorderLayout

```
package com.seed.project.layoutProject;
import java.awt.*;
import javax.swing.*;
class DemoBorder extends JFrame
{
    JButton b1, b2, b3, b4, b5;

    public DemoBorder ()
    {
        Container cp=getContentPane ();
        cp.setLayout (new BorderLayout ());
        b1=new JButton ("NORTH");
        cp.add (b1, BorderLayout.NORTH);

        b2=new JButton ("SOUTH");
        cp.add (b2, BorderLayout.SOUTH);

        b3=new JButton ("WEST");
        cp.add (b3, BorderLayout.WEST);

        b4=new JButton ("EAST");
        cp.add (b4, BorderLayout.EAST);

        b5=new JButton ("CENTER");
        cp.add (b5, BorderLayout.CENTER);
        //cp.add (b5);
    }
}
```

```

    }
    public static void main (String[] args)
    {
        DemoBorder d=new DemoBorder();
        d.setSize(300,300);
        d.setVisible (true);
    }
}

```

GridLayout

GridLayout arranges the components in the form of rows and columns. All the cells are of the same size.



GridLayout API

Following are some important methods of GridLayout:

Method	Description
GridLayout(int rows, int cols)	Creates a grid layout with the specified number of rows and columns. All components in the layout are given equal size. One, but not both, of rows and columns can be zero, which means that any number of objects can be placed in a row or in a column.
GridLayout(int rows, int cols, int hgap, int vgap)	Creates a grid layout with the specified number of rows and

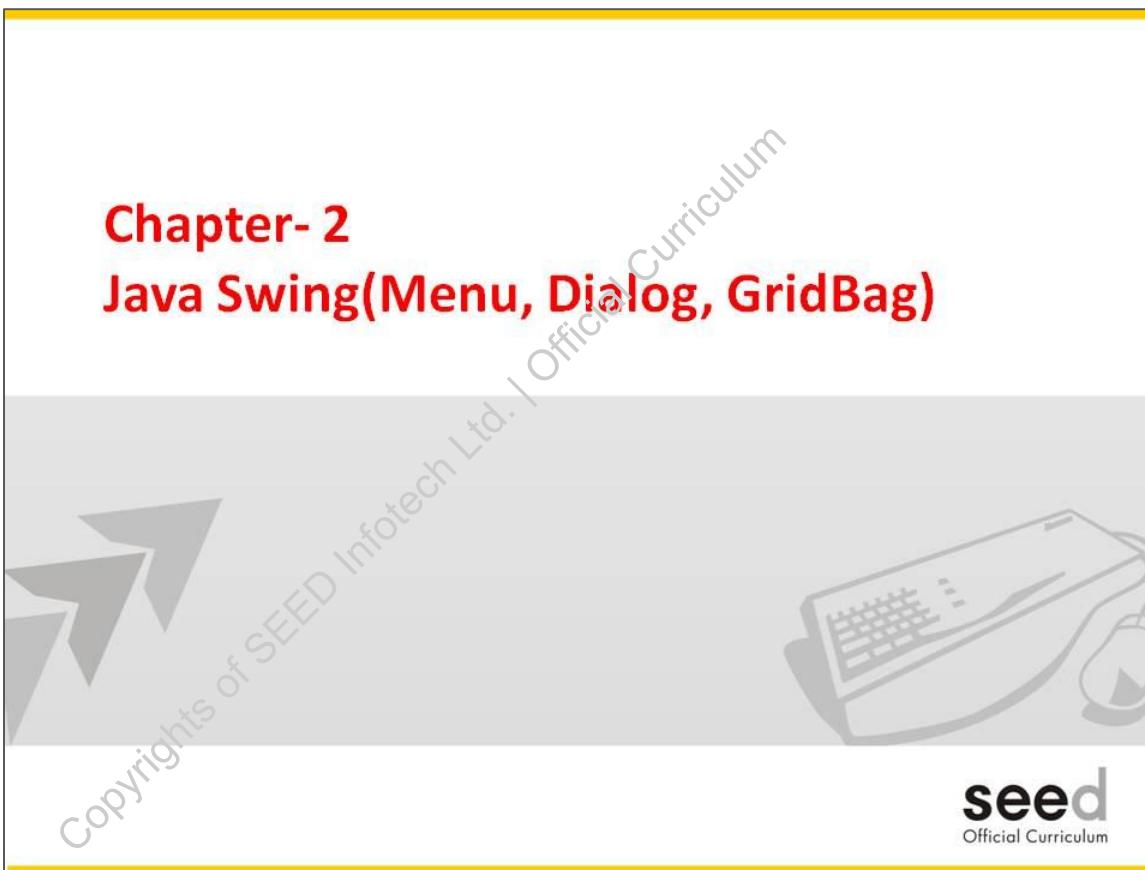
	columns. In addition, the horizontal and vertical gaps are set to the specified values. Horizontal gaps are places between each of columns. Vertical gaps are placed between each of the rows.
void addLayoutComponent(Component comp, Object constraints)	Adds the specified component to the layout, using the specified constraint object.

Code Example

Following code is for the understanding the concept of GridLayout :

```
package com.seed.project.layoutProject;
import java.awt.*;
class DemoGrid extends JFrame
{
    JButton b1, b2, b3, b4, b5, b6, b7;
    public DemoGrid ()
    {
        Container cp=getContentPane ();
        b1=new JButton ("Button 1");
        cp.add (b1);
        b2=new JButton ("Button 2");
        cp.add (b2);
        b3=new JButton ("Button 3");
        cp.add (b3);
        b4=new JButton ("Button 4");
        cp.add (b4);
        b5=new JButton ("Button 5");
        cp.add (b5);
        b6=new JButton ("Button 6");
        cp.add (b6);
        b7=new JButton ("Button 7");
        add (b7);
    }
}
```

```
        cp.setLayout (new GridLayout (2,3)); //many con  
can be added here  
    }  
    public static void main(String[] args)  
    {  
        DemoGrid d=new DemoGrid();  
        d.setSize(300,300);  
        d.setVisible(true);  
    }  
}
```



A slide titled "Chapter- 2 Java Swing(Menu, Dialog, GridBag)" featuring a background image of a computer keyboard and mouse, and a large grey arrow pointing right.

Chapter- 2
Java Swing(Menu, Dialog, GridBag)

Copyrights of SEED Infotech Ltd. | Official Curriculum

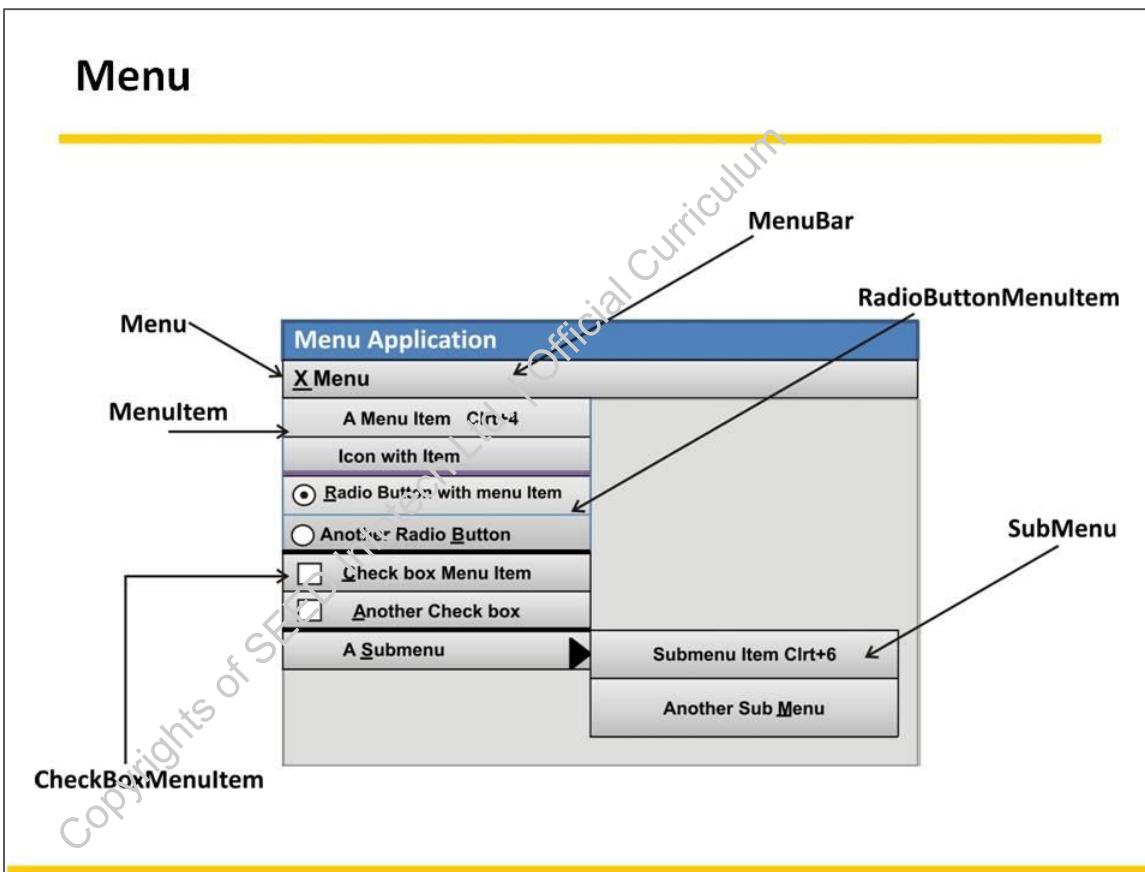
seed
Official Curriculum

This chapter covers GUI applications using Menus and Dialogs, in-built dialogs such as `JFileChooser`, `JColorChooser`; Layouts like `GridBagLayout`, custom layout. This chapter also covers 2D graphics.

Objectives

At the end of this chapter you will be able to:

- Construct application UI with menus.
- Use pop up or context menu.
- Understand the concept of dialog box.
- Use the in-built dialogs as part of the application like `JFileChooser`.
- Create modal or modeless dialogs using `JDialog`.
- Create an UI using `GridBagLayout`.
- Create an UI using Custom Layout.
- Working with 2D Graphics.



All the applications require user interaction at some point. Controls are provided to interact with application. There may be so many controls that the user workable area will be reduced. For example, in a text editor application, there are three controls - Cut, Copy and Paste. The user may want to perform only one operation at a time. All the three operations are performed on text and are related to text editing. There will also be controls like Open, Save, and Close. These actions are performed on a file and again, not all will be used at the same time but having them on the user interface all the time will reduce some more workable area. So, can anything be done to group these operations and make available only one of them at a time? At the same time, can more workable area be made available to the user?

The answer to all these questions is the menu . A menu card that is given in a restaurant is a very good example. In the menu card, all the items that are available are categorized and grouped together under categories like starters, soups, salads, main course, desserts, etc.

Similarly, in any application, all the operations to be performed can be categorized and grouped together in a menu. In the above example of a text editor, the Cut, Copy and Paste will become the items in the Edit menu. Hence they are known as menu items. So, in the above example, we can have two menus - File and Edit. All file related operations will come under the File menu and the text editing operations will come under Edit menu.

Menu

- Used to represent the user-driven actions with execution choices
- Two types of menus available
 - MenuBar based menus for entire application
 - Popup Menus for context-based execution

Menus are primarily used to provide the user with a possibility of directing the flow of execution. Normally the menu options coincide with the actions a user is going to perform in the real world. E.g. File related activities could be represented by a menu option called as File. Similarly opening of a file could be a sub-option under the umbrella menu option File -> Open. To provide functionality for menus in Java, Swing makes use of two modes of menu creation.

JMenuBar

A menu bar is a simple bar or strip which is generally placed at the top of the interface.

There can be more than one menu on the menu bar.

JMenu

A menu is a component which contains number of menu items.

JMenuItem

A menu can have several tasks or sub-tasks related to it. These are called menu items.

JSeparator

To distinguish between menu items, a line is used. This line is called a separator.

JCheckBoxMenuItem

A checkbox is provided when the user is allowed to select more than one option among many.

So, when a user can select more than one option from a menu, a check box menu item is provided.

JRadioButtonMenuItem

A radio button is provided when the user can select only one option.

So, when a user can select only one option from a menu, a radio button menu item should be provided.

Menu in Java Swing

- Menus in Swing are UI components which may contain other UI components.
- A MenuBar acts as a container for multiple menu items.
- MenuItem can be a Check Box or Radio Button.
- Events are associated with individual MenuItem.

Menus in swing are UI component which may contain other UI component.

JMenu

Following are some important methods of JMenu class:

Method	Description
JMenuBar ()	Creates a menu bar.
JMenu add (JMenu)	Adds the menu to the end of the menu bar.
void setJMenuBar (JMenuBar) JMenuBar getJMenuBar ()	Sets or gets the menu bar of an applet, dialog, frame, internal frame, or root pane.
JMenu () JMenu (String)	Creates a menu. The string specifies the text to display for the menu. The

JMenu (Action)	Action specifies the text and other properties of the menu
JMenuItem add(JMenuItem) JMenuItem add(String)	<p>Adds a menu item to the current end of the menu. If the argument is a string, then the menu automatically creates a JMenuItem object that displays the specified text.</p> <p>Version Note: Before 1.3, the only way to associate an Action with a menu item was to use menu's add(Action) method to create the menu item and add it to the menu. As of 1.3, that method is no longer recommended. You can instead associate a menu item with an Action using the setAction method.</p>
void addSeparator()	Adds a separator to the current end of the menu.
JMenuItem insert(JMenuItem, int) void insert(String, int) void insertSeparator(int)	Inserts a menu item or separator into the menu at the specified position. The first menu item is at position 0, the second at position 1, and so on. The JMenuItem and String arguments are treated the same as in the corresponding add methods.
void remove(JMenuItem) void remove(int) void removeAll()	Removes the specified item(s) from the menu. If the argument is an integer, then it specifies the position of the menu item to be removed.

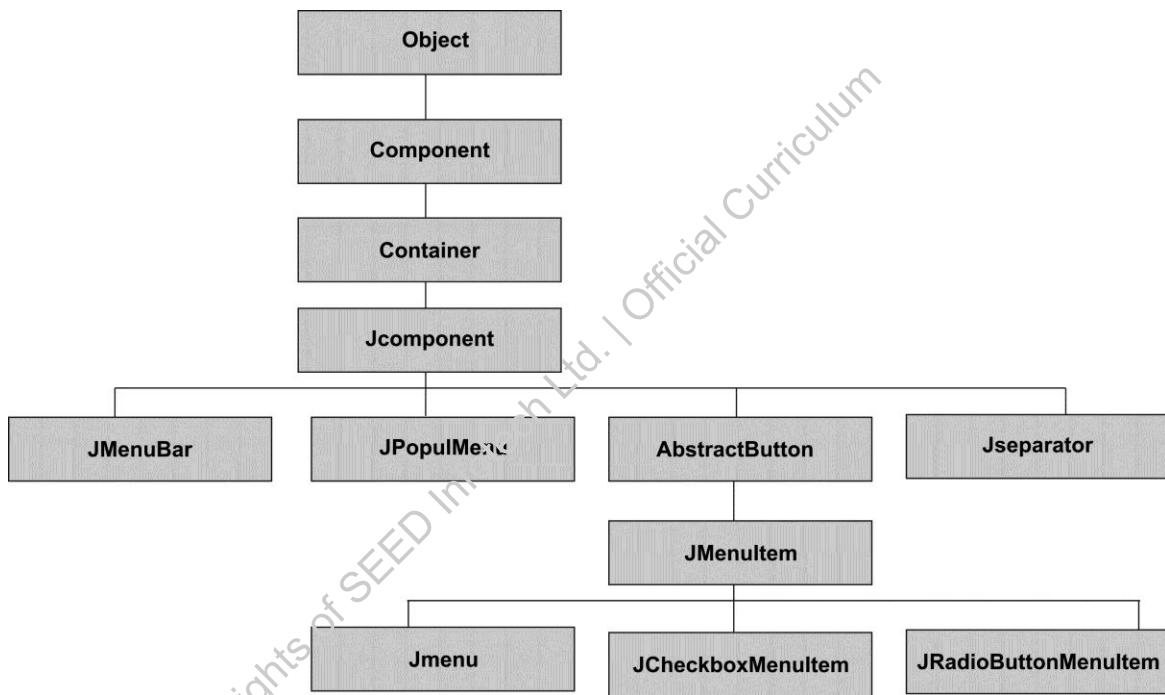
JMenuItem

Following are some important methods of JMenuItem class:

Method	Description
JMenuItem() JMenuItem(String) JMenuItem(Icon) JMenuItem(String, Icon) JMenuItem(String, int) JMenuItem(Action)	Creates an ordinary menu item. The icon argument, if present, specifies the icon that the menu item should display. Similarly, the string argument specifies the text that the menu item should display. The integer argument specifies the keyboard mnemonic to use. The constructor with the Action parameter, which was introduced in 1.3, sets the menu item's Action, causing the menu item's properties to be initialized from the Action.
JCheckBoxMenuItem() JCheckBoxMenuItem(String) JCheckBoxMenuItem(Icon) JCheckBoxMenuItem(String, Icon) JCheckBoxMenuItem(String, boolean) JCheckBoxMenuItem(String, Icon, boolean)	Creates a menu item that looks and acts like a check box. The string argument, if any, specifies the text that the menu item should display. If you specify true for the boolean argument, then the menu item is initially selected (checked). Otherwise, the menu item is initially unselected.
JRadioButtonMenuItem() JRadioButtonMenuItem(String)	Creates a menu item that looks and acts like a radio

<pre>JRadioButtonMenuItem(Icon) JRadioButtonMenuItem(String, Icon) JRadioButtonMenuItem(String, boolean) JRadioButtonMenuItem(Icon, boolean) JRadioButtonMenuItem(String, Icon, boolean)</pre>	<p>button. The string argument, if any, specifies the text that the menu item should display. If you specify true for the boolean argument, then the menu item is initially selected. Otherwise, the menu item is initially unselected.</p>
<pre>void setState(boolean) boolean getState() (in JCheckBoxMenuItem)</pre>	<p>Set or get the selection state of a check box menu item.</p>
<pre>void setEnabled(boolean)</pre>	<pre>void setEnabled(boolean)</pre>
<pre>void setMnemonic(int)</pre>	<p>Set the mnemonic that enables keyboard navigation to the menu or menu item. Use one of the VK constants defined in the KeyEvent class.</p>
<pre>void setAccelerator(KeyStroke)</pre>	<p>Set the accelerator that activates the menu item.</p>
<pre>void setActionCommand(String)</pre>	<p>Set the name of the action performed by the menu item.</p>
<pre>void addActionListener(ActionListener) void addItemListener(ItemListener)</pre>	<p>Add an event listener to the menu item.</p>
<pre>void setAction(Action)</pre>	<p>Set the Action associated with the menu item.</p>

Menu-hierarchy Diagram



Algorithm to create Menus

- Steps to create Menu:

1. Create Menu bar –using class JMenuBar
2. Add MenuBar to frame – setJMenuBar (bar)
3. Create Menu- using class JMenu
4. Add JMenu to JMenuBar
5. Create menu item – using class JMenuItem
6. Add JMenuItem to JMenu

Following code example shows the use of APIs for creating menus.

```
package com.seed.project.me
import java.awt.*;
import javax.swing.*;
class MenuDemo extends JFrame{
    MenuDemo() {
        JMenuBar mb=new JMenuBar(); Create JMenuBar
        setJMenuBar(mb);
        JMenu file=new JMenu("File");
        JMenu edit=new JMenu("Edit"); Create JMenu
        JMenu format=new JMenu("Format");
        mb.add(file);
        mb.add(edit); add JMenu to JMenuBar
        mb.add(format);
        JMenuItem cut=new JMenuItem("Cut");
        JMenuItem copy=new JMenuItem("Copy");
    }
}
```

```
JMenuItem paste=new JMenuItem("P  
edit.add(cut); }  
edit.add(copy); }  
edit.add(paste); }  
}  
public static void main(String args)  
MenuDemo m=new MenuDemo();  
m.setSize(500,500);  
m.setVisible(true);  
}  
}
```

Create JMenuItem

add JMenuItem to JMenu

Pop-up Menus

- This menu is also known as context menu.
- It has no fixed place on the user interface.
- It can appear wherever the right mouse button is clicked.
- Steps to create pop-up Menu:
 1. Create Pop-up Menu- using class JPopupMenu.
 2. Create menu item – using class JMenuItem.
 3. Add JMenuItem to JPopupMenu.
 4. Add JPopupMenu to current JFrame.



A Pop-up Menu is a menu which is not attached to a JMenuBar.

A JPopupMenu is similar to a Menu as it contains JMenuItem objects. The Pop-up Menu can be popped over any component. This is also called as trigger pop-up menu.

Menu Items are added in a similar way to Menu class. To explicitly show pop-up Menu you have to use show() method.

JPopupMenu API

Following are some important methods of **JPopupMenu** class

Method	Description
JPopupMenu() JPopupMenu(String)	Creates a popup menu. The optional string argument specifies the title that a look and feel might display as part of the popup window.
JMenuItem	Adds a menu item to the current end of the popup

<pre>add(JMenuItem) JMenuItem add(String)</pre>	<p>menu. If the argument is a string, then the menu automatically creates a JMenuItem object that displays the specified text.</p> <p>Version Note: Before 1.3, the only way to associate an Action with an item in a popup menu was to use the popup menu's add(Action) method to create the menu item and add it to the popup menu. As of 1.3, that method is no longer recommended. You can instead associate a menu item with an Action using the setAction method.</p>
<pre>void addSeparator()</pre>	<p>Adds a separator to the current end of the popup menu.</p>
<pre>void insert(Component, int)</pre>	<p>Inserts a menu item into the menu at the specified position. The first menu item is at position 0, the second at position 1, and so on. The Component argument specifies the menu item to add.</p>
<pre>void remove(int) void removeAll()</pre>	<p>Removes the specified item(s) from the menu. If the argument is an integer, then it specifies the position of the menu item to be removed.</p>
<pre>static void setLightWeightPopupEnabled(boolean)</pre>	<p>By default, Swing implements a menu's window using a lightweight component. This can cause problems if you use any heavyweight components in your Swing program. As a workaround, invoke JPopupMenu.setLightWeightPopupEnabled (false).</p>
<pre>void show(Component, int, int)</pre>	<p>Display the popup menu at the specified x,y position (specified in that order by the integer arguments) in the coordinate system of the specified component.</p>

Code Example

Following code snippet explains the simple methods of a JPopupMenu.

```
package com.seed.project.menuProject;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class PopupMenuDemo extends JFrame{
    JButton b;
    JTextField msg;
    PopupAppMenu m;
    public PopupMenuDemo() {
        Container cp= getContentPane();
        setSize(200, 200);
        b = new JButton("Pop-up Menu");
        cp.add(b, BorderLayout.NORTH);
        msg = new JTextField();
        msg.setEditable(false);
        add(msg, BorderLayout.SOUTH);
        m = new PopupAppMenu(this); //create instance of
        popup menu
        add(m); //adding popup menu to this frame
        b.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e) {
                m.show(b, 20, 20);
            }
        });
        addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent e) {
                if (e.isPopupTrigger())
                    m.show(e.getComponent(), e.getX(), e.getY());
            }
            public void mouseReleased(MouseEvent e) {
                if (e.isPopupTrigger())
                    m.show(e.getComponent(), e.getX(), e.getY());
            }
        });
    }
}
```

```
}

public static void main(String[] args) {
    PopupMenuDemo app = new PopupMenuDemo();
    app.setVisible(true);
}

class PopupAppMenu extends JPopupMenu
implements ActionListener{
    PopupMenuDemo ref;
    public PopupAppMenu(PopupMenuDemo ref) {
        super("File");
        this.ref = ref;
        JMenuItem mi;
        //JFrame f=new JFrame("my popup Frame");
        add(mi = new JMenuItem("Copy"));
        mi.addActionListener(this);
        add(mi = new JMenuItem("Open"));
        mi.addActionListener(this);
        add(mi = new JMenuItem("Cut"));
        mi.addActionListener(this);
        add(mi = new JMenuItem("Paste"));
        mi.addActionListener(this);
    } //add menuItems to popup menu and add listeners.
    public void actionPerformed (ActionEvent e){
        String item = e.getActionCommand();
        ref.msg.setText("Option Selected: " + item);
    }
}
```

Dialog Box

- What is 'Dialog Box' ?
 - A dialog-box is a type of window used to enable common communication or dialog between a computer and its user.
 - A dialog box is most often used to provide the user with the means for specifying how to implement a command, or to respond to a question or an "alert".
 - Dialog is an independent sub window that is used to perform a temporary activity away from the main application.

In a Graphical User Interface, a window that is used to communicate with the user or establish a dialog between the user and the application is called a dialog box . It may communicate information to the user; prompt the user for a response, or both.

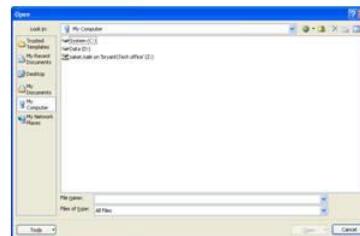
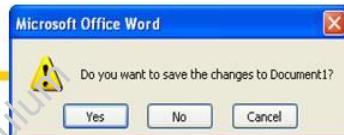
The simplest type of dialog box is the warning, which displays a message and may require the user to acknowledge that the message has been read, usually by clicking "OK", or a decision as to whether or not an action should continue, by clicking "OK" or "Cancel".

Some dialog boxes are standard - like the warning message, error message, save file, enter password. These are called standard dialog boxes.

A dialog box can also be customized. Such a dialog box is called a custom dialog box.

Type of Dialogs

- Modal dialogs
 - Doesn't allow any other window in the application to be accessed till Dialog is closed.
 - Closing dialog window
 - File open dialog window
 - Color chooser dialog window
 - Modeless dialogs
 - Allows other windows in the application to be accessed while dialog is displayed.
 - Find, Replace dialog boxes.



There are two types of Dialog boxes:

- Modal Dialog Box
- Modeless Dialog Box

Modal Dialog Box

A dialog box that temporarily halts the application and the user cannot continue until the dialog has been closed is called a modal dialog box. The application may require some additional information before it can continue, or may simply wish to confirm that the user wants to proceed with a potentially dangerous course of action. The application continues to execute only after the dialog box is closed; until then the application halts.

For example, to start an application it is necessary to enter the password. Unless the user enters the correct password, the application does not proceed or when saving a file, the user gives a name of an existing file; a warning is shown that a file with the same name exists, whether it should be overwritten or be saved in different name. The file will not be saved unless the user selects "Ok" or "Cancel".

Modeless Dialog Box

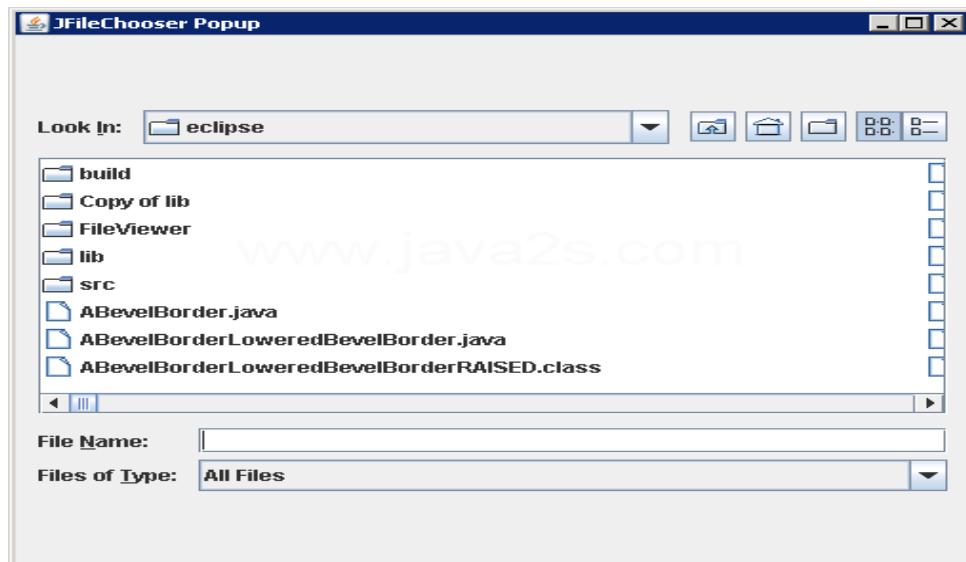
Another type of dialog box that is used is a modeless dialog box . It is used when the requested information is not essential to continue, and so the window can be left open while work continues somewhere else. For example, when working in a text editor, the user wants to find and replace a particular word. This can be done using a dialog box which asks for the word to be found and replaced. The user can continue to work even if this box is open.

Built-in Dialog Box

- Java provides readymade dialog box. e.g. JFileChooser, JColorChooser.
- It is a modal dialog box.
- This control facilitates the selection of a file from the file system.

Java provides readymade dialog box. E.g. JFileChooser, JColorChooser
JFileChooser

As the name itself suggests, this control facilitates the selection of a file from the file system.



JFileChooser API

Following are the some important methods of JFileChooser class

Method	Description
<pre>JFileChooser() JFileChooser(File) JFileChooser(String)</pre>	<p>Creates a file chooser instance. The File and String arguments, when present, provide the initial directory.</p>
<pre>int showOpenDialog(Component) int showSaveDialog(Component) int showDialog(Component, String)</pre>	<p>Shows a modal dialog containing the file chooser. These methods return APPROVE_OPTION if the user approved the operation and CANCEL_OPTION if the user cancelled it. Another possible return value is ERROR_OPTION, which means an unanticipated error occurred.</p>
<pre>void setSelectedFile(File) File getFile()</pre>	<p>Sets or obtains the currently selected file or (if directory selection has been enabled) directory.</p>
<pre>void setSelectedFiles(File[]) File[] getSelectedFiles()</pre>	<p>Sets or obtains the currently selected files if the file chooser is set to allow multiple selection.</p>
<pre>void setFileSelectionMode(int) int getFileSelectionMode() boolean isDirectorySelectionEnabled()</pre>	<p>Sets or obtains the file selection mode. Acceptable values are FILES_ONLY (the</p>

<pre>boolean isFileSelectionEnabled()</pre>	<p>default), DIRECTORIES_ONLY, and FILES_AND_DIRECTORIES.</p> <p>Interprets whether directories or files are selectable according to the current selection mode.</p>
<pre>void setMultiSelectionEnabled(boolean) boolean isMultiSelectionEnabled()</pre>	<p>Sets or interprets whether multiple files can be selected at once. By default, a user can choose only one file.</p>
<pre>void setAcceptAllFileFilterUsed(boolean) boolean isAcceptAllFileFilterUsed()</pre>	<p>Sets or obtains whether the AcceptAll file filter is used as an allowable choice in the choosable filter list; the default value is true.</p>
<pre>void ensureFileIsVisible(File)</pre>	<p>Scrolls the file chooser's list such that the indicated file is visible.</p>
<pre>void setCurrentDirectory(File) File getCurrentDirectory()</pre>	<p>Sets or obtains the directory whose files are displayed in the file chooser's list.</p>
<pre>void changeToParentDirectory()</pre>	<p>Changes the list to display the current directory's parent.</p>
<pre>void rescanCurrentDirectory()</pre>	<p>Checks the file system and updates the chooser's list.</p>
<pre>void setDragEnabled(boolean) boolean getDragEnabled()</pre>	<p>Sets or obtains the property that determines whether automatic drag handling is</p>

	enabled.
void setAccessory (javax.swing.JComponent) JComponent getAccessory()	Sets or obtains the file chooser's accessory component.
void setFileFilter(FileFilter) FileFilter getFileFilter()	Sets or obtains the file chooser's primary file filter.
void setFileView(FileView) FileView getFileView()	Sets or obtains the chooser's file view.
FileFilter[] getChoosableFileFilters() void addChoosableFileFilter (FileFilter) boolean removeChoosableFileFilter (FileFilter) void resetChoosableFileFilters() FileFilter getAcceptAllFileFilter()	Sets, obtains, or modifies the list of user-choosable file filters.
void setFileHidingEnabled (boolean) boolean isFileHidingEnabled()	Sets or obtains whether hidden files are displayed.
void setControlButtonsAreShown (boolean) boolean getControlButtonsAreShown ()	Sets or obtains the property that indicates whether the Approve and Cancel buttons are shown in the file chooser. This property is true by default.

Code Example

Following code snippet explains the simple methods of a JFileChooser

```
package com.seed.project.filechooserProject;
//import statements
```

```
public class FileSamplePanel {  
  
    public static void main(String args[]) {  
        JFrame frame = new JFrame("JFileChooser Popup");  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)  
);  
        Container contentPane = frame.getContentPane();  
  
        final JLabel directoryLabel = new JLabel(" ");  
        directoryLabel.setFont(new Font("Serif", Font.BOLD  
| Font.ITALIC, 36));  
        contentPane.add(directoryLabel, BorderLayout.NORTH)  
;  
  
        final JLabel filenameLabel = new JLabel(" ");  
        filenameLabel.setFont(new Font("Serif", Font.BOLD |  
Font.ITALIC, 36));  
        contentPane.add(filenameLabel, BorderLayout.SOUTH);  
  
        JFileChooser fileChooser = new JFileChooser(".");  
        fileChooser.setControlButtonsAreShown(false);  
        contentPane.add(fileChooser, BorderLayout.CENTER);  
  
        ActionListener actionPerformed = new ActionListener()  
) {  
            public void actionPerformed(ActionEvent actionEve  
nt) {  
                JFileChooser theFileChooser = (JFileChooser) ac  
tionEvent  
                    .getSource();  
                String command = actionEvent.getActionCommand()  
;  
                if (command.equals(JFileChooser.APPROVE_SELECTI  
ON)) {  
                    File selectedFile = theFileChooser.getSelecte  
dFile();  
                }  
            }  
        };  
    }  
}
```

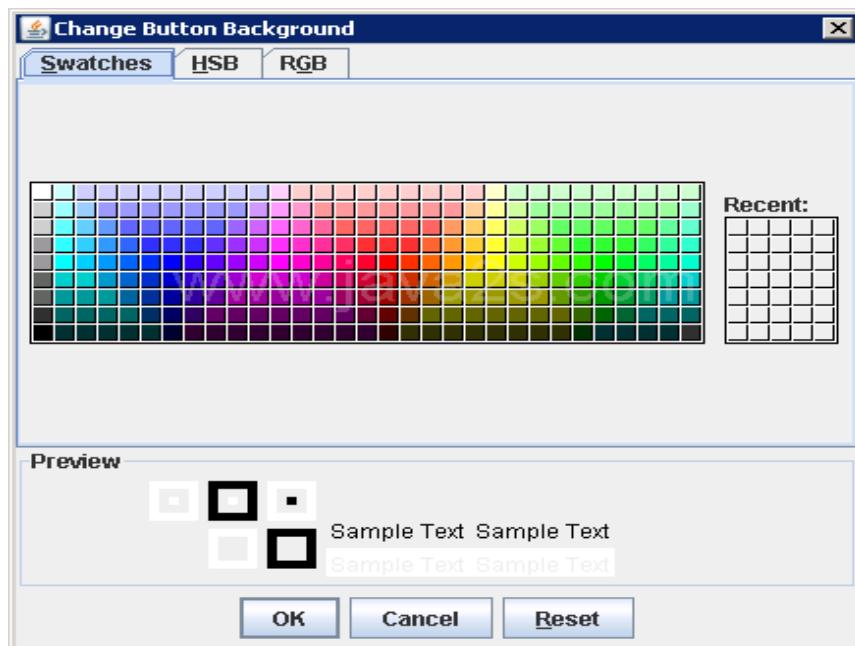
```
        directoryLabel.setText(selectedFile.getParent()
());
        filenameLabel.setText(selectedFile.getName())
;
    } else if (command.equals(JFileChooser.CANCEL_SELECTION)) {
        directoryLabel.setText(" ");
        filenameLabel.setText(" ");
    }
}
};

fileChooser.addActionListener(actionListener);

frame.pack();
frame.setVisible(true);
}

}
```

JColorChooser



JColorChooser API

Following are the some important methods of JFileChooser

Method	Description
JColorChooser() JColorChooser(Color) JColorChooser(ColorSelectionModel)	Create a color chooser. The default constructor creates a color chooser with an initial color of Color.white. Use the second constructor to specify a different initial color. The ColorSelectionModel argument, when present, provides the color chooser with a color selection model.
Color showDialog(Component, String, Color)	Create and show a color chooser in a modal dialog. The Component argument is the parent of the dialog, the String argument specifies the dialog title, and the Color argument specifies the chooser's initial color.
JDialog createDialog(Component, String, boolean, JColorChooser, ActionListener, ActionListener)	Create a dialog for the specified color chooser. As with showDialog, the Component argument is the parent of the dialog and the String argument specifies the dialog title. The other arguments are as follows: the boolean specifies whether the dialog is modal, the JColorChooser is the color chooser to display in the dialog, the first ActionListener is for the OK button, and the second is for the Cancel button.
void setPreviewPanel(JComponent)	Set or get the component used to preview the color selection. To remove the preview panel, use new JPanel() as an argument. To specify the default

<code>getPreviewPanel()</code>	preview panel, use null.
<code>void setChooserPanels(AbstractColorChooserPanel[])</code> <code>AbstractColorChooserPanel[] getChooserPanels()</code>	Set or get the chooser panels in the color chooser.
<code>void addChooserPanel(AbstractColorChooserPanel)</code> <code>AbstractColorChooserPanel removeChooserPanel(AbstractColorChooserPanel)</code>	Add a chooser panel to the color chooser or remove a chooser panel from it.
<code>void setDragEnabled(boolean)</code> <code>boolean getDragEnabled()</code>	Set or get the dragEnabled property, which must be true to enable drag handling on this component. The default value is false.
<code>void setColor(Color)</code> <code>void setColor(int, int, int)</code> <code>void setColor(int)</code> <code>Color getColor()</code>	Set or get the currently selected color. The three integer version of the setColor method interprets the three integers together as an RGB color. The single integer version of the setColor method divides the integer into four 8-bit bytes and interprets the integer as an RGB color as follows:
<code>void setSelectionModel(ColorSelectionModel)</code>	Set or get the selection model for the color chooser. This object contains the current selection and fires change events

ColorSelectionModel getSelectionModel ()	to registered listeners whenever the selection changes.
---	---

Code Example

Following code snippet explains the simple methods of a JColorChooser

```
package com.seed.project.colorchooserProject;  
// import statements  
  
public class ColorSample {  
    public static void main(String args[]) {  
        JFrame f = new JFrame("JColorChooser Sample");  
  
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
        final JButton button = new JButton("Pick to Change Background");  
  
        ActionListener actionListener = new ActionListener()  
        {  
            public void actionPerformed(ActionEvent actionEvent) {  
  
                Color initialBackground = button.getBackground();  
                Color background = JColorChooser.showDialog(null, "JColorChooser Sample", initialBackground);  
                if (background != null) {  
                    button.setBackground(background);  
                }  
            }  
        };  
        button.addActionListener(actionListener);  
        f.add(button, BorderLayout.CENTER);  
        f.setSize(300, 200);  
        f.setVisible(true);  
    }  
}
```

Custom Dialog

- Java provides number of standard dialogs such as JFileChooser, JColorChooser etc.
- To create custom dialog box use `JDialog` class.
- `JDialog` class inherits this behavior from the AWT Dialog class.

Dialogs are generally used for displaying error conditions, warnings or accepting user information. Every dialog box is depending on a Frame component. Java provides number of standard dialogs such as `JFileChooser` etc. To create custom dialog box use `JDialog` class.

Some of the important methods of `JDialog` class:

Method	Description
<code>JDialog()</code> <code>JDialog(Dialog)</code> <code>JDialog(Dialog, boolean)</code> <code>JDialog(Dialog, String)</code> <code>JDialog(Dialog, String, boolean)</code> <code>JDialog(Dialog, String, boolean, GraphicsConfiguration)</code>	Creates a <code>JDialog</code> instance. The <code>Frame</code> argument, if any, is the frame (usually a <code>JFrame</code> object) that the dialog depends on. Make the <code>boolean</code> argument true to specify a modal dialog, false or absent to specify a non-modal dialog.

```

JDialog(Frame)
JDialog(Frame, boolean)
JDialog(Frame, String)
JDialog(Frame, String,
boolean)
JDialog(Frame, String,
boolean,
GraphicsConfiguration)
JDialog(Window owner)
JDialog(Window owner,
Dialog.ModalityType
modalityType)
JDialog(Window owner,
String title)
JDialog(Window owner,
String title,
Dialog.ModalityType
modalityType)
JDialog(Window owner,
String title,
Dialog.ModalityType
modalityType,
GraphicsConfiguration gc)

```

```

void
setContentPane(Container)
Container getContentPane()

```

Get and set the content pane, which is usually the container of all the dialog's components.

```

void
setDefaultCloseOperation
(int)
int
getDefaultCloseOperation()

```

Get and set what happens when the user tries to close the dialog. Possible values:
 DISPOSE_ON_CLOSE,
 DO NOTHING ON CLOSE,
 HIDE ON CLOSE (the default).

Some of the important methods of JOptionPane class:

Method	Description
<pre>JOptionPane() JOptionPane(Object) JOptionPane(Object, int) JOptionPane(Object, int, int) JOptionPane(Object, int, int, Icon) JOptionPane(Object, int, int, Icon, Object[]) JOptionPane(Object, int, int, Icon, Object[], Object)</pre>	Creates a JOptionPane instance.
<pre>static void showMessageDialog(Component, Object) static void showMessageDialog(Component, Object, String, int) static void showMessageDialog(Component, Object, String, int, Icon)</pre>	Show a one-button, modal dialog that gives the user some information. The arguments specify (in order) the parent component, message, title, message type, and icon for the dialog.
<pre>static int showOptionDialog(Component, Object, String, int, int, Icon, Object[], Object)</pre>	Show a customized modal dialog. The arguments specify (in order) the parent component, message, title, option type, message type, icon, options, and initial value for the dialog.
<pre>static int showConfirmDialog(Component, Object) static int showConfirmDialog(Component, Object, String, int)</pre>	Show a modal dialog that asks the user a question. The arguments specify (in order) the parent component, message, title, option type, message type, and icon for the dialog.

```
static int  
showConfirmDialog(Component,  
Object, String, int, int)  
static int  
showConfirmDialog(Component,  
Object, String, int, int,  
Icon)
```

Following code snippet explains the simple methods of a custom Dialog Box.

```
//modal dialog box  
package com.seed.project.jdialogProject;  
import java.awt.*;  
import java.awt.event.*;  
class MyDialog extends JDialog implements  
ActionListener  
{  
//class extends from JDialog  
    JTextField t1,parentTf;  
    JButton b1,b2;  
    String str;  
  
    public MyDialog (JFrame parent , boolean modal,  
JTextField tf)  
    {  
        super(parent,modal);  
        setLayout (new FlowLayout());  
        parentTf = tf;  
        t1=new JTextField(10);  
        b1=new JButton("ok");  
        b2=new JButton("cancel");  
        add(t1);  
        add(b1);  
        add (b2);  
        b1.addActionListener (this);  
        b2.addActionListener(this);  
        setSize (100,100);
```

```
}

public void actionPerformed(ActionEvent e)
{
    if (e.getSource() == b1)
    {
        str=t1.getText();
        parentTf.setText(str);
        dispose();
    }
    if (e.getSource() == b2)
    {
        dispose();
        parentTf.setText("");
    }
}

public class DialogFrame1 extends JFrame implements
ActionListener
{
    JTextField tf = new JTextField(20);
    JButton b;
    MyDialog dlg;
    String str;
    public dialogframe1(Mydialog d1)
    {
        dlg=d1;
    }

    DialogFrame1()
    {
        b= new JButton("Show");
        //create a instance of JDialo
        dlg= new MyDialog(this,true,tf);
        setLayout(new FlowLayout());
        setTitle("TestFrame");
    }
}
```

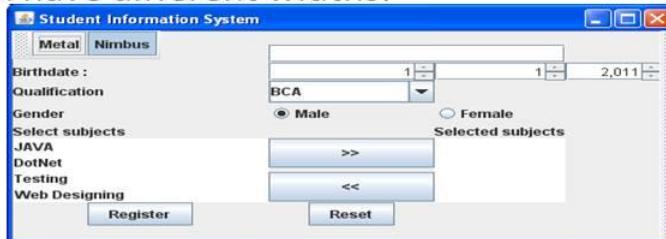
```
add(tf);
add(b);
setSize(200,200);
setVisible(true);
b.addActionListener(this);
}
public void actionPerformed(ActionEvent e)
{
    if (e.getSource() == b)
    {
        dlg.setVisible(true);
    }
}
public static void main(String[] args)
{
    DialogFrame1 f1 = new DialogFrame1();
}
}
```

Note

For modeless dialog just change boolean parameter to 'false' .

Layouts

- **GridBagLayout**
 - **GridBagLayout** is a sophisticated, flexible layout manager.
 - It aligns components by placing them within a grid of cells.
 - Components can size to more than one cell.
 - The rows in the grid can have different heights, and grid columns can have different widths.



Following code snippet explains the simple methods of a **GridBagLayout**:

```
package com.seed.project.projectGrid;
// import statements
public class GridBagDemo extends JFrame {

    public GridBagDemo() {
        JButton button;
        Container pane=getContentPane();
        pane.setLayout(new GridBagLayout());
        GridBagConstraints gridCon = new
GridBagConstraints();
        boolean shouldFill = false;
        if (shouldFill) {
            //natural height, maximum width
            gridCon.fill =
GridBagConstraints.HORIZONTAL;
        }
    }
}
```

```
button = new JButton("Create");
boolean shouldWeightX = false;
if (shouldWeightX) {
    gridCon.weightx = 0.5;
}
gridCon.fill = GridBagConstraints.HORIZONTAL;
gridCon.gridx = 0;
gridCon.gridy = 0;
pane.add(button, gridCon);
button = new JButton("Update");
gridCon.fill =
GridBagConstraints.FIRST_LINE_END;
gridCon.weightx = 0.5;
gridCon.gridx = 1;
gridCon.gridy = 0;
pane.add (button, gridCon);
button = new JButton("Delete");
gridCon.fill = GridBagConstraints.NORTHEAST;
gridCon.weightx = 0.5;
gridCon.gridx = 2;
gridCon.gridy = 1;
pane.add (button, gridCon);
button = new JButton("What else can be done?");
gridCon.fill = GridBagConstraints.CENTER;
gridCon.ipady = 25;      //make this component
tall
gridCon.weightx = 0.0;
gridCon.gridwidth = 3;
gridCon.gridx = 0;
gridCon.gridy = 3;
pane.add (button, gridCon);
button = new JButton("Retrieve");
gridCon.fill = GridBagConstraints.HORIZONTAL;
gridCon.ipady = 0;        //reset to default
gridCon.weighty = 1.0;    //request any extra
vertical space
```

```

        gridCon.anchor = GridBagConstraints.PAGE_END;
//bottom of space
        gridCon.insets = new Insets(10,0,0,0); //top
padding
        gridCon.gridx = 1;
        gridCon.gridwidth = 2; //2 columns wide
        gridCon.gridy = 3; //third row
        pane.add(button, gridCon);
    }
public static void main(String[] args) {
    GridBagDemo gd=new GridBagDemo();
    gd.setSize(500,500);
    gd.setVisible(true);
}
}

```

GridBag Constraints

Following are the some important properties of Grid Bag constraints

Constraints	Description
gridx, gridy	Specify the row and column at the upper left of the component. The leftmost column has address gridx=0 and the top row has address gridy=0. Use GridBagConstraints.RELATIVE (the default value) to specify that the component be placed just to the right of (for gridx) or just below (for gridy) the component that was added to the container just before this component was added. We recommend specifying the gridx and gridy values for each component rather than just using GridBagConstraints.RELATIVE; this tends to result in more predictable layouts.
gridwidth, gridheight	Specify the number of columns (for gridwidth) or rows (for gridheight) in the component's display area. These constraints specify the number of cells the component uses, <i>not</i> the

	<p>number of pixels it uses. The default value is 1.</p> <p>Use <code>GridBagConstraints.REMAINDER</code> to specify that the component be the last one in its row (for gridwidth) or column (for gridheight).</p> <p>Use <code>GridBagConstraints.RELATIVE</code> to specify that the component be the next to last one in its row (for gridwidth) or column (for gridheight).</p> <p>For more predictable layouts use gridwidth and gridheight for each component individually.</p> <p>Note: <code>GridBagLayout</code> does not allow components to span multiple rows unless the component is in the leftmost column or you have specified positive.gridx and.gridy values for the component.</p>
fill	Used when the component's display area is larger than the component's requested size to determine whether and how to resize the component. Valid values (defined as <code>GridBagConstraints</code> constants) include <code>NONE</code> (the default), <code>HORIZONTAL</code> (make the component wide enough to fill its display area horizontally, but do not change its height), <code>VERTICAL</code> (make the component tall enough to fill its display area vertically, but do not change its width), and <code>BOTH</code> (make the component fill its display area entirely).
ipadx, ipady	Specifies the internal padding: how much to add to the size of the component. The default value is zero. The width of the component will be at least its minimum width plus <code>ipadx*2</code> pixels, since the padding applies to both sides of the component. Similarly, the height of the component will be at least its minimum height plus <code>ipady*2</code> pixels.
insets	Specifies the external padding of the component -- the minimum amount of space between the component and the edges of its display area. The value is specified as an <code>Insets</code>

	object. By default, each component has no external padding.
anchor	<p>Used when the component is smaller than its display area to determine where (within the area) to place the component. Valid values (defined as GridBagConstraints constants) are CENTER (the default), PAGE_START, PAGE_END, LINE_START, LINE_END, FIRST_LINE_START, FIRST_LINE_END, LAST_LINE_END, and LAST_LINE_START.</p> <p>Version note: The PAGE_* and *LINE_* constants were introduced in 1.4. Previous releases require values named after points of the compass.</p>
weightx, weighty	<p>Weights are used to determine how to distribute space among columns (weightx) and among rows (weighty); this is important for specifying resizing behavior.</p> <p>Weights are specified with 0.0 and 1.0 as the extremes: Larger numbers indicate that the component's row or column should get more space. For each column, the weight is related to the highest weightx specified for a component within that column, with each multicolumn component's weight being split somehow between the columns the component is in. Similarly, each row's weight is related to the highest weighty specified for a component within that row. Extra space tends to go toward the rightmost column and bottom row.</p>

Custom Layouts

- **AbsoluteLayout**
 - **AbsoluteLayout** is not a Layout
 - Means absence of layout
 - Allows the developer to position and size the components
 - Gives greater freedom for placing components
 - Developer needs to spend time on deciding the custom look-and-feel.

Absolute layout allows the developer to choose a layout of his/her own choosing. In case the developer does not wish to use the pre-defined layouts available with Swing, then the developer has to follow the following steps

1. Set the container's layout manager to null by calling `setLayout(null)`.
2. Call the Component class's `setBounds()` method for each of the container's children.
3. Call the Component class's `repaint` method.

```
pane.setLayout (null);
JButton b1 = new JButton ("one");
pane.add (b1);
Insets insets = pane.getInsets ();
Dimension size = b1.getPreferredSize ();
b1.setBounds (25 + insets.left, 5 +
insets.top, size.width, size.height);
```



Best Practice

Layouts – Best fit scenarios

Need to display a component in as much space as it can get

BorderLayout

Need to display a few components in a compact row

FlowLayout

Need to display a few components of the same size in tabular format

GridLayout

You have a complex layout with many components

GridBagLayout

Copyrights of SEE Intertech Ltd. | Official Curriculum

Working with 2D Graphics

```
public class MyGraphics extends Frame{  
    public void paint(Graphics g){  
        g.drawString("Hello Java",200,300);  
    }  
    public static void main(String args[]){  
        MyGraphics mg=new MyGraphics();  
        mg.setSize (500,500);  
        mg.setVisible (true);  
    }  
}
```

Graphics class

All graphics are drawn relative to window. Window can be a frame, panel, canvas or an applet.

An output to window takes place through a GraphicsContext which is encapsulated by Graphics class.

Graphics class remembers settings associated with the window. E.g. foreground color, font etc...

Graphics

Used for painting basic shapes like lines and rectangles and for painting images. The actual implementation of these methods is operating system dependent and done with the help of peer classes.

Color

Represents a color with methods for converting between RGB and HSB values.

Font

Represents a font.

FontMetrics

Used for determining information about a font, such as height and character widths.

Image

Represents an image with methods for retrieving its dimensions.

Media Tracker

Used for preloading images.

The AWT supports a rich assortment of graphics methods. All graphics are drawn relative to a window. This can be the main window of an applet, a child window of an applet, or a stand-alone application window. The origin of each window is at the top-left corner and is 0, 0. Coordinates are specified in pixels. All output to a window takes place through a graphics context. A graphics context is encapsulated by the Graphics class and is obtained in two ways:

- It is passed to an applet when one of its various methods, such as `paint()` or `update()`, is called.
- It is returned by the `getGraphics()` method of Component.

For the subsequent of the examples in this chapter, we will be demonstrating graphics in the main applet window. However, the same techniques will apply to any other window.

The Graphics class defines a number of drawing functions. Each shape can be drawn edge-only or filled. Objects are drawn and filled in the currently selected graphics color, which is black by default. When a graphics object is drawn that exceeds the dimensions of the window, output is automatically clipped.

paint() Method

Most components do, in fact, do all drawing operations in their `paint()` methods. The `paint()` method should be smart enough to correctly redraw the component at any time, using data stored in instance variables that record the

state of the component. If in the middle of some other method you realize that the appearance of the component should change, then you should change the values of the instance variables and call the component's `repaint()` method.

This tells the system that it should redraw the component as soon as it gets a chance (by calling the component's `paint()` method).

Now, as it happens, the system does not actually call the `paint()` method directly. There is another method called `update()` which is the one actually called directly by the system. The built-in `update` procedure first fills in the entire component with a background color. Then it calls the `paint()` method. The `paint()` method draws on a rectangular area that has already been filled with the background color. Usually, this is the right thing to do. However, if the `paint` method itself fills in the entire rectangle, so that none of the background color is visible, then filling in the background was a wasted step. In that case, you can override `update()` to read simply:

```
public void update(Graphics g) {  
    paint(g);  
    // Don't fill with background color; just call paint.  
}
```

There is difference between the `paint()` and `repaint()` method is as follows:

The `paint` method causes the component and all its parts to be painted on the `Graphics` context that is passed in as parameter.

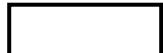
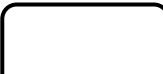
If the component is a light-weight component, then a call to the components `repaints` method which invokes the `paint` method as soon as possible. If the component is not a light-weight component, then calling `repaint` is same as calling the `update` method.

Graphics API

Some of the important methods of Graphics class:

Method	Description
<code>Graphics()</code>	Constructs a new <code>Graphics</code> object.
<code>void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)</code>	Draws the outline of a circular or elliptical arc covering the specified rectangle.
<code>void drawImage(Image img, int x, int y, Color bgcolor, ImageObserver observer)</code>	Draws as much of the specified image as is currently available.
<code>drawLine(int x1, int y1, int x2, int y2)</code>	Draws a line, using the current color, between the points (x_1, y_1) and (x_2, y_2) in this graphics context's coordinate system.
<code>drawOval(int x, int y, int width, int height)</code>	Draws the outline of an oval.
<code>drawPolyline(int[] xPoints, int[] yPoints, int nPoints)</code>	Draws a sequence of connected lines defined by arrays of x and y coordinates.
<code>drawRect(int x, int y, int width, int height)</code>	Draws the outline of the specified rectangle.
<code>drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)</code>	Draws an outlined round-cornered rectangle using this graphics context's current color.
<code>drawString(String str, int x, int y)</code>	Draws the text given by the specified string, using this graphics context's current font and color

Hence, Graphics class has different method to draw the different shapes like:

Line	drawLine(int x1, int y1, int x2, int y2)	
Oval	drawOval(int x, int y, int width, int height)	
Rectangle	drawRect(int x, int y, int width, int height)	
RoundRectangle	drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)	
Arc	void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)	

Creating an image

Following code snippet explains the simple methods of an image.

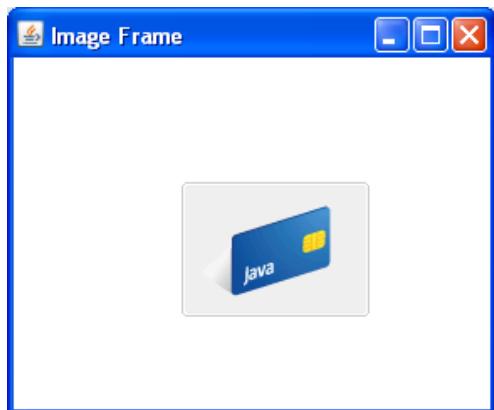
```
//write a java application to create an image
package com.seed.projet.imageProject;
import java.awt.*;
import java.awt.event.*;

public class CreateImage extends Frame{
    Image img;
    public static void main (String[] args){
        CreateImage ai = new CreateImage();
```

```
}

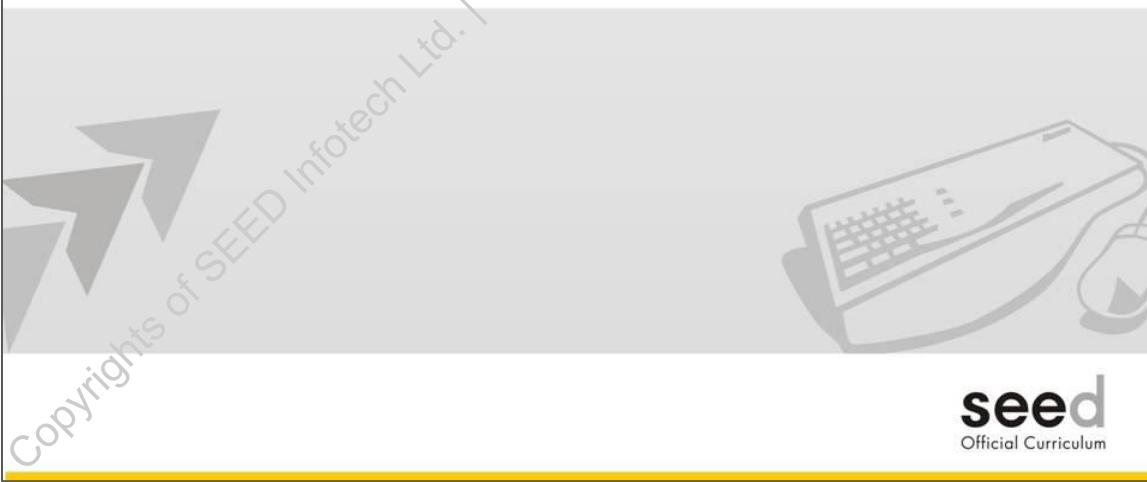
public CreateImage() {
super("Image Frame");
MediaTracker mt = new MediaTracker(this);
img =
Toolkit.getDefaultToolkit().getImage("362933.gif");
mt.addImage(img, 0);
setSize(400, 400);
setVisible(true);
addWindowListener(new WindowAdapter() {
public void windowClosing(WindowEvent we) {
dispose();
}
});
}
public void update(Graphics g) {
paint(g);
}

public void paint(Graphics g) {
if(img != null)
g.drawImage(img, 100, 100, this);
else
g.clearRect(0, 0, getSize().width, getSize().height);
}
}
```



Chapter - 3

Java Swing(Containers, MVC)



This chapter covers GUI applications using display controls like `JProgressBar`, `JToolTip`. Some general purpose controls like `JToolBar`, `JTabbedPane` and `JSplitPane`, specific containers `JDesktopPane` and `JLayeredPane`, some advance controls like `JTree`, `JTable`, `JSpinner` are also discussed .This topic also covers MVC in Swing.

Objectives

At the end of this chapter you will be able to:

- Use adapter classes for event handling.
- Use anonymous inner class for event handling.
- Use display controls like `JProgressBar`, `JToolTip` in an application.
- Use general purpose containers like `JToolBar`, `JTabbedPane`, and `JSplitPane`.
- Use Specific containers like `JDesktopPane`, `JLayeredPane` to build more sophisticated UI.
- Use advance UI controls like `JTree`, `JTable`, and `JSpinner`.
- Understand the concept of Model-View-Controller in Swing.

Adapter classes

- Certain listeners listen for more than one event.
 - E.g., `MouseListener` listens to mouse click, release, entry, exit, press .
- Application may want to handle only a subset of these events.
- Swing design forces to implement all methods.
 - As listeners are interfaces.

In certain cases, the user may be interested in having only a subset of activities associated with a component to be customized, while leaving the other activities untouched. As per the Swing architecture design, the listeners are constructed as interfaces. This forces a developer to provide coding for each and every method listed in the interface.

To reduce the burden on the developer the idea of adapter class is utilized in case of Swing as well. As per the Adapter concept, the interface must be implemented by a concrete class which provides default / empty implementation for the interface methods. The developer creates a component which extends from this concrete class and overrides the behavior that he/she wants customized.

This approach of programming ensures that all the methods in the interface are implemented, and the developer still has the freedom to provide implementation for the behavior of the application's choice.

The adapter classes provided by Swing are:

FocusAdapter	gaining or losing a focus
KeyAdapter	key press / release or key type
MouseAdapter	mouse clicked / pressed / released
MouseMotionAdapter	mouse moved / dragged
WindowAdapter	window minimized / maximized / closed

Display Controls

- Certain situations dictate that the information should only be displayed.
- User needs to show process progress.
 - JProgressBar
- User needs to provide component level help.
 - JToolTip

JProgressBar

It is used to display progress of an operation initiated by user or application. It is indication to user that the task is occurring, how long the task might take ,and how much work has already been done.



JProgressBar API

Following are the some important methods of JProgressBar:

Method	Description
JProgressBar() JProgressBar(int, int)	Create a horizontal progress bar. The no-argument constructor initializes the progress bar with a minimum and initial value of 0 and a maximum of 100. The

	constructor with two integer arguments specifies the minimum and maximum values.
JProgressBar(int) JProgressBar (int, int, int)	Create a progress bar with the specified orientation, which can be either JProgressBar.HORIZONTAL or JProgressBar.VERTICAL. The optional second and third arguments specify minimum and maximum values.
JProgressBar (BoundedRangeModel)	Create a horizontal progress bar with the specified range model.
void setValue(int) int getValue()	Set or get the current value of the progress bar. The value is constrained by the minimum and maximum values.
double getPercentComplete()	Get the percent complete for the progress bar.
void setMinimum(int) int getMinimum()	Set or get the minimum value of the progress bar.
void setMaximum(int) int getMaximum()	Set or get the maximum value of the progress bar.
void setModel(BoundedRangeModel) BoundedRangeModel getModel()	Set or get the model used by the progress bar. The model establishes the progress bar's constraints and values, so you can use it directly as an alternative to using the individual set/get methods listed above.
void setOrientation(int) int getOrientation()	Set or get whether the progress bar is vertical or horizontal. Acceptable values are JProgressBar.VERTICAL or

	JProgressBar.HORIZONTAL.
void setBorderPainted(boolean) boolean isBorderPainted()	Set or get whether the progress bar has a border.
void setStringPainted(boolean) boolean isStringPainted()	Set or get whether the progress bar displays a percent string. By default, the value of the percent string is the value returned by getPercentComplete formatted as a percent. You can set the string to be displayed with setString.
void setString(String) String getString()	Set or get the percent string.

Code Example

Following code snippet explains the simple methods of a JProgressBar:

```
package com.seed.demos;
import javax.swing.JProgressBar;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

public class Main {
    public static void main(String[] argv) throws Exception {
        int minimum = 0;
        int maximum = 100;
        JProgressBar progress = new JProgressBar(minimum, maximum);

        progress.addChangeListener(new ChangeListener() {
            public void stateChanged(ChangeEvent evt) {
                JProgressBar comp = (JProgressBar) evt.getSource();
                int value = comp.getValue();
                int min = comp.getMinimum();
```

```
        int max = comp.getMaximum();  
    }  
} );  
}  
}
```

JToolTip

It is used as a simple help that is displayed when mouse hovers over a component.



JToolTip API

Following are the some important methods of JToolTip:

Method	Description
JToolTip()	Creates a tool tip.
AccessibleContext	Gets the AccessibleContext associated with this JToolTip.
getAccessibleContext()	
JComponent getComponent()	Returns the component the tooltip applies to.
String getTipText()	Returns the text that is shown when the tool tip is displayed.
ToolTipUI getUI()	Returns the L&F object that renders this component
String getUIClassID()	Returns the name of the L&F class that renders this component.
String paramString()	Returns a string representation of this JToolTip.

void setComponent (JComponent c)	Specifies the component that the tooltip describes.
void setTipText (String tipText)	Sets the text to show when the tool tip is displayed.
void updateUI ()	Resets the UI property to a value from the current look and feel.

Code Example

Following code snippet explains the simple methods of a JToolTip:

```
package com.seed.demos;
import javax.swing.JButton;
import javax.swing.JFrame;
public class ToolTipDemo {
    public static void main(String args[]) {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JButton b = new JButton("Button");
        frame.add(b, "Center");
        b.setToolTipText("A button Component");
        frame.setSize(300, 200);
        frame.setVisible(true);
    }
}
```

General Purpose containers

- Used to contain components.
- Generally used to optimize space on UI.
- User wants to display commonly used Actions or controls.
 - JToolBar
- User wants to display multiple input areas.
 - JTabbedPane
- User wants two areas side-by-side.
 - JSplitPane

General purpose containers are used to contain components. They are used to optimized space on UI.

JToolBar

It is used to show pictorial representation of menu shortcuts. This makes user operation easier. JToolBar provides a component that is useful for displaying commonly used actions or controls.



JToolBar API

Following are the some important methods of JToolBar:

Method	Description
JToolBar()	Creates a tool bar. The optional int parameter lets you specify the orientation;
JToolBar(int)	

JToolBar(String) JToolBar(String, int)	the default is HORIZONTAL. The optional String parameter allows you to specify the title of the tool bar's window if it is dragged outside of its container.
Component add(Component)	Adds a component to the tool bar. You can associate a button with an Action using the setAction(Action) method defined by the AbstractButton.
void addSeparator()	Adds a separator to the end of the tool bar.
void setFloatable(boolean) boolean isFloatable()	The floatable property is true by default, and indicates that the user can drag the tool bar out into a separate window. To turn off tool bar dragging, use toolBar.setFloatable (false). Some types of look and feel might ignore this property.
void setRollover(boolean) boolean isRollover()	The rollover property is false by default. To make tool bar buttons be indicated visually when the user passes over them with the cursor, set this property to true. Some types of look and feel might ignore this property.

Code Example

Following code snippet explains the simple methods of a JToolBar :

```
package com.seed.demos;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JToolBar;

public class ToolBarApp {

    public static void main(String [] a) {
```

```

JFrame frame = new JFrame();
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JToolBar toolBar = new JToolBar("Can drag out");
toolBar.setFloatable(true);
toolBar.setRollover(true);

toolBar.add (new JButton ("New"));
toolBar.addSeparator ();
toolBar.add (new JButton ("Open"));
toolBar.addSeparator ();
toolBar.add (new JButton ("Anything Else"));

frame.add (toolBar, "North");

frame.setSize (500, 400);
frame.setVisible (true);
}
}

```

JTabbedPane

A component that lets the user switch between a group of components by clicking on a tab with a given title and/or icon.



JTabbedPane API

Following are the some important methods of JTabbedPane:

Method	Description
JTabbedPane ()	Creates an empty TabbedPane with a default tab placement of JTabbedPane.TOP.

<code>JTabbedPane(int tabPlacement)</code>	Creates an empty TabbedPane with the specified tab placement of either: <code>JTabbedPane.TOP</code> , <code>JTabbedPane.BOTTOM</code> , <code>JTabbedPane.LEFT</code> , or <code>JTabbedPane.RIGHT</code> .
<code>JTabbedPane(int tabPlacement, int tabLayoutPolicy)</code>	Creates an empty TabbedPane with the specified tab placement and tab layout policy.
<code>Component add(Component component)</code>	Adds a component with a tab title defaulting to the name of the component which is the result of calling <code>component.getName()</code> .
<code>Component add(Component component, int index)</code>	Adds a component at the specified tab index with a tab title defaulting to the name of the component.
<code>void add(Component component, Object constraints)</code>	Adds a component to the tabbed pane.
<code>void add(Component component, Object constraints, int index)</code>	Adds a component at the specified tab index.
<code>void addChangeListener(ChangeListener l)</code>	Adds a ChangeListener to this tabbedpane.
<code>void addTab(String title, Component component)</code>	Adds a component represented by a title and no icon.

void addTab(String title, Icon icon, Component component)	Adds a component represented by a title and/or icon, either of which can be null.
void addTab(String title, Icon icon, Component component, String tip)	Adds a component and tip represented by a title and/or icon, either of which can be null.
AccessibleContext getAccessibleContext()	Gets the AccessibleContext associated with this JTabbedPane.
SingleSelectionMode getModel()	Returns the model associated with this tabbedPane.
int selectedIndex()	Returns the currently selected index for this tabbedPane.
int getTabCount()	Returns the number of tabs in this tabbedPane.
String getTitleAt(int index)	Returns the tab title at index.
setMnemonicAt(int tabIndex, int mnemonic)	Sets the keyboard mnemonic for accessing the specified tab.

Code Example

Following code snippet explains the simple methods of a JTabbedPane

```
package com.seed.demos;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JTabbedPane;
```

```

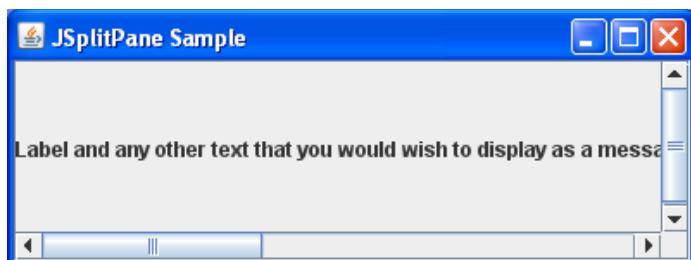
public class TabbedDemo extends JFrame
{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    TabbedDemo()
    {
        JButton b1=new JButton ("first");
        JButton b2=new JButton("second");
        JButton b3=new JButton ("third");
        JTabbedPane tb=new JTabbedPane ();
        tb.add ("First", b1);
        tb.add ("Second",b2);
        tb.add ("Third",b3);
        tb.createToolTip();
        getContentPane ().add (tb);
        setSize(100,100);
        setVisible(true);

    }
    public static void main(String a[])
    {
        new TabbedDemo();
    }
}

```

JSplitPane

JSplitPane is used to divide two (and only two) Components. The two Components are graphically divided based on the look and feel implementation, and the two Components can then be interactively resized by the user.



JSplitPane API

Following are the some important methods of JSplitPane

Method	Description
<pre>JSplitPane() JSplitPane(int newOrientation) JSplitPane(int newOrientation, boolean newContinuousLayout) JSplitPane(int newOrientation, boolean newContinuousLayout, Component newLeftComponent, Component newRightComponent)</pre>	Creates a new JSplitPane configured to arrange the child components side-by-side horizontally with no continuous layout, using two buttons for the components.
<pre>void addImpl(Component comp, Object constraints, int index)</pre>	Adds the specified component to this split pane.
<pre>AccessibleContext getAccessibleContext()</pre>	Gets the AccessibleContext associated with this JSplitPane.
<pre>Component getBottomComponent()</pre>	Returns the component below, or to the right of the divider.
<pre>int getOrientation()</pre>	Returns the orientation.
<pre>Void setOrientation(int orientation)</pre>	Sets the orientation, or how the splitter is divided.

Code Example

Following code snippet explains the simple methods of a JSplitPane

```
package com.seed.demos;
import java.awt.BorderLayout;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JSplitPane;
public class MovingJSplitPaneDivider {
    public static void main(String[] a) {
        JFrame horizontalFrame = new JFrame();

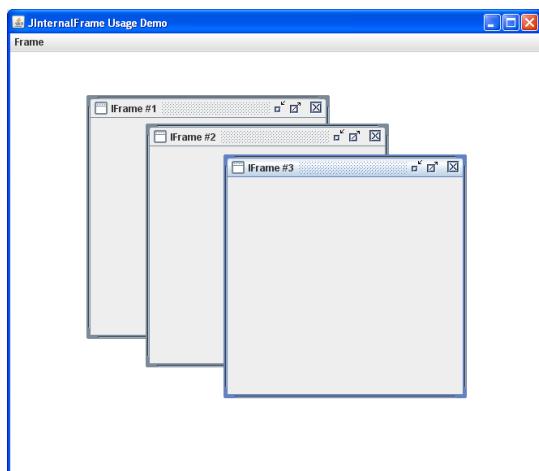
        horizontalFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JComponent topButton = new JButton("Left");
        JComponent bottomButton = new JButton("Right");
        final JSplitPane splitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT);
        splitPane.setTopComponent(topButton);
        splitPane.setBottomComponent(bottomButton);
        horizontalFrame.add(splitPane,
BorderLayout.CENTER);
        horizontalFrame.setSize(150, 150);
        horizontalFrame.setVisible(true);
        splitPane.setDividerLocation(0.5);
    }
}
```

Specific Containers

- As name suggests these containers are used for specific purposes.
- A window-within-window is needed.
 - JDesktopPane / JInternalFrame
- A view stacking is required .
 - Stack of images – JLayeredPane

Specific containers are used for specific purpose. For example JDesktopPane and JLayeredPane

JDesktopPane



These are the special purpose container used for specific functionality. It is commonly used for creating Multiple Document Interface (MDI) applications.

JDesktopPane API

Following are the some important methods of JDesktopPane

Method	Description
<code>JDesktopPane()</code>	Creates a new JDesktopPane.
<code>void addImpl(Component component, Object constraints, int index)</code>	Adds the specified component to this container at the specified index.
<code>AccessibleContext getAccessibleContext()</code>	Gets the AccessibleContext associated with this JDesktopPane.
<code>JInternalFrame[] getAllFrames()</code>	Returns all JInternalFrames currently displayed in the desktop.
<code>DesktopManager getDesktopManager()</code>	Returns the DesktopManger that handles desktop-specific UI actions.
<code>int getDragMode()</code>	Gets the current "dragging style" used by the desktop pane.
<code>JInternalFrame getSelectedFrame()</code>	Returns the currently active JInternalFrame in this JDesktopPane, or null if no JInternalFrame is currently active.
<code>String getUIClassID()</code>	Returns the name of the L&F class that renders this component.
<code>void remove(int index)</code>	Remove the indexed component from this pane.
<code>JInternalFrame selectFrame(boolean forward)</code>	Selects the next JInternalFrame in this desktop pane.
<code>void setDragMode(int dragMode)</code>	Sets the "dragging style" used by the desktop pane.

Code Example

Following code snippet explains the simple methods of a JDesktopPane:

```
package com.seed.demos;
import javax.swing.JInternalFrame;
import javax.swing.JDesktopPane;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JMenuBar;
import javax.swing.JFrame;
import java.awt.event.*;
import java.awt.*;
```

```
public class JDesktopFrameDemo extends JFrame {
    JDesktopPane jdpDesktop;
    static int openFrameCount = 0;
    public JDesktopFrameDemo () {
        super("JDesktopFrame Usage Demo");
        // Make the main window positioned as 50 pixels
from each edge of the
        // screen.
        int inset = 50;
        Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
        setBounds(inset, inset, screenSize.width -
inset * 2,
                  screenSize.height - inset * 2);
        // Add a Window Exit Listener
        addWindowListener(new WindowAdapter() {

            public void windowClosing(WindowEvent e) {
                System.exit (0);
            }
        });
        // Create and Set up the GUI.
        jdpDesktop = new JDesktopPane();
```

```
// A specialized layered pane to be used with
JInternalFrames
    createFrame(); // Create first window
    setContentPane(jdpDesktop);
    setJMenuBar(createMenuBar());
    // Make dragging faster by setting drag mode to
Outline

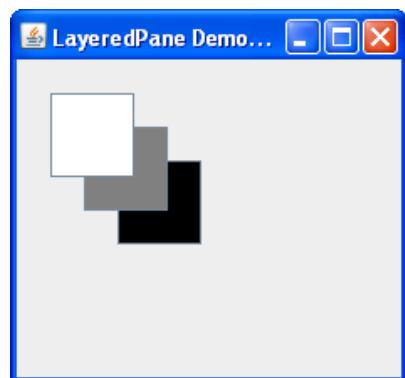
    jdpDesktop.putClientProperty("JDesktopPane.dragMode",
", "outline");
}
protected JMenuBar createMenuBar() {
    JMenuBar menuBar = new JMenuBar();
    JMenu menu = new JMenu("Frame");
    menu.setMnemonic(KeyEvent.VK_N);
    JMenuItem menuItem = new JMenuItem("New
IFrame");
    menuItem.setMnemonic(KeyEvent.VK_N);
    menuItem.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
{
        createFrame();
    }
});
menu.add(menuItem);
menuBar.add(menu);
return menuBar;
}
protected void createFrame() {
    MyInternalFrame frame = new MyInternalFrame();
    frame.setVisible(true);
    // Every JInternalFrame must be added to
content pane using JDesktopPane
    jdpDesktop.add(frame);
    try {
        frame.setSelected (true);
    }
}
```

```
        } catch (java.beans.PropertyVetoException e) {
    }
}

public static void main(String[] args) {
    JInternalFrameDemo frame = new
JInternalFrameDemo();
    frame.setVisible(true);
}
class MyInternalFrame extends JInternalFrame {
    static final int xPosition = 30, yPosition =
30;
    public MyInternalFrame() {
        super("IFrame #" + (++openFrameCount),
true, // resizable
                true, // closable
                true, // maximizable
                true); // iconifiable
        setSize (300, 300);
        // Set the window's location.
        setLocation (xPosition * openFrameCount,
yPosition
                * openFrameCount);
    }
}
}
```

JLayeredPane

JLayeredPane adds depth to a JFC/Swing container, allowing components to overlap each other when needed.



JLayeredPane API

Following are the some important methods of JLayeredPane

Method	Description
JLayeredPane ()	Create a layered pane.
JLayeredPane getLayeredPane ()	Get the automatic layered pane in an applet, dialog, frame, or internal frame.
void add(Component) void add(Component, Object) void add(Component, Object, int)	Add the specified component to the layered pane. The second argument, when present, is an Integer that indicates the layer. The third argument, when present, indicates the component's position within its layer. If you use the one-argument version of this method, the component is added to layer 0. If you use the one- or two-argument version of this method, the component is placed underneath all other components currently in the same layer.
void setLayer(Component, int) void setLayer(Component, int, int)	Change the component's layer. The second argument indicates the layer. The third argument, when present, indicates the component's position within its layer.
int getLayer(Component) int getLayer(JComponent)	Get the layer for the specified component.
int getComponentCountInLayer (int)	Get the number of components in the specified layer. The value returned by

	this method can be useful for computing position values.
Component [] getComponentsInLayer(int)	Get an array of all the components in the specified layer.
int highestLayer() int lowestLayer()	Compute the highest or lowest layer currently in use.
void setPosition(Component, int) int getPosition(Component)	Set or get the position for the specified component within its layer.
void moveToFront(Component) void moveToBack(Component)	Move the specified component to the front or back of its layer.

Code Example

Following code snippet explains the simple methods of a `JLayeredPane`

```
package com.seed.demos;
import java.awt.Color;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLayeredPane;

public class LayeredApp extends JFrame {
    public LayeredApp () {
        super("LayeredPane Demonstration");
        setSize(200, 150);
        setDefaultCloseOperation (EXIT_ON_CLOSE);
        JLayeredPane lp = getLayeredPane();
        JButton top = new JButton();
        top.setBackground(Color.white);
        top.setBounds(20, 20, 50, 50);
        JButton middle = new JButton();
        middle.setBackground(Color.gray);
        middle.setBounds(40, 40, 50, 50);
        JButton bottom = new JButton();
```

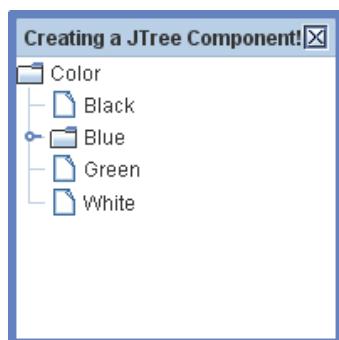
```
bottom.setBackground(Color.black);
bottom.setBounds(60, 60, 50, 50);
lp.add(middle, new Integer(2));
lp.add(top, new Integer(3));
lp.add(bottom, new Integer(1));
}
public static void main(String[] args) {
    LayeredApp sl = new LayeredApp ();
    sl.setSize(400,400);
    sl.setVisible(true);
}
```

Advanced Controls

- Apart from basic actions, there are certain activities which require the controls themselves to respond to user inputs.
- Special set of controls which hold data and react to user activity.
- User needs to display data in tabular form.
 - JTable
- User needs to display a directory structure.
 - JTree
- User needs to select a number or an object value from an ordered sequence.
 - JSpinner

JTree

It is commonly used to represent hierarchical data in a structured manner. The top node is known as root and all the nodes containing other child nodes become their parent. The nodes without a child is known as leaf.



JTree API

Following are the some important methods of JTree

Method	Description
JTree(TreeNode) JTree(TreeNode, boolean) JTree(TreeModel) JTree() JTree(Hashtable) JTree(Object[]) JTree(Vector)	<p>Create a tree. The TreeNode argument specifies the root node, to be managed by the default tree model. The TreeModel argument specifies the model that provides the data to the table. The no-argument version of this constructor is for use in builders; it creates a tree that contains some sample data. If you specify a Hashtable, array of objects, or Vector as an argument, then the argument is treated as a list of nodes under the root node (which is not displayed), and a model and tree nodes are constructed accordingly.</p> <p>The boolean argument, if present, specifies how the tree should determine whether a node should be displayed as a leaf. If the argument is false (the default), any node without children is displayed as a leaf. If the argument is true, a node is a leaf only if its getAllowsChildren method returns false.</p>
void setCellRenderer (TreeCellRenderer)	Set the renderer that draws each node.
void setEditable(boolean) void	The first method sets whether the user can edit tree nodes. By default,

<code>setCellEditor(TreeCellEditor)</code>	tree nodes are not editable. The second sets which customized editor to use.
<code>void setRootVisible(boolean)</code>	Set whether the tree shows the root node. The default value is false if the tree is created using one of the constructors that takes a data structure, and true otherwise.
<code>void setShowsRootHandles(boolean)</code>	Set whether the tree shows handles for its leftmost nodes, letting you expand and collapse the nodes. The default is false. If the tree does not show the root node, then you should invoke <code>setShowsRootHandles(true)</code> .
<code>void setDragEnabled(boolean)</code> <code>boolean getDragEnabled()</code>	Set or get the <code>dragEnabled</code> property, which must be true to enable drag handling on this component. The default value is false.
<code>void addTreeSelectionListener(TreeSelectionListener)</code>	Register a listener to detect when the a node is selected or deselected.
<code>void setSelectionModel(TreeSelectionModel)</code> <code>TreeSelectionModel getSelectionModel()</code>	Set or get the model used to control node selections. You can turn off node selection completely using <code>setSelectionModel(null)</code> .
<code>void setSelectionMode(int)</code> <code>int getSelectionMode()</code>	Set or get the selection mode. The value can be <code>CONTIGUOUS_TREE_SELECTION</code> , <code>DISCONTIGUOUS_TREE_SELECTION</code> , or <code>SINGLE_TREE_SELECTION</code> (all defined in <code>TreeSelectionModel</code>).

void expandPath(TreePath) void collapsePath(TreePath)	Expand or collapse the specified tree path.
void setScrollsOnExpand(boolean) boolean getScrollsOnExpand()	Set or get whether the tree attempts to scroll to show previous hidden nodes. The default value is true.
void setToggleClickCount(int) int getToggleClickCount()	Set or get the number of mouse clicks before a node will expand or close. The default is two.

Code Example

Following code snippet explains the simple methods of a JTree

```
package com.seed.demos;
import javax.swing.JFrame;
import javax.swing.JRootPane;
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
public class TreeApp{
    public static void main(String[] args) {
        JFrame frame = new JFrame("Creating a JTree Component!");
        DefaultMutableTreeNode parent = new DefaultMutableTreeNode("Color", true);
        DefaultMutableTreeNode black = new DefaultMutableTreeNode("Black");
        DefaultMutableTreeNode blue = new DefaultMutableTreeNode("Blue");
        DefaultMutableTreeNode nBlue = new DefaultMutableTreeNode("Navy Blue");
        DefaultMutableTreeNode dBlue = new DefaultMutableTreeNode("Dark Blue");
        DefaultMutableTreeNode green = new DefaultMutableTreeNode("Green");
```

```

DefaultMutableTreeNode white = new
DefaultMutableTreeNode("White");
parent.add(black);
parent.add(blue);
blue.add(nBlue);
blue.add(dBlue);
parent.add(green );
parent.add(white);
JTree tree = new JTree(parent);
frame.add(tree);

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setUndecorated(true);

frame.getRootPane().setWindowDecorationStyle(JRootPane.PLAIN_DIALOG);
frame.setSize(200,200);
frame.setVisible(true);
}
}
}

```

JTable

It is used mainly to show data in tabular format. It is useful for displaying reports. The `JTable` is used to display and edit regular two-dimensional tables of cells.

Name	Subject	Batch	Marks1	Marks2	Marks3
Rohit	java	Batch-1	50	50	50
Sumit	vb	Batch-2	60	50	50
Roma	java	Batch-3	50	80	50
Rupesh	vb	Batch-1	80	50	50
Rutu	java	Batch-1	90	80	50

JTable API

Following are the some important methods of JTable

Method	Description
<code>JTable()</code>	Constructs a default JTable that is initialized with a default data model, a default column model, and a default selection model.
<code>JTable(int numRows, int numColumns)</code>	Constructs a JTable with numRows and numColumns of empty cells using DefaultTableModel
<code>JTable(Object[][] rowData, Object[] columnNames)</code>	Constructs a JTable to display the values in the two dimensional array, rowData, with column names, columnNames
<code>JTable(TableModel dm)</code>	Constructs a JTable that is initialized with dm as the data model, a default column model, and a default selection model.
<code>void addColumn(TableColumn aColumn)</code>	Appends aColumn to the end of the array of columns held by this JTable's column model.
<code>void addColumnSelectionInterval(int index0, int index1)</code>	Adds the columns from index0 to index1, inclusive, to the current selection.
<code>void changeSelection(int rowIndex, int columnIndex, boolean toggle, boolean extend)</code>	Updates the selection models of the table, depending on the state of the two flags: toggle and extend.
<code>void columnAdded(TableColumnModelEvent</code>	Invoked when a column is added to the table column model.

e)	
void columnMoved(Table ColumnModelEvent e)	Invoked when a column is repositioned.
void columnRemoved (Tab leColumnModelEven t e)	Invoked when a column is removed from the table column model.
void columnSelectionCh anged(ListSelecti onEvent e)	Invoked when the selection model of the TableColumnModel is changed.
boolean editCellAt(int ro w, int column)	Programmatically starts editing the cell at row and column, if those indices are in the valid range, and the cell at those indices is editable.
TableCellEditor getCellEditor()	Returns the active cell editor, which is null if the table is not currently editing.
TableColumn getColumn (Object identifier)	Returns the TableColumn object for the column in the table whose identifier is equal to identifier, when compared using equals.
String getColumnName (int column)	Returns the name of the column appearing in the view at column position column.
Color getGridColor ()	Returns the color used to draw grid lines.
TableModel getModel ()	Returns the TableModel that provides the data displayed by this JTable.
int getRowCount ()	Returns the number of rows that can be

	shown in the <code>JTable</code> , given unlimited space.
<code>int getRowHeight ()</code>	Returns the height of a table row, in pixels.
<code>int getRowHeight (int row)</code>	Returns the height, in pixels, of the cells in row
<code>int getSelectedColumn ()</code>	Returns the index of the first selected column, -1 if no column is selected.
<code>int getSelectedColumn Count ()</code>	Returns the number of selected columns.
<code>int []getSelectedColu mns ()</code>	Returns the indices of all selected columns.
<code>int getSelectedRow ()</code>	Returns the index of the first selected row, -1 if no row is selected.
<code>int getSelectedRowCou nt ()</code>	Returns the number of selected rows.
<code>int [] getSelectedRows ()</code>	Returns the indices of all selected rows.
<code>JTableHeader getTableHeader ()</code>	Returns the <code>tableHeader</code> used by this <code>JTable</code> .
<code>boolean isCellEditable(in t row, int column)</code>	Returns true if the cell at <code>row</code> and <code>column</code> is editable.
<code>boolean isCellSelected(in t row,</code>	Returns true if the specified indices are in the valid range of rows and columns and the cell at the specified position is selected.

int column)	
isRowSelected(int row)	Returns true if the specified index is in the valid range of rows, and the row at that index is selected
void setTableHeader(JTableHeader tableHeader)	Sets the tableHeader working with this JTable to newHeader.
void setModel(TableModel dataModel)	Sets the data model for this table to newModel and registers with it for listener notifications from the new data model.

Code Example

Following code snippet explains the simple methods of a JTable

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

class DemoTable extends JFrame
{
    DemoTable ()
    {
        Container cp=getContentPane ();
        cp.setLayout (new BorderLayout ());

        String [] col={ "Name" , "Subject"
,"Batch","Marks1","Marks2","Marks3" };

        Object [][] data={
        { "Rohit" , "java", "Batch-1", "50", "50", "50" },
        { "Sumit", "vb", "Batch-2" , "60", "50", "50" },
        { "Roma" , "java", "Batch-3", "50", "80", "50" },
        { "Rupesh" , "vb", "Batch-1", "80", "50", "50" },
        { "Rutu" , "java", "Batch-1", "90", "80", "50" },
        };

    }
}
```

```

JTable tb=new JTable (data, col);

    int
v=ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS;
    int
h=ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS;

    JScrollPane jp=new JScrollPane (tb, v, h);
cp.add (jp, BorderLayout.CENTER);
}
public static void main(String args[])
{
    DemoTable d=new DemoTable ();
    d.setVisible (true);
    d.setSize (300,300);
}
}

```

JSpinner

It is used to display data having a range from minimum to maximum.



JSpinner API

Following are the some important methods of JSpinner:

Method	Description
JSpinner() JSpinner(SpinnerModel)	Creates a new JSpinner. The no-argument constructor creates a spinner with an integer SpinnerNumberModel with an initial value of 0 and no minimum or maximum limits. The optional

	parameter on the second constructor allows you to specify your own SpinnerModel.
void setValue(java.lang.Object) Object getValue()	Sets or gets the currently displayed element of the sequence.
Object getNextValue() Object getPreviousValue()	Gets the object in the sequence that comes before or after the object returned by the getValue method.
SpinnerModel getModel() void setModel(SpinnerModel)	Gets or sets the spinner's model.
JComponent getEditor() void setEditor(JComponent)	Gets or sets the spinner's editor, which is often an object of type JSpinner.DefaultEditor.
JSpinner.NumberEditor (JSpinner, String)	Creates a JSpinner.NumberEditor instance that displays and allows editing of the number value of the specified spinner. The string argument specifies the format to use to display the number.
JSpinner.DateEditor(JSpinner, String)	Creates a JSpinner.DateEditor instance that displays and allows editing of the Date value of the specified spinner. The string argument specifies the format to use to display the date.
JFormattedTextField getTextField()	Gets the formatted text field that provides the main GUI for this editor.
void setList(List)	Sets or gets the List that defines the

<code>List getList()</code>	sequence for this model.
<code>void setValue(Object)</code> <code>Date getDate()</code> <code>Object getValue()</code>	Sets or gets the current Date for this sequence.
<code>void setStart(Comparable)</code> <code>Comparable getStart()</code>	Sets or gets the first Date in this sequence. Use null to specify that the spinner has no lower limit.
<code>void setEnd(Comparable)</code> <code>Comparable getEnd()</code>	Sets or gets the last Date in this sequence. Use null to specify that the spinner has no upper limit.
<code>void setCalendarField(int)</code> <code>int getCalendarField()</code>	Sets or gets the size of the date value increment used by the <code>getNextValue</code> and <code>getPreviousValue</code> methods. This property is not used when the user explicitly increases or decreases the value; instead, the selected part of the formatted text field is incremented or decremented. The specified parameter must be one of the following constants, defined in <code>Calendar</code> : ERA, YEAR, MONTH, WEEK_OF_YEAR, WEEK_OF_MONTH, DAY_OF_MONTH, DAY_OF_YEAR, DAY_OF_WEEK, DAY_OF_WEEK_IN_MONTH, AM_PM, HOUR_OF_DAY, MINUTE, SECOND, MILLISECOND.
<code>void setValue(Object)</code> <code>Number getNumber()</code>	Sets or gets the current value for this sequence.
<code>void setMaximum(Comparable)</code> <code>Comparable getMaximum()</code>	Sets or gets the upper bound for numbers in this sequence. If the

	maximum is null, there is no upper bound.
void setMinimum(Comparable) Comparable getMinimum()	Sets or gets the lower bound for numbers in this sequence. If the minimum is null, there is no lower bound.
void setStepSize(Number) Number getStepSize()	Sets or gets the increment used by getNextValue and getPreviousValue methods.

Code Example

Following code snippet explains the simple methods of a JSpinner:

```
package com.seed.demos;

import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSpinner;
import javax.swing.SpinnerDateModel;
import javax.swing.SpinnerModel;

public class SwingApp
{
    public static void main(String args[])
    {
        JFrame frame = new JFrame("JSpinner Sample");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        SpinnerModel model1 = new SpinnerDateModel();
        JSpinner spinner1 = new JSpinner(model1);
        JLabel label1 = new JLabel("Dates/Date");
        JPanel panel1 = new JPanel(new BorderLayout());
        panel1.add(label1, BorderLayout.WEST);
        panel1.add(spinner1, BorderLayout.CENTER);
```

```
frame.add(panel1, BorderLayout.CENTER);  
frame.setSize(200, 90);  
frame.setVisible(true);  
}  
}
```

Copyrights of SEED Infotech Ltd. | Official Curriculum

MVC in Swing

- Data controls normally reflect data in GUI format
- Data holding code extracted out of component – Model
- Data holding code can pick data from any data source – improved flexibility
- Also called as Separable Model Architecture

To build Swing based applications there are certain goals are there to achieve this required ‘Layard Application’.

- De-couple business logic from the presentation. Change in business logic layer does not affect the presentation layer and vice-versa.
- Cross –platform consistency and easier maintenance.
- Multiple look-and-feel to single application so that end-user will not depend on only single view or look.

To fulfill these requirements swing application is routed to MVC architecture.

M = Model

V = View

C = Controller

Model

Model focuses on creation of business objects. It represents business data as an application.

So in short, a model represents the data for the application.

Responsibilities of Model are:

- Performing DB queries
- Calculating the business process
- Processing Orders.

View

View display information according to end-user requirement. It also shows the results of business logic which is calculated in a model. In short, the view that is the visual representation of that data.

View is not concerned with how the information was obtained, or from where it was obtained.

Controller

Controller is the flow of application. It is basically used for decisions.

It is responsible for making decisions among multiple presentations. In short, a controller that takes user input on the view and translates that to changes in the model.

MVC in Swing

In Swing, several components can be used with objects that implement corresponding models. For example, Swing's `JTree` component can be used with an object that implements the `TreeModel` interface. Similarly, a `JTable` can be used with a `TableModel`, and a `JList` can be used with a `ListModel`. In this case model interfaces use only accessors methods. This model interfaces perform no operations to manipulate the structure of the data rather no methods to insert or remove data items, or to change the order of data in the model. To implement MVC in swing let's take an example of `JTree`.

`JComboBox` requires its model to tell it what text to display as a choice and how many choices exist. `JSpinner` requires its model to tell it what text to display, and also what the previous and next choices are. `JList` also requires its model to tell it what text to display as a choice and how many choices exist. `JTable`

requires much more: It requires the model to tell it how many columns and rows exist, the column names, the class of each column, and what text to display in each cell. JTree requires its model to tell it the root node, the parents, and the children for the entire tree.

In JComboBox when you want to add items to the data by calling method addItem(). Suppose in an application there are 25 choices, and they are continuously changing every time you have to call addItem(). Instead of JComboBox contains a method call setModel() that accepts an instance of the ComboBoxModel class. In that scenario use setModel() method instead of addItem() method to create the data in a JComboBox.

Code Example

Following code snippet explains the How to implements JComboBox without model.

```
import javax.swing.JComboBox;
import javax.swing.JFrame;

public class AddingItemToComboBox {

    public static void main(String[] a) {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    JComboBox jComboBox1 = new JComboBox();
    jComboBox1.addItem("a");
    jComboBox1.addItem ("b");
    Object cmboitem = jComboBox1.getSelectedItem ();
    System.out.println (cmboitem);
    frame.add (jComboBox1);
    frame.setSize (300, 200);
    frame.setVisible (true);
}

}
```

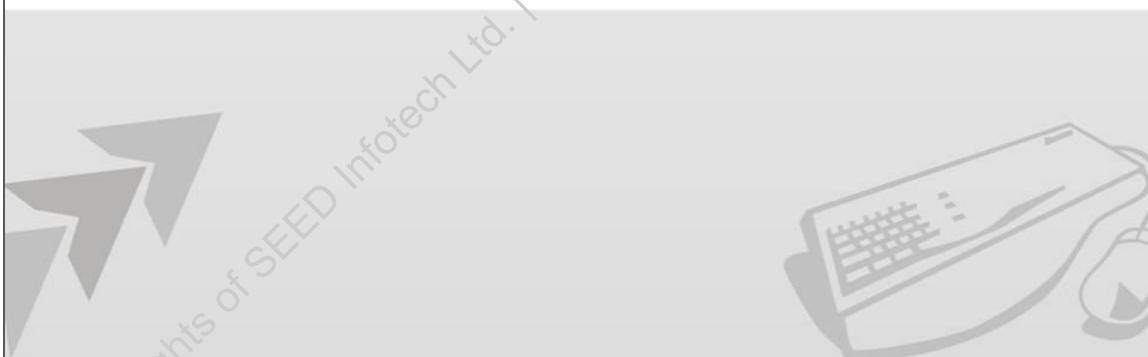
Following code snippet explains the how to implements model in JComboBox

Suppose you have an ArrayList with the alphabet as its data ("A", "B", "C", etc.)

```
MyComboModel model = new MyComboModel (alphaList);
//Assume myComboBox is an object of JComboBox.
myComboBox.setModel (model);
public class MyComboModel implements ComboBoxModel
{
    private List data = new ArrayList ();
    private int selected = 0;
    public MyComboModel(List list)
    {
        data = list;
    }
    public void setSelectedItem (Object o)
    {
        selected = data.indexOf (o);
    }
    public Object getSelectedItem()
    {
        return data.get (selected);
    }
    public int getSize ()
    {
        return data.size ();
    }
    public Object getElementAt (int i)
    {
        return data.get (i);
    }
}
```

Chapter 4

Java Multithreading Basics



This chapter covers multithreading concept and also how to write multithreaded applications.

Objectives

At the end of this chapter you will be able to :

- Differentiate between multi-processing and multi-threading.
- Identify need for a thread.
- Define a thread.
- Create thread using Thread class.
- Create a thread using Runnable interface.
- Describe thread life cycle.
- List steps for constructing multi-threaded application.
- Describe effect of Thread priorities on thread behavior.
- Implement Lowering thread priority using yield for cooperating thread.
- Implement combining two threads using join () .

Demystifying Multi

- Multitasking [ability to do more than one task at the same time]
- Multiprocessing
- Running more than one process concurrently.
 - WinAmp – listening songs
 - WinWord – Creating a document
- Multithreading
- Running more than one threads concurrently.
 - Winword – creating a document
 - Spellcheck – spell check for the same document

Ability to do more than one task at the same time is called multitasking. The term is equally applicable to humans and computers. Though user may feel he/she is keeping the computer busy for its task, for most of the time CPU is idle. To use the CPU more effectively, it is a good idea to execute multiple applications at the same time. For example, a user might be working with a document in an editing tool and a media player might be playing a song in the background simultaneously. Modern application platforms like Java, .NET provide support for such concurrent programming through programming languages and various libraries.

On a single CPU computer, concurrency can be achieved through two different basic units of execution: Process and Thread.

Multiprocessing

Process is self-contained execution environment synonymous to applications. It has its own resources like memory, I/O, registers etc. For example Java Virtual Machine (JVM), Microsoft Word runs as a process.

Multithreading

Thread is said to be a path of execution. It may be part of the application code. All the threads share resources of a process like memory. Hence, creating new thread takes less resource than creating a new process. Sometimes thread is referred as lightweight process.



Interview Tip

Focus on the difference between multiprocessing, multitasking and multithreading.

Multiprocessing – Context switch

- When OS switches from process to process, data needs to be stored temporarily.
- Data to be stored – Context
 - Instruction set
 - Current instruction
 - Current data structures in use
- Processes do not share same data, hence context switch is resource heavy.
- Scheduling of process is controlled by OS.
- Two ways
 - Pre-emptive
 - Non Pre-emptive(Cooperative)

Multitasking can be done even on the systems which have only one processor. Operating system takes each process request from the user and loads it into the memory. It keeps on switching between each process one after the other. This switching takes place very fast and creates an illusion of multiple processes running concurrently. Every process in execution has its own set of data structures known as ‘context’ and the switching between processes is aptly called as ‘context switch’. This context which is specific to a process needs to be preserved during the switching process. Context stores current status of the execution of a process including memory address space, registers etc. Operating system uses it to resume the execution while switching back to the same process. Thus, the process of context switch is resource intensive.

Multitasking on a single processor can be achieved in two different ways:

Pre-emptive

On a single CPU (typically referred as core) machine, illusion of programs running concurrently is created by the operating system with the help of a technique called time slicing. In ‘time slicing’, each unit of execution (thread or process) is allowed

to use the processor for a pre-decided fixed quantum of time called ‘time slice’. When the given quantum of time is complete, the operating system scheduler can interrupt (forcefully stop) and suspend (“swap out”) the currently running unit (thread or process) and start or resume (“swap-in”) another task. This is called pre-emptive multitasking where no process or thread can completely own the processor beyond a time slice.

Non pre-emptive

When the process or thread is allowed to run its course completely before letting control of the processor to another process is known as non pre-emptive multitasking. This is sometimes called as cooperative multitasking. In this type of multitasking, one process may take over the processor and not allow any other application to execute. It is typically used in case of batch processing.

Thread

- Not all modules are dependent on earlier modules.
- While the system responds to a module's request, CPU remains idle.
- This idle time can be utilised by queuing up other tasks of the process with CPU.
- Such independent intra-process executions are called as threads.

Beginning from the initial startup code, the program execution can be thought of as an execution flow. There might be multiple execution flows in a process. Putting it simply, there might be multiple modules in an application, which are not really dependent on each other but are part of the process for a bigger goal.

For example, consider the scenario

Task 1 – Initialization of resources

Task 2 – Accepting inputs from UI

Task 3 – Setting up database connectivity

Task 4 – Firing a query to retrieve data

Task 5 – setting up remote connectivity

Task 6 – Encapsulating the retrieved data for transfer

Task 7 – Transferring data to remote machine

Task 8 – Accepting results from the remote machine

Task 9 – presenting the result to the UI

In this scenario, the Tasks 2 and 5 are not really related to each other. It is not necessary for the Tasks 2 and 3 to complete, for Task 5 to begin execution. In classic programming practice, because Tasks 5 sequentially at number 5 it will have to wait for its turn to come before starting execution. On closer inspection it will become clear that even though remote connectivity has got nothing to do with databases, it still has to wait even though CPU may be idle.

To overcome this problem, we can think of splitting the entire process execution into two separate flows.

1-2-3-4-6-7-8-9 and 1-5-7-8-9

Now it is observed that the process 5 and process 2-3-4-6 are actually independent of each other can execute on their own. In this case these two are called independent threads which help in completing the intra-process execution of tasks.

Thread

- A thread is a lightweight, smallest unit of processing.
- A process is typically made up of multiple threads.
 - Each thread has its own stack (runtime stack) to handle its data.
 - A thread shares the memory space with other threads of the same process.
- When to use multithreading?
 - Performing operations that take a large amount of time
 - Prioritization of tasks
 - Application has to wait for some event to occur

As discussed earlier, a thread is nothing but a sub-division of the process under execution. Typically a thread will be the smallest atomic unit of the process execution which can be isolated without impacting the process execution. In other words, every sub-task which can run in isolation from other sub-tasks is a thread. Compared to a process, definitely a thread is going to be lightweight, as it is involved with less amount of execution and will need fewer amounts of resources.

The local variables declared within a thread need to be protected from malicious access by other threads/processes, hence the thread will have its own memory space allocated to it. This memory space is managed by the VM inside the shared memory space of the process itself, but it's a local stack private to each independent thread, it is also known as runtime stacks.

If there are any global data structures associated with the process execution, in Java's case the instance variables of a class, these are shared by all the threads and they can access and manipulate the instance variable with impunity. This leads to its own set of unique problems, which we have to deal with in some ways.

As in case of multiprocessing, the CPU is going to switch between executions of individual threads one after the other. Similar to the problem we have faced earlier, the OS will have to store the context data temporarily.

The context switching between threads is lightweight as compared the context switching for processes, for obvious reasons.

The execution schedule of the independent threads in a process is the prerogative of the OS; hence we do not have control over the sequence in which the threads are going to execute once they are started. One thing can be guaranteed though; the CPU idle time will be reduced to a great extent as depicted with the earlier example.

Now when the Task T2 is waiting for some response to come from the I/O call, the idle CPU is given the Task 3 to perform as there is no relation between T2 and T3. The CPU will keep on switching between T2 and T3 till the time both the Tasks / Threads complete their execution. There is a possibility that the Task T3 may finish even before Task T2 completes its execution, which will lead to increased performance for the application.

Creating a Thread

- In Java Thread can be created two ways
 1. Using Thread class
 - Create a user-defined class which will have `java.lang.Thread` as super class
 - `public class HelloThread extends Thread`
 2. Using Runnable interface
 - Create a user-defined class which will implement `java.lang.Runnable` interface
 - `public class HelloRunnable implements Runnable`

So far, conceptual details of threading and how it is going to help increase the performance has been discussed. How threads can be created in a Java class has been discussed in the following paragraphs.

As mentioned, threading can be enabled in the application two ways, either by sub-classing the `java.lang.Thread` class or by implementing `java.lang.Runnable` interface.

The advantage of using the `Runnable` interface is that the Java class is now free to extend from any other class if required. `Runnable` has one method `run()` in its definition, which has to be overridden regardless of whether `Thread` is extended or `Runnable` is implemented. The instruction set belonging to that specific thread will be coded into the `run()` method of the specific class.

```
class SimpleThread extends Thread
{
    String name;
    SimpleThread (String name) {
        super(name);
        this.name=name;
    }
    public void run ()
    {
        for (int i = 0;i < 10; i++)
        {
            System.out.println (i + " " +
Thread.currentThread ().getName ());
        }
        System.out.println ("DONE! ");
    }
}
class ThreadsTest
{
    public static void main (String [] args)
    {
        new SimpleThread ("seed").start();
        new SimpleThread ("Infotech").start ();
    }
}
```

Creating a Thread-using Runnable interface

```
class SimpleThread implements Runnable
{
    public void run()
    {
        for (int i = 0; i < 10; i++)
        {
            System.out.println(i + " " +
                Thread.currentThread().getName());
        }
        System.out.println("DONE! ");
    }
}
```

When a Thread application is written that class is extended from Thread **class**. Suppose, the application is a swing based application and is multithreaded, Can the Thread class be extended to Swing based application? The answer is “No”, because of multiple inheritance. Java can extend only one class. Solution is interface called Runnable Interface. Implementing Runnable gives the ability to treat the new class as a Runnable object.

Linking back to the example seen earlier, the entire problem statement can be divided into three distinct parts.

1. The business logic which is collection of individual modules
2. The code which is responsible for data related activities (2, 3, 4, 6)
3. The code which is responsible for remote connectivity (5)

This code be written as

```
public class BusinessLogic
{
    public static void main(String[] args)
    {
        //Task1
        Runnable t1 = new DataThread ();
        Runnable t2 = new RemoteThread ();
        //start t1
        //start t2
        //wait for t1 to complete
        //wait for t2 to complete
        //Task 7
        //Task 8
        //Task 9
    }
} // end of class

public class DataThread implements Runnable
{
    public void run ()
    {
        //Task 2
        //Task 3
        //Task 4
        //Task 6
    }
}

public class RemoteThread implements Runnable
{
    public void run()
    {
        //Task 5
    }
}
```

There are two interesting things to be noted here

1. The main method execution is being treated as if it is a thread. Every Java class execution has two threads running by default. First the main method execution, which is the parent thread for all other threads being created, and second the garbage collector, which runs in the background and is called as the daemon thread.
2. Interesting thing is that the Task 7 is actually dependent on both the individual threads completing successfully, hence before starting the execution of task 7, the main thread has to suspend the execution and wait for both the threads to finish. How to achieve this will be discussed a little later.

Thread Interruption and InterruptedException

- An interrupt is an indication to a thread that it should stop .
- Thread is interrupted by calling `interrupt()` on Thread Object.
- The thread is interrupted, either before or during the activity
 - throws `InterruptedException`.

The Thread Interruption is a mechanism where thread is waiting (or sleeping) can be made to **prematurely stop waiting**.

The `InterruptedException` is thrown when a thread is waiting, sleeping, or otherwise occupied, and the thread is interrupted, either before or during the activity. Occasionally a method may wish to test whether the current thread has been interrupted, and if so, to immediately throw this exception. The following code can be used to achieve this effect:

```
if(Thread.interrupted ()) // Clears interrupted  
status!  
    throw new InterruptedException ();
```

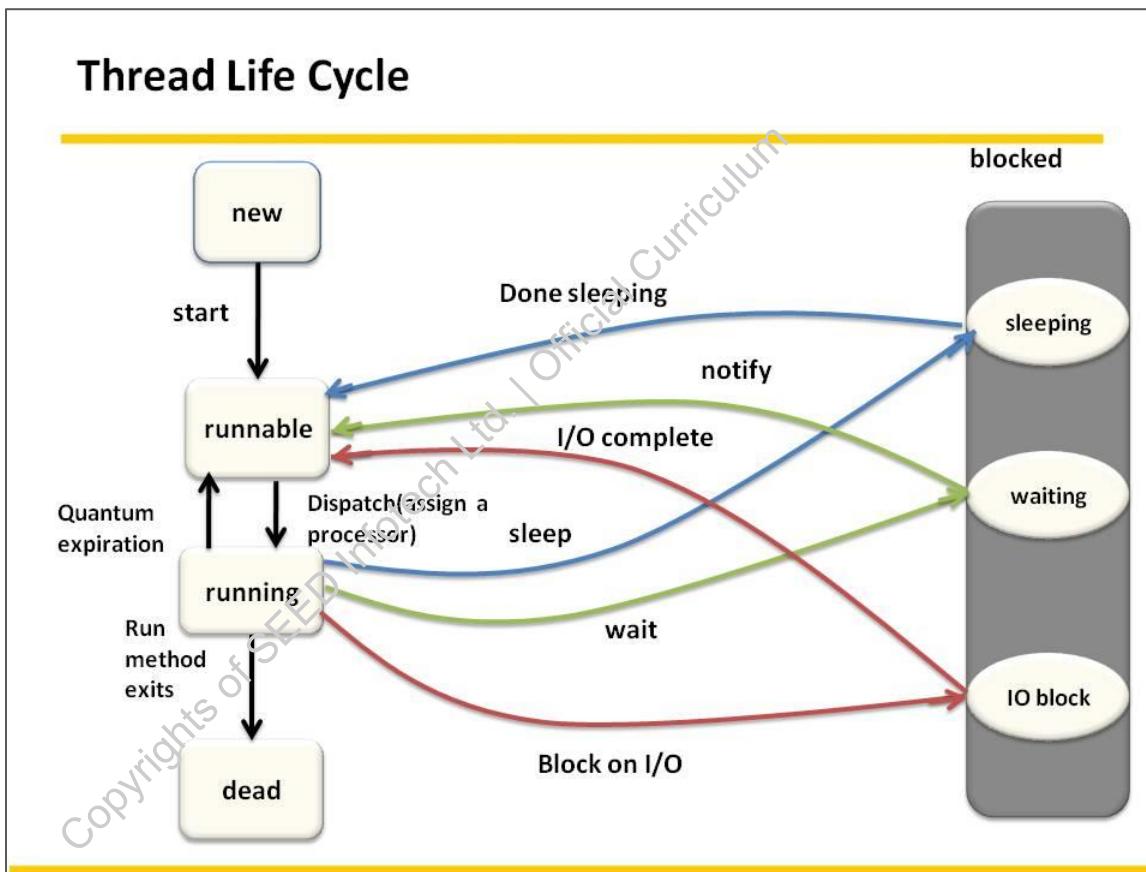
Initially, there is method available called `Thread.stop()` this method stops a running thread. This method is deprecated and unsafe.

The example of InterruptedException is-

```
public void run()
{
    while(true)
    {
        //some code here
        try
        {
            Thread.sleep(10);
        }
        catch(InterruptedException e)
        {
            e.printStackTrace();
        }
    }
}
```

The thread is in waiting, sleeping or paused for a particular time period and another thread

Interrupts it using interrupt method in Thread class. So, Thread throws an exception that exception is called InterruptedException.



Thread is a path of execution in a running application. Each process has at least one thread called the primary thread. It is the entry point of any application. It is the `main()` method in Java. There can be multiple threads in n application.

A thread has its own life cycle. Each thread exists in one of the states as shown in the diagram.

New Threads

When a thread is created using `new` operator, it is not running. This means that it is in `new` state. This state also called as ‘Born State’.

Runnable Thread

Once you invoke the `start()` method, the thread is in `runnable` state. A `runnable` thread may not yet be `running`. The priority of execution of a thread is decided only by the operating system.

Once a thread is `running` it is in `running` state `run()` method is invoked on the thread.

Preemptive scheduling systems give each runnable thread a slice of time to perform its task. When that slice of time is exhausted, the operating system preempts the thread and gives another thread an opportunity to work.

Blocked

Thread enters blocked state when one of the following events occurs:

- Someone calls `sleep()` method of the thread.
- The thread calls `wait()` method.
- The thread calls an operation that is blocking on input / output.
A thread tries to acquire a lock that is currently held by another thread.

Preemptive scheduling systems give each runnable thread a slice of time to perform its task. When that slice of time is exhausted, the operating system preempts the thread and gives another thread an opportunity to work.

A thread moves out of blocked state to runnable state in these scenarios:

- If the thread is put to sleep, the specified sleeping time elapses.
- If the thread is blocked by calling `wait()` method of the Object, the object's `notify()` or `notifyAll()` method is called.
- If the thread is blocked on I/O, the specified I/O operation completes.
- If a thread is waiting for a lock that was owned by another thread, then the other thread must relinquish ownership of the lock.

Dead Threads

A thread is dead when

- It dies a natural death because the `run()` method exits.
- `Stop` method is deprecated in jdk 5, so it is no longer advisable to use `stop()` to kill a thread.



The suspend method is deprecated since jdk 5 because suspend is inherently deadlock-prone. If the target thread holds a lock on the monitor protecting a critical system resource when it is suspended, no thread can access this resource until the target thread is resumed. If the thread that would resume the target thread attempts to lock this monitor prior to calling resume, deadlock results.

MultiThreading - An Algorithm

1. Break up an application into individual modules.
2. List execution flow of application.
3. Identify modules blocking CPU.
4. Identify modules independent of each other.
5. Build alternate execution flows.
6. Create threads for each execution flow.
7. Encapsulate business logic into separate threads.
8. Start threads.

Case Example

User enters some string on the UI (a query), which is executed against the database. The query returns some data that is needed by some other machine. The second machine is contacted and the data is transferred. The second machine performs its own operations on the data and returns back some message. This message needs to be displayed on the User Interface for further processing.

Step 1 - Identifying individual modules

- Task 1 – Initialization of resources
- Task 2 – Accepting inputs from UI
- Task 3 – Setting up database connectivity
- Task 4 – Firing a query to retrieve data
- Task 5 – setting up remote connectivity
- Task 6 – Encapsulating the retrieved data for transfer
- Task 7 – Transferring data to remote machine
- Task 8 – Accepting results from the remote machine

- Task 9 – presenting the result to the UI

Step 2 - Execution flow of the application

As we are considering normal execution flows, each module will be executed after the successful execution of the earlier module, resulting in the sequence 1-2-3-4-5-6-7-8-9.

Step 3 - Identify modules blocking CPU usage

In this scenario, the process for establishing connectivity with the database is time consuming, and it is entirely dependent on the database availability, connection availability and host of other non-programming related factors. The success of the process is also dependent not on the program code but on external components. The entire process might be time consuming, and the CPU executing the code has no option but to wait for some signal/acknowledgement/error message to come back before it can process further. This is called as CPU blockage.

Similarly, establishing connection to the second machine is also dependent on external components and is time consuming, leading to CPU blockage.

To reduce this CPU blockage, alternative processes are identified which can execute while the CPU is idle, but to do that modules that are not dependent on the successful completion of current execution will have to be identified.

Step 4 - Identify modules independent of each other

In the case study, tasks 2-3-4, Input acceptance-Database connectivity-Data Retrieval are dependent, but Task 5, Remote connectivity has no connection with these tasks and has to wait while process 2 to 4 finish their execution.

As Task 5 has no dependency on the earlier tasks, it is a prime candidate for execution when the CPU is idle and waiting for external activities to be complete.

For better utilization of the CPU, such independent execution flows need to be identified.

Step 5 - Build alternate execution flows

Task Number	Dependent on Task	Task Dependent on
1	-	2,3,4,5,6,7,8,9
2	1	3,4,6,7,8,9
3	2	4,6,7,8,9
4	3	6,7,8,9
5	1	7,8,9
6	4	7,8,9
7	5,6	8,9
8	7	9
9	8	-

Based on the above table these can be two independent execution flows:

1-2-3-4-6-7-8-9

1-5-7-8-9

In this case the tasks 1 and 7 to 9 are common to both flows, these will be executed as the main thread and remaining tasks will be split into two threads:

1. 2-3-4-6
2. 5

The combined execution of these modules will have to be encapsulated as individual threads, so that they can be executed as a single unit.

One observation worth noting here is that the task 7 is actually dependent on both execution flows executing. Data Transfer cannot happen if the first thread (Data Thread) has not finished execution, neither will it happen if second thread (Connectivity Thread) is not complete.

Hence, task 7 cannot start unless and until the two threads have completed successfully.

Step 6 - Create threads for individual flows

Now that the independent execution flows have been identified, it is time to convert them into actual implementation objects. The conversion of these flows into actual Thread objects can be handled in three broad ways.

1. The application is split into $1 + \langle\text{no. of flows}\rangle$ parts. The common tasks in the flow are executed through the main thread, and each flow is identified with a separate executable component, that is a class, having threading capability.
2. Extending Thread class.
3. Implementing Runnable interface.

The entire application can be built using a single class having threading capabilities and in the `run()` method of the class, differentiate between different threads by using the `currentThread` method of the `Thread` class.

To reduce the maintenance and readability issues it is recommended that each flow should be implemented using separate components.

Step 7 - Invoke independent modules

```
public class DataThread extends Thread
{
    // class that will take care of data based
activities
    public void run ()
    {
        //Task 2
        //Task 3
        //Task 4
        //Task 6
    }
}
public class RemoteThread extends Thread
{
    // class that will take care of remote
connectivity
    public void run()
    {
```

```
    //Task 5  
}  
}
```

Step 8 - Start the threads

```
public class BusinessLogic  
{  
    public static void main (String [] args) {  
        //start the main thread  
        //Task1  
        Thread t1 = new DataThread (); //start first  
execution flow  
        Thread t2 = new RemoteThread (); //start second  
execution flow  
        t1.start ();  
        t2.start ();  
        //wait for t1 to complete  
        //wait for first flow to complete  
        //wait for t2 to complete  
        //wait for second flow to complete  
  
        //Task 7  
        //Task 8  
        //Task 9  
    }  
} // end of class
```

In this example, when instructed to start threads t1 and t2, the CPU will alternate between both the threads and the process execution will be carried out for both independent execution flows.

Thread Priority

- Used by thread scheduler to decide when each thread should be allowed to run
- Threads of equal priority compete equally for CPU.
- Thread scheduler picks up the highest-priority thread that is currently runnable.
- Priority levels range from 1 (MIN_PRIORITY) to 10 (MAX_PRIORITY)
- 5 (NORM_PRIORITY) is the default priority level

Every thread in Java has a priority. By default, a thread inherits the priority of its parent thread can increase or decrease. The priority of any thread with the `setPriority()` method programmatically. Priority can be set to any value between `MIN_PRIORITY(1)` and `MAX_PRIORITY(10)`, `NORM_PRIORITY` is defined as 5.

Whenever the thread-scheduler has a chance to pick a new thread, it picks the highest priority thread that is currently Runnable. The highest priority Runnable thread keeps running until it either:

- Yields by calling the `yield` method,
- Ceases to be Runnable (either by dying or by entering the blocked state), or
- Is replaced by a higher-priority thread that has become Runnable (because it has slept long enough, because its I/O operation is complete or because someone called `notify`).

In such cases the scheduler looks into the threads currently in the Runnable state and picks the highest priority thread amongst all.

Field	Description
static int MAX_PRIORITY 10	The maximum priority that a thread can have.
static int MIN_PRIORITY 1	The minimum priority that a thread can have.
static int NORM_PRIORITY 5	The default priority that is assigned to a thread.

Priority API

Following are the some important methods of Thread class:

Method	Description
void setPriority(int newPriority)	Changes the priority of this thread.
int getPriority()	Returns this thread's priority.

Code Example

Following code snippet explains the simple methods of a Thread class:

```
package com.seed.threadProject;
public class PriorityClass extends Thread{
    public static void main(String[] args) {
        PriorityClass thread1 = new PriorityClass("thread 1");
        thread1.setPriority(10);
        PriorityClass thread2 = new PriorityClass("thread 2");
        thread2.setPriority(1);
        thread2.start();
        thread1.start();
    }
}
```

Thread methods

- start()
- sleep(long millisecond)
- join()
- yield()
- interrupt()
- interrupted()
- isAlive()
- currentThread()

Here are the some of the important methods of the Thread class.

sleep()

The signature of method is:

```
public static void sleep (long millis) throws  
InterruptedException
```

Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds, subject to the precision and accuracy of system timers and schedulers. The thread does not lose ownership of any monitors.

Overloaded sleep methods

void sleep(long milli)	Causes the currently executing thread to sleep for specify number of milliseconds.
void sleep(long millis, int nanos)	Causes the currently executing thread to sleep (cease execution) for the specified

number of milliseconds plus the specified number of nanoseconds, subject to the precision and accuracy of system timers and schedulers.

Code Example

Following code is simple example of `sleep()` method:

```
public void run ()
{
    while(true)
    {
        //some code here
        try
        {
            Thread. sleep (10);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace ();
        }
    }
}
```

yield()

The signature of method is:

```
public static void yield () throws InterruptedException
```

`yield()` method causes currently executing thread to yield. If there are other runnable threads whose priority is at least as high as this thread, they will be scheduled next.

Method

static void <code>yield()</code>	Causes the currently executing thread object to temporarily pause and allow other threads to execute.
-------------------------------------	---

Code Example

Following code is simple example of `yield()` method:

```
package com.seed.threadProjects;
public class YieldDemo implements Runnable
{
    @Override
    public void run()
    {
        for (int x = 1; x <= 4; x++)
        {
            if (x == 2)
            {
                Thread.yield();
            }
            else
            {
                System.out.println(x +" "+ "Thread name is "
                    +Thread.currentThread().getName());
            }
        }
    }

    public static void main(String[] args)
    {
        Thread t1 = new Thread(new YieldDemo());
        Thread t2 = new Thread(new YieldDemo());
        Thread t3 = new Thread(new YieldDemo());
        t1.start();
        t2.start();
        t3.start();
    }
}
```

Output is

```
1 Thread name is Thread-0
1 Thread name is Thread-2
1 Thread name is Thread-1
3 Thread name is Thread-2
```

```
3 Thread name is Thread-0
4 Thread name is Thread-2
3 Thread name is Thread-1
4 Thread name is Thread-1
4 Thread name is Thread-0
```

join()

The signature of method is:

```
public final void join() throws InterruptedException
```

This method waits for a thread to terminate. To do this invoke join() on the thread object.

Overloaded join methods

void join()	Waits for this thread to die.
void join(long millis)	Waits at most millis milliseconds for this thread to die. A timeout of 0 means to wait forever.
void join(long millis, in t nanos)	Waits at most millis milliseconds plus nanos nanoseconds for this thread to die.

Code Example

Following code is simple example of join() method

```
package com.seed.demos;
public class JoinDemo implements Runnable
{
    public void run()
    {
        for (int x = 1; x <= 3; x++)
        {
            System.out.println("this is thread "
                +Thread.currentThread().getName());
        }
    }
}
```

```

public static void main(String[] args) throws
Exception
{
    JoinDemo j1 = new JoinDemo();
    Thread t1 = new Thread(j1, "1");
    Thread t2 = new Thread(j1, "2");
    Thread t3 = new Thread(j1, "3");
    t1.start();
    t1.join();
    t2.start();
    t3.start();
}
}

```

Output is

```

this is thread 1
this is thread 1
this is thread 1
this is thread 2
this is thread 3
this is thread 2
this is thread 3
this is thread 2
this is thread 3

```

ThreadGroup

Some programs contain number of threads .so it is very difficult to maintain such threads.

It would be useful to categorize them by functionality. So that you can work simultaneously with number of threads. Class ThreadGroup contains methods for creating and manipulating groups of threads.

ThreadGroup API

Following are the some important methods of ThreadGroup class:

Method	Description
ThreadGroup (String name)	Construct a new Thread Group
ThreadGroup (ThreadGroup parent, String name)	Creates a new Thread Group
ThreadGroup getParent ()	Returns the parent of this thread group.
int activeCount ()	Returns an estimate of the number of active threads in this thread group.
void setMaxPriority (int pri)	Sets the maximum priority of the group.
void interrupt ()	Interrupts all threads in this thread group.
void checkAccess ()	Determines if the currently running thread has permission to modify this thread group.

Code Example

Following code snippet explains the simple methods of a ThreadGroup class:

```
package com.seed.threadProject;
public class Threadgroup
{
    public static void main(String[] args)
    {
        ThreadGroup grp = new ThreadGroup("group1");
        Mythread m1 = new Mythread(grp, "thread1");
        Mythread m2 = new Mythread(grp, "thread2");
        Mythread m3 = new Mythread(grp, "thread3");
        m1.start();
        m2.start();
        m3.start();
    }
}
```

```

}
class Mythread extends Thread
{
    public Mythread(ThreadGroup g, String s)
    {
        super(g, s);
    }
@Override
public void run()
{
    System.out.println("Thread group, prority, thread
name "
+
Thread.currentThread());
}
}

```

Daemon threads

In Java, any thread can be a Daemon thread. Daemon threads are like service providers. A daemon thread is a thread that has single role in /' Daemon threads are background thread threads. The good example of deamon thread is `System.gc()`.

Daemon Threads methods

<code>void setDaemon(boolean daemon)</code>	Changes the daemon status of this thread group.
<code>boolean isDaemon()</code>	Tests if this thread group is a daemon thread group.

Code Example

Following code is a simple example of Daemon Thread:

```

package com.seed.threadProject;
public class DaemonThread extends Thread
{
    public void run()

```

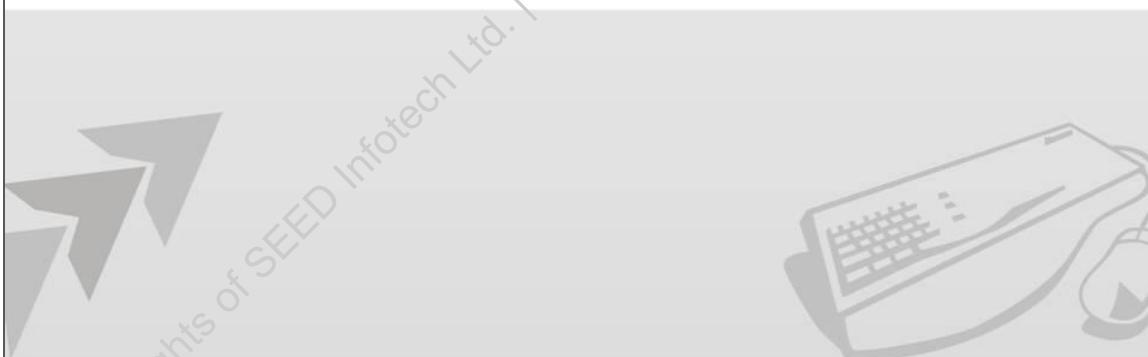
```
{  
    System.out.println("Entering run method");  
    try  
    {  
        System.out.println("In run Method: currentThread()  
is"  
        + Thread.currentThread());  
  
        while (true)  
        {  
            try  
            {  
                Thread.sleep(500);  
            }  
            catch (InterruptedException x)  
            {  
                x.printStackTrace();  
            }  
            System.out.println("In run method: woke up again")  
;  
        }  
    }  
    finally  
    {  
        System.out.println("Leaving run Method");  
    }  
}  
public static void main(String[] args)  
{  
    System.out.println("Entering main Method");  
    DaemonThread t = new DaemonThread();  
    t.setDaemon(true);  
    t.start();  
    try  
    {  
        Thread.sleep(3000);  
    }
```

```
catch (InterruptedException x)
{
    x.printStackTrace();
}
System.out.println("Leaving main method");
}
```

Copyrights of SEED Infotech Ltd. | Official Curriculum

Chapter - 5

Java Thread Communication



This chapter covers mutual exclusion using synchronization and atomic access. This chapter explains the limitations of thread synchronization and solution to these limitations.

Objectives

At the end of this chapter you will be able to:

- Implement mutual exclusion using synchronization.
- Differentiate between synchronized method and block.
- Describe Atomic variables and access.
- Demonstrate use of collections with atomic access to reduce data corruption possibility.
- Describe Thread communication using methods like `wait()`, `notify()`, `notifyAll()`.
- Demonstrate alternate mechanism of synchronization using Lock.

Thread Interference

- Common data can be accessed by various threads at the same time.
- Threads are made up of multiple operations.
- Two set of instructions can overlap on a common data item and manipulate it.
- Leads to data corruption issues.

As already seen, two threads can access the shared data (instance variables) and manipulate it as part of their own operation. By design, threads work in isolation of other threads, hence it is not possible for one thread to know what and how the other thread is changing the same piece of data that it is manipulating. Hence there is a high probability that a thread may end up working with data that is not valid. This is known as data corruption problem.

Looking at some examples will make the data corruption problem clear.

Thread Interference

C = 0

- Thread A: Retrieve c.
- Thread B: Retrieve c.
- Thread A: Increment retrieved value; result is 1.
- Thread B: Decrement retrieved value; result is -1.
- Thread A: Store result in c; c is now 1.
- Thread B: Store result in c; c is now -1.

Consider an example of an account deposit operation

```
public void deposit(Account a, float amount) {  
    float temp;  
    temp = a.bal;  
    temp+=amount;  
    a.bal = temp;  
}
```

This statement is not atomic; it is made up of 3 other statements.

load balance.

add the amount.

save the balance.

If more than one thread is accessing the same method, Account a will show a corrupted balance.

```
float temp;
```

```
temp = a.bal;
temp+=amount;
a.bal = temp
```

This statement is not atomic; it is made up of 3 other statements.

```
load balance.
add the amount
save the balance
```

Assume Account A has a balance of Rs4000. Let's also assume that thread T1 wants to deposit Rs5000 into Account A, and thread T2 Rs3000.

T1 goes through step 1 and 2, A has $4000+5000 = 9000$ Rs. but before T1 can save back, its time-slice of execution is over and Thread T2 get a chance to execute. This is possible because as we have mentioned the process is not an atomic one.

Thread T2 loads the previous balance i.e. Rs4000, adds 3000 to it and saves the result back, i.e. Rs 7000. T1 now gets back the chance at execution and stores its result. Thus at the end of the operation, account ends up with only Rs9000.

If more than one thread is accessing the same method, Account A will definitely show an incorrect balance.

Task	Amount	Amount in runtime
stacks		
T1 reads amount	4000	4000
T1 adds to amount	5000	9000
// Switch		
T1 reads amount	4000	4000
T2 adds to amount	3000	7000
T2 saves	7000	7000
//switch		
T1 saves	9000	9000

Thread Synchronization

- Key to synchronization is the concept of the monitor / mutex.
- A monitor is an object that is used as a mutually exclusive lock.
- Only one thread can own a monitor at a given time.
- When a thread acquires a lock, it is said to have entered the monitor.
- All other threads attempting to enter the locked monitor will be suspended until the first thread exits the monitor.

As seen in the earlier example, due to multiple threads accessing the same common data problem of incorrect data arises. To solve this problem in Java the access to the data should be synchronized (literal meaning timed). Simplest way of doing this would be to ensure that the common data gets accessed by only one thread at a time and other threads that are interested in the data would wait for it to become free.

The problem is not really with the data, the problem lies with the set of instructions that manipulate the data. These instructions are what are going to be executed by multiple threads simultaneously; there has to be a way to associate a thread identity with the contentious code being executed. This is facilitated in Java by the idea of monitors.

A monitor is a virtual lock on the piece of data that is being manipulated. As it is a lock, there will be no data corruption issues, as the other threads which can change the data are no longer in a position to access the data. This works on a

simple idea of mutual exclusivity (mutex). Hence, sometimes a monitor is also referred to as a mutex.

The monitor protected code can be thought of as a bank where there is only a single key to come in or get out, and a thread must get the key to enter the monitor and access these protected resources.

Once a thread gets a monitor's key, the thread can access any of the resources controlled by that monitor countless times, as long as the thread still owns the key. At any time, a thread can hold of many monitors keys. Different threads can hold keys for different monitors at the same time.

There are various ways to use synchronization for thread safe object manipulation.

Synchronization is built around an internal entity known as the intrinsic lock or monitor lock. (The API specification often refers to this entity simply as a "monitor.") Intrinsic locks play a role in both aspects of synchronization: enforcing exclusive access to an object's state and establishing happens-before relationships that are essential to visibility.

Synchronization

- A thread becomes the owner of the object's monitor in one of two ways:
 1. By executing a synchronized instance method of that object.
 2. By executing the body of a synchronized statement that synchronizes on the object.
- Only one thread at a time can own an object's monitor.

To perform any operation on a shared object the thread locks as discussed earlier. This mechanism, creates a problem in certain situations, and leads to unnecessary bottle necks reducing performance. To overcome this developer can identify critical section of the code and lock only that code.

So, locking the code can be the process of either method-level locks or code-level blocks.

There are two ways to create synchronization in java.

Synchronized methods

By using synchronized method access problems occurring due to concurrent update can be eliminated. Note that 'synchronized' qualifier to method prohibits access of multiple threads to single object - If there are two Account objects; situation might arise where two different threads are accessing same synchronized method on each object. In essence synchronized keyword locks objects, not code segments.

```
public synchronized void someMethod ()  
{  
}
```

Synchronized blocks

Synchronized block is a powerful construct to introduce controlled synchronized behavior of method invocation.

If you use using syntax as

```
synchronized (obj) {  
    //...  
}
```

Synchronized Blocks

- Synchronized blocks allow for “activity-centered” Synchronization.
- Lock of an object is acquired prior to entry into block and released upon exit from block.

```
public void printValue(Point p) {  
    // variable declaration  
    synchronized(p) {  
        // code  
    }  
    // more code of the printValue  
    method  
}
```

The disadvantage of using synchronized method is equivalent to the example used. If cash in the bank is to be secured, what is preferable? Locking up the bank or locking only the cash vault!!

The same case applies to synchronization as well. Synchronizing an entire method will ensure that the data will never get corrupted, but it will also mean that only one thread will be able to access the code in a method at any given point in time.

The method may also be dealing with tasks other than data manipulation; these tasks will also be blocked from along with the data changing methods. To overcome this only the data critical code part which is liable to throw data corruption issues is locked.

This will ensure that the threads which are interested in using the methods non-data related activities can still use it and the amount of time a thread gets into wait state will be reduced by a great margin.

Atomic Access

- Actions executed in one go are known as Atomic actions.
 - Reads and writes for reference and primitive variables (except long and double).
 - Reads and writes for all variables declared volatile.
- Atomic actions are executed without the threat of thread interference.

Atomic actions are the ones that can be successfully executed only once. Atomic action cannot be interrupted during its execution. It either happens completely or it doesn't happen at all. No side effects of an atomic action are visible until the action is complete. Some actions can be specified as atomic:

- Reads and writes for reference and primitive variables (except long and double).
- Reads and writes for all variables declared volatile.

Atomic actions are executed without the threat of thread interference. `java.util.concurrent.atomic` package defines atomic operations on single variables. All the classes have get and set methods that simulate read and write on volatile variable.

Using simple atomic variable access is more efficient than accessing these variables through synchronized code, but requires more care by the programmer to avoid memory consistency errors. Whether the extra effort is worthwhile

depends on the size and complexity of the application. It reduces the possibility of data corruption.

Let's take an example, suppose we want to access the counter variable in synchronized blocks or methods, or else use a volatile variable. These types of operations are somehow risky because some updates could be missed if they happen concurrently. To avoid this problem

`java.util.concurrent.atomic` package defines atomic data type like `AtomicInteger`. The importance of `AtomicInteger` class is when it is used such as method `incrementAndGet()` can be called. This method wraps round a machine instruction (or instructions) and will read increment and set the underlying value in memory as a 'locked together' (atomic) action. `java.util.concurrent.atomic` package defines atomic operations on single variables. get and set methods simulate read and write on volatile variable. Changes on volatile variables are visible to all threads. When a thread modifies the variable; effect is visible to all threads currently using the variable. Reduces the possibility of data corruption.



Additional Reading

The keyword **volatile** is used in the multithreaded environment.
Read more about volatile keyword use the following link

<http://www.javabeat.net/tips/169-volatile-keyword-in-java.html>

Concurrent Collections

- ConcurrentHashMap
 - Sub interface of `java.util.Map`
 - Atomic methods that replace or remove a key-value pair only if key is present
 - Atomic methods that add a key-value pair only if key is present
 - Avoids synchronization
 - Implemented by `ConcurrentHashMap`
- ConcurrentNavigableMap
 - Provides navigable support analogous to `NavigableMap`

Java comes with a new interface `ConcurrentMap`. This interface is in a package called `java.util.concurrent`. It is a sub-interface of `Map` interface. The `Atomic` methods add a key-value pair only if the key is present. For adding to the map, `putIfAbsent()` method. It accepts the key and value to add to the `ConcurrentMap` implementation, like the normal `put()` method, but will only add the key to the map if the map does not contain the key. If the map already contains the key, the existing value for the key is preserved. The `putIfAbsent()` method is atomic. It avoids synchronization.

ConcurrentMap API

Some of the important methods of `ConcurrentMap` interface:

Method	Description
<code>V putIfAbsent (K key, V value)</code>	If the specified key is not already associated with a value, associate it with the given value.

boolean remove(Object key, Object value)	Removes the entry for a key only if currently mapped to a given value.
V replace(K key, V value)	Replaces the entry for a key only if currently mapped to some value.
boolean replace(K key, V oldValue, V newValue)	Replaces the entry for a key only if currently mapped to a given value.

Code Example

Following code snippet explains the simple methods of a CoundcurrentMap:

```
package com.seed.threadProject;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.concurrent.*;
class DemoConcurr
{
    public static void main (String [] args)
    {
        ConcurrentMap<String, String> myMap = new
ConcurrentHashMap<String,
String>();
        myMap.put ("1", "1");
        myMap.put ("2", "1");
        myMap.put ("3", "1");
        myMap.put ("4", "1");
        myMap.put ("5", "1");
        myMap.put ("6", "1");
        System.out.println (myMap);
        /* print map*/
        myMap.remove ("1","1");
        System.out.println (myMap);
        /*Removes the entry for a key only if currently
mapped to a given value.*/
    }
}
```

```

System.out.println (myMap.putIfAbsent ("1","8"));
    System.out.println(myMap);
    /*If the specified key is not already associated
with a value, associate it
        with the given value.*/
    System.out.println (myMap.putIfAbsent ("1","9"));
    System.out.println(myMap);
    /*If the specified key is not already associated
with a value, associate it
        with the given value.*/
    myMap.replace ("8","10");
    System.out.println(myMap);
    /*Replaces the entry for a key only if currently
mapped to some value.*/
    System.out.println (myMap.putIfAbsent ("8","8"));
    System.out.println (myMap);
    /*If the specified key is not already associated
with a value, associate it
        with the given value.*/
    myMap.replace ("8","10");
    System.out.println (myMap);
    /*Replaces the entry for a key only if currently
mapped to some value.*/
}
}

```

Output is

```

{1=1, 5=1, 6=1, 3=1, 4=1, 2=1}
{5=1, 6=1, 3=1, 4=1, 2=1}
null
{1=8, 5=1, 6=1, 3=1, 4=1, 2=1}
8
{1=8, 5=1, 6=1, 3=1, 4=1, 2=1}
{1=8, 5=1, 6=1, 3=1, 4=1, 2=1}
null
{1=8, 8=8, 5=1, 6=1, 3=1, 4=1, 2=1}
{1=8, 8=10, 5=1, 6=1, 3=1, 4=1, 2=1}

```

Thread Synchronization

- Limitations
 - Thread waiting for lock acquisition can not be interrupted.
 - Lock release to be executed in the reverse order.
- To overcome the limitations with Synchronization Lock Interface can be used.
 - Lock Acquisition and release in different scopes
 - Allows multiple locks to be acquired and released in any order .
 - Explicit unlocking is required.
 - Code to execute in try-catch /try-catch-finally block.

Functional limitations to thread synchronization

1. Not possible to interrupt a thread that is waiting for acquiring Lock
2. Synchronization needs locks to be releases in the same order in which they were acquired.

Solution to the above problem is implementing Lock interface .

Chain locking is possible using Lock.

For example: Some algorithms for traversing concurrently accessed data structures require the use of "hand-over-hand" or "chain locking":

Algorithm for chain locking:

1. Acquire the lock of node A
2. Acquire the lock of node B
3. Release A and acquire C
4. Release B and acquire D and so on.

Implementing Lock interface enable

1. Lock acquisition and released in different scopes.
2. Allows multiple locks to be acquired and released in any order.

This flexibility comes with additional responsibility of unlocking explicitly.

Locking achieved in the following manner:

```
Lock l = new ReentrantLock();
try
{
    l.lock();
    //Update object state
}
finally
{
    l.unlock();
}
```

Since locking and unlocking happens in different scopes, it needs to be ensured that the entire code that is executed while lock is acquired is executed in try-catch-finally or try-finally blocks.

Lock interface

- The biggest advantage of implicit locks is their ability to back out of an attempt to acquire a lock.
- It is available in package
`java.util.concurrent.locks`
- Locks allow more flexible structuring.
- A `Lock` can also provide behavior and semantics that is quite different from that of the implicit monitor lock, such as guaranteed ordering, non-reentrant usage, or deadlock detection.

In a synchronized application it is not possible to interrupt a thread that is waiting for acquiring lock. Syncronization needs locks to be released in the same order in which they were acquired. To overcome to this problem Lock interface come into the picture. A `Lock` interface provides different behavior and semantics than implicit lock. It supports to guaranteed ordering, on-reentrant usage or deadlock detection. Locks increase the flexibility in an application. This increased flexibility comes additional responsibility.

```
Lock l = ...;
l.lock();
try {
    // access the resource protected by this lock
} finally {
    l.unlock();
}
```

It has to be ensured that the lock is released when necessary.

All Lock implementations must enforce the same memory synchronization semantics as provided by the built-in monitor lock:

- A successful lock operation acts like a successful monitorEnter action.
- A successful unlock operation acts like a successful monitorExit action.

Lock API

Some of the important methods of Lock interface:

Method	Description
void lock()	Acquires the lock.
void lockInterruptibly()	Acquires the lock unless the current thread is interrupted.
Condition newCondition()	Returns a new Condition instance that is bound to this Lock instance.
boolean tryLock()	Acquires the lock only if it is free at the time of invocation.
boolean tryLock(long time, TimeUnit unit)	Acquires the lock if it is free within the given waiting time and the current thread has not been interrupted.
void unlock()	Releases the lock.

Code Example

Following code snippet explains the simple methods of a Lock:

```
public void deposit (int amount)
{
    try
    {
        account.lock.lock ();
        int balamt= account.getBalance ();
        System.out.print ("Deposit :\t\t\t"+amount);
        int bal = balamt + amount;
    }
}
```

```
account.setBalance (bal);
balamt=account.getBalance ();
System.out.print ("\t"+balamt);
System.out.println ();
}
catch (Exception e)
{
    e.printStackTrace ();
}
finally
{
    account.lock.unlock ();
}
}
```

ReentrantLock

- Mutually exclusive lock.
- Can be obtained by the thread which has acquired lock successfully.
- Obtained without unlocking previous lock.
- It supports the features like lock polling, timed lock waits, and interruptible lock waits.
- ReentrantLock class implements Lock, Serializable interfaces.

Reentrant is mutually exclusive lock with the basic behavior same as that of lock acquired by synchronized methods but with extended functionalities. Reentrant lock is obtained by the thread which has locked successfully but is not unlocking. A thread invoking lock will return successfully acquiring lock when it is not locked by another thread.

```
public class ReentrantLock implements Lock,  
Serializable
```

ReentrantLock API

Some of the important methods of ReentrantLock class:

Method	Description
ReentrantLock ()	Creates an instance of ReentrantLock.
ReentrantLock (boolean fair)	Creates an instance of ReentrantLock with the given fairness policy.

<code>int getHoldCount ()</code>	Queries the number of holds on this lock by the current thread.
<code>Thread getOwner ()</code>	Returns the thread that currently owns this lock or null if not owned.
<code>boolean isLocked ()</code>	Queries if this lock is held by any thread.
<code>void lock ()</code>	Acquires the lock.
<code>Condition newCondition ()</code>	Returns a Condition instance for use with this Lock instance.
<code>boolean tryLock ()</code>	Acquires the lock only if it is not held by another thread at the time of invocation.
<code>boolean tryLock (long timeout, TimeUnit unit)</code>	Acquires the lock if it is not held by another thread within the given waiting time and the current thread has not been interrupted.
<code>void unlock ()</code>	Attempts to release this lock.

Code Example

Following code snippet explains the simple methods of a ReentrantLock:

```
//Account.java

import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

class Account
{
    int balance;
    //creating ReentrantLock object.
    Lock lock = new ReentrantLock();
    public Account (int balance)
    {
        this.balance=balance;
```

```
}

public int getBalance( )
{
    return balance;
}

public void setBalance(int balance)
{
    this.balance=balance;
}

//Transaction.java
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class Transaction extends Thread
{
    Account account;
    Transaction (Account account, String name)
    {
        super(name);
        this.account=account;
    }
    public void run()
    {
        int i=0;
        while(i<10)
        {
            try
            {
                if(Thread.currentThread().getName().equals("one"))
                {
                    withdraw(1000);
                    Thread.sleep(3000);
                }
                if(Thread.currentThread().getName().equals("two"))
                {
                    deposit(2000);
                }
            }
        }
    }
}
```

```
        Thread.sleep(1000);
    }
}
catch(InterruptedException ie)
{
    ie.printStackTrace();
}
i++;
}
}
public void withdraw(int amount)
{
    try
    {
        //acquiring lock
        account.lock.lock();
        int balamt = account.getBalance();
        System.out.print("Withdrawal :" +amount);
        int bal = balamt - amount;
        account.setBalance(bal);
        balamt =account.getBalance();
        System.out.print("\t" +balamt);
        System.out.println();
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
    finally
    {
        //unlock
        account.lock.unlock();
    }
}
public void deposit (int amount)
{
    try
```

```

{
    account.lock.lock ();
    int balamt= account.getBalance ();
    System.out.print ("Deposit: \t\t\t"+amount);
    int bal = balamt + amount;
    account.setBalance (bal);
    balamt=account.getBalance ();
    System.out.print ("\t"+balamt);
    System.out.println ();
}
catch (Exception e)
{
    e.printStackTrace ();
}
finally
{
    account.lock.unlock ();
}
}

public static void main (String [] args)
{
    System.out.println ("Hello World!");
    int initial=10000;
    System.out.println ("Original Balance: " + initial);
    Account account = new Account (initial);
    Thread t1 = new Transaction (account," one");
    Thread t2 = new Transaction (account," two");
    t1.start ();
    t2.start ();
}
}

```

Output is:

Original Balance:	10000		
Withdrawal:	1000	9000	
Deposit :		2000	11000
Deposit :		2000	13000

Deposit :		2000	15000
Withdrawal: 1000	14000		
Deposit :		2000	16000
Deposit :		2000	18000
Deposit :		2000	20000
Withdrawal :1000	19000		
Deposit :		2000	21000
Deposit :		2000	23000
Deposit :		2000	25000
Withdrawal :1000	24000		
Deposit :		2000	26000
Withdrawal :1000	25000		
Withdrawal :1000	24000		
Withdrawal :1000	23000		
Withdrawal :1000	22000		
Withdrawal :1000	21000		
Withdrawal :1000	20000		

wait()

- Method of the Object super class (not class Thread).
- Allows thread to wait inside a synchronized method.
- When invoked, the current thread is blocked and gives up the object lock.
- Waits to be notified by another thread of a change in this object.
- This lets another thread entry into the monitor.

wait belongs to the Object class and not to the Thread class. The reason for this is that the Thread will be waiting on an object which type of object is unknown at code time. So, wait has to be part of a super object which will always be available. As Object is a cosmic super class, wait () belongs to it so that it is available to any object.

```
public final void wait() throws InterruptedException
```

When the wait method is invoked, the current thread releases ownership of the monitor. Other threads which are waiting for the monitor will pick up the monitor and start their execution. The thread which has relinquished its hold on the monitor now has to wait till the time the other thread notifies that it has finished its work or it releases its own lock. The thread then waits until it can re-obtain ownership of the monitor and resumes execution.

The wait () method should only be called by the thread which is currently controlling the monitor and not by other threads.

Necessity for `notify()`, `notifyAll()`

- When a thread enters `wait()`, it has no way of unblocking itself.
- If all threads wait, it leads to DEADLOCKS.
- `notify()`:Wakes up a single thread that is waiting.
- This method should only be called by a thread that is the owner of the object's monitor.
- `notifyAll()` : Wakes up all threads that are waiting on this object.

The thread(s) that are waiting on an object wait in a queue `notifyAll()` will wake up all the objects in that queue.

```
public final native void notify()
```

Here give example of the producer-consumer program.

The following methods are available in class `Object`:

Method	Description
<code>void notify()</code>	Wakes up a single thread that is waiting on this object's monitor.
<code>void notifyAll()</code>	Wakes up all threads that are waiting on this object's monitor.
<code>void wait()</code>	Causes current thread to wait until another thread invokes the <code>notify ()</code> method or the <code>notifyAll ()</code> method for this object.

void wait(long timeout)	Causes current thread to wait until either another thread invokes the <code>notify ()</code> method or the <code>notifyAll ()</code> method for this object, or a specified amount of time has elapsed.
void wait(long timeout, int nanos)	Causes current thread to wait until another thread invokes the <code>notify ()</code> method or the <code>notifyAll ()</code> method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

Code Example

Following code snippet explains the use of `wait`, `notify` and `notifyAll`:

```
public class ProducerConsumerTest
{
    public static void main(String[] args)
    {
        CubbyHole c = new CubbyHole();
        Producer p1 = new Producer(c, 1);
        Consumer c1 = new Consumer(c, 1);
        p1.start();
        c1.start();
    }
}

class CubbyHole
{
    private int contents;
    private boolean available = false;
    public synchronized int get()
    {
        while (available == false)
        {
            try
            {
                wait();
            }
            catch (InterruptedException e)
            {
                System.out.println("Exception caught");
            }
        }
        return contents;
    }

    public void put(int value)
    {
        contents = value;
        available = true;
        notify();
    }
}
```

```
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }

    }
    available = false;
    notifyAll();
    return contents;
}

public synchronized void put(int value)
{
    while (available == true)
    {
        try
        {
            wait();
        }
        catch(InterruptedException e)
        {
            e.printStackTrace();
        }
    }
    contents = value;
    available = true;
    notifyAll();
}
}

class Consumer extends Thread
{
    private CubbyHole cubbyhole;
    private int number;
    public Consumer(CubbyHole c, int number)
    {
        cubbyhole = c;
        this.number = number;
```

```
}

public void run()
{
    int value = 0;
    for (int i = 0; i < 10; i++)
    {
        value = cubbyhole.get();
        System.out.println("Consumer #" + this.number
                           + " got: " + value);
    }  }
}

class Producer extends Thread
{
    private CubbyHole cubbyhole;
    private int number;
    public Producer(CubbyHole c, int number)
    {
        cubbyhole = c;
        this.number = number;
    }
    public void run()
    {
        for (int i = 0; i < 10; i++)
        {
            cubbyhole.put(i);
            System.out.println("Producer #" + this.number + " put: " + i);
            try
            {
                sleep((int)(Math.random() * 100));
            }
            catch(InterruptedException e)
            {
                e.printStackTrace();
            }
        }
    }
}
```

Condition Interface

- Condition replaces the use of Monitor object methods.
- One thread waits till notified by another thread about condition becoming true.
- Condition state is shared by the threads .
- A condition instance is associated with the lock instance.
- For Example
 - Thread retrieving and storing items in a data structure.

Condition replaces the use of Object Monitor methods. With condition implementation, one thread waits till notified by another thread about the condition becoming true. Since condition state is shared by the threads locking is required. A condition instance is associated with a lock. To identify the condition instance associated with this lock its newCondition () method can be used.

For Example: A thread is getting items from the data structure which supports insert and retrieve methods. If an attempt is made to retrieve and item is not available then the thread will block until an item is made available.

If an attempt is made to insert an item when data structure is full, thread will wait till some items are removed which makes the underlying data structure somewhat empty.

Separate notification criteria will have to be associated with separate conditions. This can achieved a specific condition with the lock and linking it with corresponding with code.

Condition API

Some of the important methods of Condition interface:

Method	Description
void await()	Causes the current thread to wait until it is signalled or interrupted.
boolean await(long time, TimeUnit unit)	Causes the current thread to wait until it is signalled or interrupted, or the specified waiting time elapses.
long awaitNanos(long nanosTimeout)	Causes the current thread to wait until it is signalled or interrupted, or the specified waiting time elapses.
void awaitUninterruptibly()	Causes the current thread to wait until it is signalled.
boolean awaitUntil(Date deadline)	Causes the current thread to wait until it is signalled or interrupted, or the specified deadline elapses.
void signal()	Wakes up one waiting thread.
void signalAll()	Wakes up all waiting threads.

Code Example

Following code snippet explains the simple methods of a Condition:

```
package com.seed.condition;
class DataStorage
{
    final Lock lock = new ReentrantLock();
    final Condition storageFull = lock.newCondition();
    final Condition storageEmpty = lock.newCondition();
    final Object[] items = new Object[10];
    int addptr, removeptr, counter;
```

```
public void add(Object x) throws  
InterruptedException  
{  
    lock.lock();  
    try  
    {  
        while (counter == items.length)  
            storageFull.await();  
        items[addptr] = x;  
        if (++addptr == items.length) addptr = 0;  
        ++counter;  
        storageEmpty.signal();  
    }  
    finally  
    {  
        lock.unlock();  
    }  
}  
public Object remove() throws InterruptedException  
{  
    lock.lock();  
    try  
    {  
        while (count == 0)  
            storageEmpty.await();  
        Object x = items[removeptr];  
        if (++removeptr == items.length) addptr = 0;  
        --count;  
        storageFull.signal();  
        return x;  
    }  
    finally  
    {  
        lock.unlock();  
    }  
}
```

```
public static void main(String args[]) throws
Exception
{
    DataStorage ds=new DataStorage();
    Date d1=new Date(3,3,80);
    Date d2=new Date(2,2,80);
    ds.add(d1);
    ds.add(d2);
    ds.remove();
}
}
class Date
{
    int dd,mm,yy;
    public Date()
    {
        dd=12;
        mm=4;
        yy=75;
    }
    public Date(int d,int m,int y)
    {
        dd=d;
        mm=m;
        yy=y;
    }
    public String toString()
    {
        return dd+"/"+mm+"/"+yy;
    }
}
```

Chapter - 6

Java Applets



This chapter covers applet life cycle. It contains concepts like, appletContext web start and inter applet communication using javascript etc.

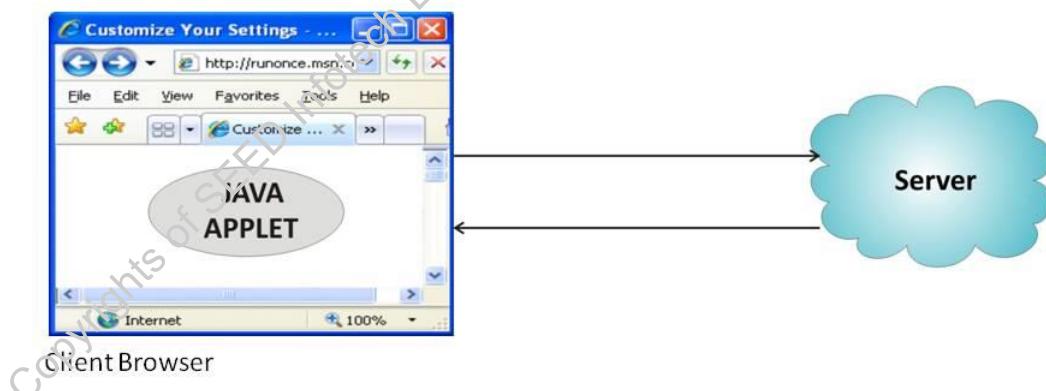
Objectives

At the end of this chapter you will be able to:

- Identify the need for Applets.
- Describe Applet Life Cycle.
- Construct an Applet.
- Construct the HTML file with Applet tag.
- State the concept of AppletContext class.
- Use Parameters with Applet tag.
- State the restrictions on Applets.
- Identify need of web start.
- Describe inter Applet communication.

Why Applets

- Classic client-server architecture requires UI to be separate from business logic.
- UI code on client side results in heavy clients and frequent application installations.



In client-Server architecture, the clients are typically thick and require more memory due to capabilities of graphical UI for example Visual Basic or Power builder. Secondly, they need installations to be done on each of the client machines and updated with every version upgrade. This becomes an issue for environment like internet.

Instead of installing first and then connecting, a dynamic approach was used. For this, idea used was putting the client program on the server side and loading it to the client for execution whenever needed. Once the job is done, it can be destroyed. To make this dynamic download efficient, the code size required to be small but powerful. What was needed was a small application which could be downloaded at runtime from the server and which will work on all the platforms. And an Applet was born!!!

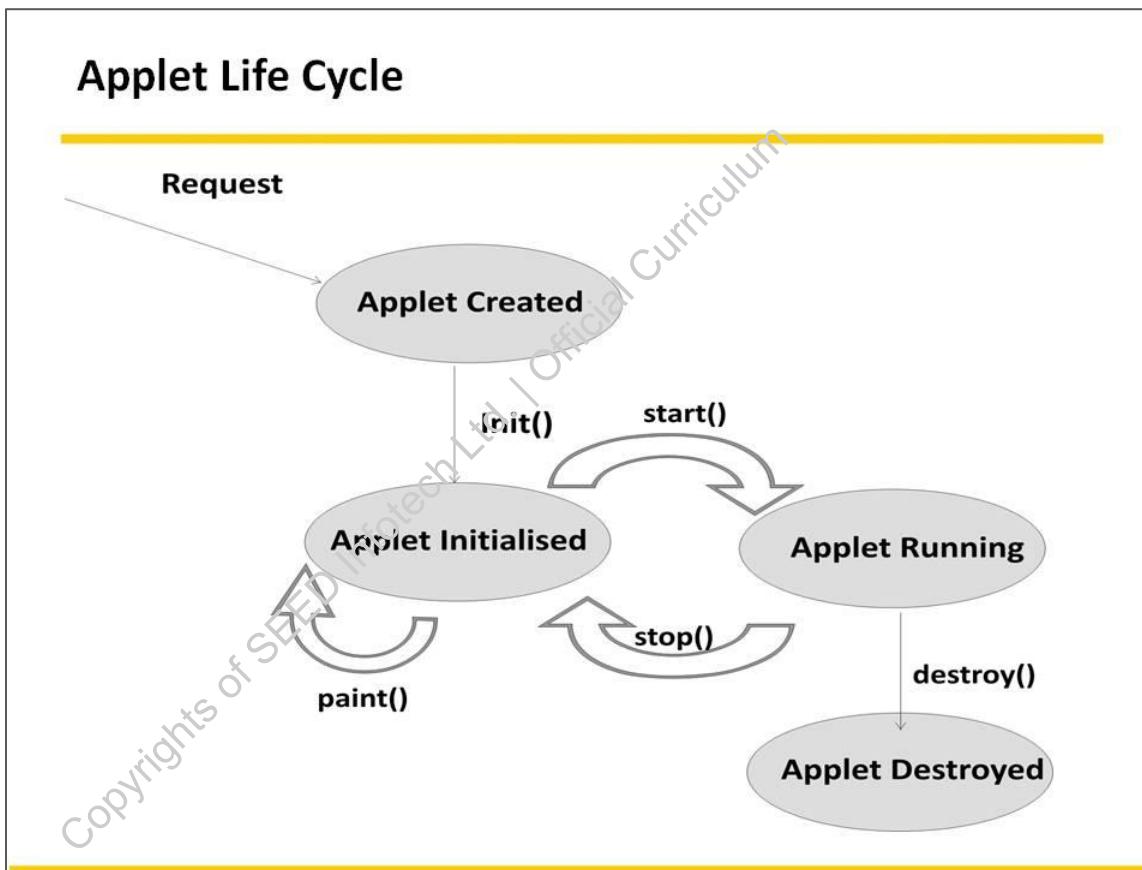
The program code for an applet resides somewhere on the network/internet on a server. Using HTTP protocol it will be transferred to client i.e. Web browser. A Java-enabled browser would have Java runtime to execute the Java code of the applet.



An applet has more processing power as compared with simple HTML or Javascript. Recently there are many tools for creating rich client interfaces like Action Script, Java FX, etc.; however Applet was the first attempt of web client rich user interface.

Code Example: Following code shows life cycle methods for an applet:

```
//HelloApplet.java
package com.seed.demos;
import javax.swing.JApplet;
public class MySEEDApplet extends JApplet
{
    public void init ()
    {
        super.init ();
    }
    public void destroy ()
    {
        super.destroy ();
        System.out.println("Applet being destroyed");
    }
    public void start ()
    {
        //Invoked whenever the Applet gains focus
        super.start ();
    }
    public void stop ()
    {
        // Invoked whenever the Applet loses focus
        super.stop ();
    }
    public void paint (Graphics arg0)
    {
        super.paint (arg0);
        arg0.drawString ("Hello Applet: Welcome to SEED
Infotech", 50, 50);
    }
}
```



Applet is embedded in a webpage or HTML page by using a special HTML tag. The applet execution cycle starts with a call to the HTML page containing `<applet>` tag.

The browser identifies the Applet class and instantiates an object for the Applet class. Container of the applet is a web browser which instantiates the applet object. To perform an initialization, a method called `init()` is provided which is invoked after applet object creation.

init

This method acts like a constructor - it is automatically called by the system when the browser launches the applet for the first time. Typical activities in an applet initialization process include processing parameter values from the applet tag and adding user-interface components. It is valid for an applet to have a default constructor but all initialization should be part of `init` method, instead of the default constructor.

start

This method is triggered by successful execution of the `init()` method. It is also called whenever the Applet-page regains the focus of the user (user may visit some other page while applet is executing and come back). This means that the `start` method will be called repeatedly, unlike the `init` method. All tasks which need to be repeated for every execution have to be put inside `start` method. For example, restarting a thread for an applet to resume an animation. If the applet does not perform operations that need to restart when the focus is regained, it is not necessary to implement this method.

stop

This method is automatically called when the Applet-page loses focus. It will be called repeatedly for the same applet. The purpose of this method is to enable suspension of a time-consuming activity and preventing slowing down of the system when the user is not paying attention to the applet. This method is not to be called explicitly. If the applet does not perform operations that need to be suspended when the focus is lost, it is not necessary to implement this method.

destroy

This method is invoked when the browser on the client machine is closed. This is a callback method, provided so that the developer can write the house-cleaning operations if required. Whichever resources have been consumed in the execution of the Applet are to be released in the method.

paint

This method is provided to draw the content on the screen of the applet. The applet's `paint()` method is called when the applet displays on the window. It is called whenever the contents of the applet need to be redrawn. This might happen if the applet loses focus and regains it or when window of the browser is scrolled, and the applet comes into view.

Embedding applet inside HTML Page

- **Attributes of <applet> tag**
 - CODE
 - CODEBASE
 - WIDTH / HEIGHT
 - ALIGN
 - HSPACE / VSPACE
 - NAME
- **Nested tag**
 - <PARAM> tag

Applet is embedded inside an HTML page by using an applet tag. Following are the attributes used to provide more information to the applet in terms of position, alignment, parameters etc. Following are the attributes:

CODE

This required attribute gives the name of the applet class file which needs to be loaded by the browser. Absolute path names are not acceptable.

CODEBASE

Tells browser to search for applet class files in the directory specified by the URL.

NAME

This tag is used rarely, but is necessary when two applets on the same HTML page need to communicate with each other. These names are virtual names, which are used to get a reference to the other applets for communicating with them.

WIDTH, HEIGHT

These attributes are mandatory and provide the width and height (pixels) for the applet in the HTML page.

ALIGN

This attribute specifies the alignment of the applet. There are two options available. The applet can be a central component, with text flowing around it, or the applet can be inline, floating inside a line of text.

PARAM

Applet process customization can be achieved by providing controlling data from outside the environment using the Param tag. The HTML file containing the <Applet> Tag must have the nested tag <Param>, with name and value embedded. At runtime this data can be retrieved programmatically.

Passing parameters to Applet

```
<table cellpadding="5" bgcolor="#FF0000">
  <tr>
    <td>
      <APPLET CODE="MyApplet.class" WIDTH="400"
      HEIGHT="200">
        <PARAM NAME ="fontName" VALUE="Times New
        Roman">
        <PARAM NAME ="size" VALUE="24" >
      </APPLET></td>
    </tr>
  </table>
```

Since browsers are chosen as medium of execution for Applets on the client side, even the instruction to execute the Applet needs to be given in the language the browser understands, i.e., HTML. Following code shows how to pass parameters to applet using an HTML file and use them inside the applet code.

```
package com.seed.appletDemo;
import java.awt.*;
import java.applet.*;
import javax.swing.*;
public class DemoApplet extends JApplet
{
  String str;
  int size;
  Font f;
  public void init()
  {
    str= getParameter("fontName");
    String x=getParameter ("size");
```

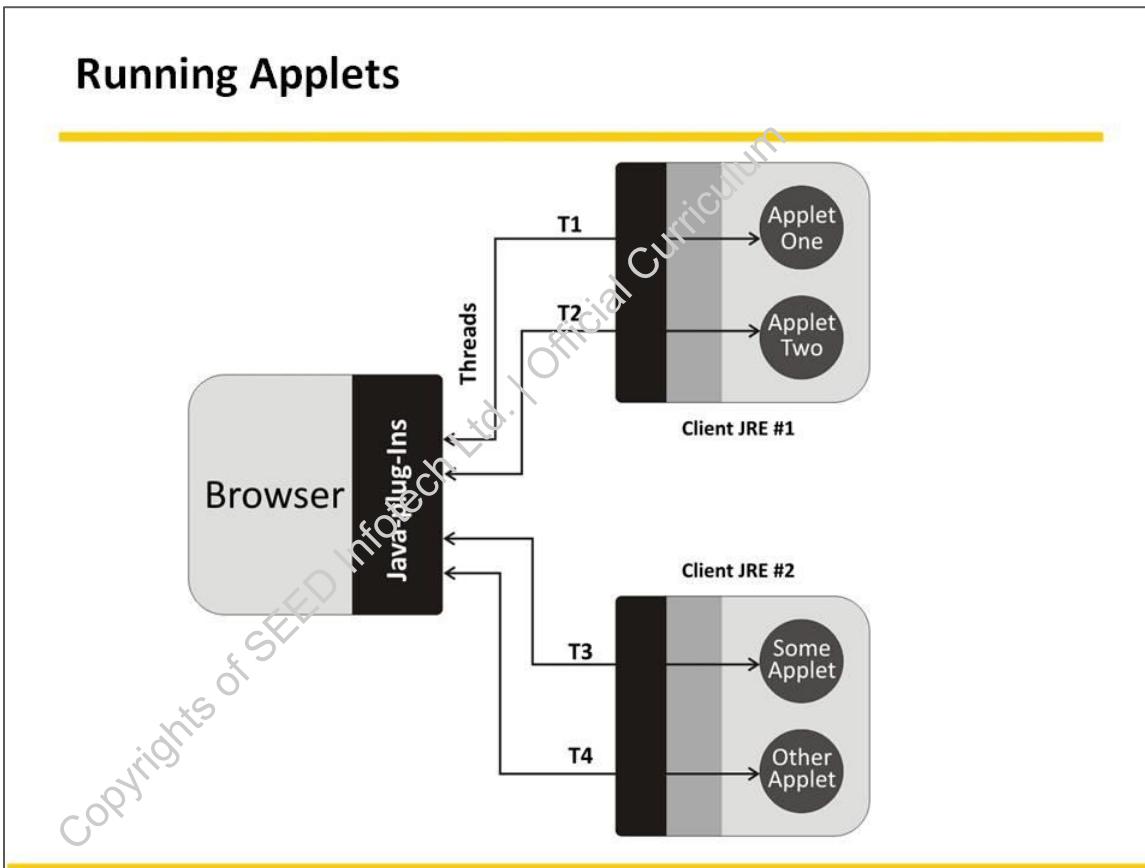
```
size=Integer.parseInt(x);
f=new Font (str, Font.ITALIC, size);
}
public void paint(Graphics g)
{
    g.setFont(f);
    g.drawString("Hello",200,200);
}
<html>





```

Running Applets



An applet uses a web-browser to execute. Almost all the browsers have a helper utility called the Java Plug-in to control and monitor the execution of the applet in local environment. The Java Plug-in software creates a worker thread for every applet. It launches an applet in an instance of the Java Runtime Environment (JRE) software. Normally, all applets run in the same instance of the JRE.

The Java Plug-in software starts a new instance of the JRE in the following cases:

- When an applet requests to be executed in a specific version of the JRE.
- When an applet specifies its own JRE startup parameters, for example, the heap size.

Applets can invoke JavaScript functions present in the web page. JavaScript functions are also allowed to invoke methods of an applet embedded on the same web page. The Java Plug-in software and the JavaScript interpreter orchestrate calls from Java code to JavaScript code and calls from JavaScript code to Java code.



Tech App

On all platforms, the new Java Plug-in locates JREs to use from the entries listed in the Java Control Panel ("Java" tab, "View" button under "Java Applet Runtime Settings"). The available JREs in this list are encoded in the `deployment.properties` file whose location is platform-dependent. On the Windows platform, it is generally located in

`C:\Documents and
Settings\[username]\Application
Data\Sun\Java\Deployment.`

The Applet Context

- Context (browser/ applet Viewer) in which the applet runs.
- Enables an applet to access features of the browser that contains it.

AppletContext



As the name suggests an AppletContext is encapsulation of the environment under which the Applet is currently running. This can be used for various purposes.

AppletContext API

Following are the some of the important methods AppletContext:

Method	Description
Applet getApplet (String name)	returns a handle to a named applet.
AudioClip getAudioClip (URL soundFileURL)	returns an audio clip object.
Image getImage (URL	returns an image.

imageFile)	
void showDocument (URL docURL)	requests the browser to display the HTML page specified in parameter.

Code Example

Following code snippet explains the simple methods of an AppletContext:

```

package com.demo.appletDemo;
import java.applet.JApplet;
import java.applet.AudioClip;
import java.awt.Graphics;
import java.net.MalformedURLException;
import java.net.URL;
/** AppletMethods --
    show stop/start and AudioClip methods */
public class MyApplet extends JApplet
{
    /** AudioClip object, used to load and play a sound file. */
    AudioClip snd = null;
    /** Initialize the sound file object and the GUI. */
    public void init()
    {
        System.out.println("In MyApplet.init()");
        try
        {
            snd = getAudioClip(new URL(getCodeBase(), "laugh.au"));
        }
        catch(MalformedURLException e)
        {
            showStatus(e.toString());
        }
        setSize(200,100); // take the place of a GUI
    }
    /** Called from the Browser when the page is ready to
    */
}

```

```
go. */
public void start()
{
    System.out.println("In MyApplet.start()");
    if (snd != null)
        snd.play();
}
/** Called from the Browser when the page is being va-
cated. */
public void stop()
{
    System.out.println("In MyApplet.stop()");
    if (snd != null)
        snd.stop(); // stop play() or loop()
}
/** Called from the Browser (when the applet is being
un-cached?).
 * Not actually used here, but the println will show
when it's called.
 */
public void destroy()
{
    System.out.println("In MyApplet.destroy()");
}

public void paint(Graphics g) {
    g.drawString("Welcome to Java while music
plays", 50, 50);
}
```

Applet Security Restrictions

- Cannot read / write files on user's file system.
- Cannot communicate with an internet site other than the one that served the web page that included the applet.
- Cannot run any executable program.
- All windows popped by an applet carry a warning message.
- Cannot find any information about the local computer.

Applets are downloaded and executed on client machine at runtime. Any maliciously created applet can access client resources and damage them beyond repair. Considering this possibility, there are many restrictions applied on applets.

An applet cannot have any way of communicating with any location other than its place of origin. Otherwise, a developer could potentially write a piece of code which would signal a third-party site to use the applet as a conduit to tunnel in to the client's machine without the client being aware of it. An applet could also gain control of an executable (local or remote) and try to run it under the client-context, which may lead to unwanted situations on client side.

All such problems are avoided by the inherent design of the Applets. Applets are executed in what is called as "SandBox" mode. It signifies that the client machine is bound to provide memory space to execute the applet, but it will also seal the applet within the same memory space, in effect turning it into a closed environment. An applet can execute its code, but cannot access anything outside the memory space (SandBox) allocated to it.



Interview Tip

It is important to Applet restrictions, what applets can do and cannot do.

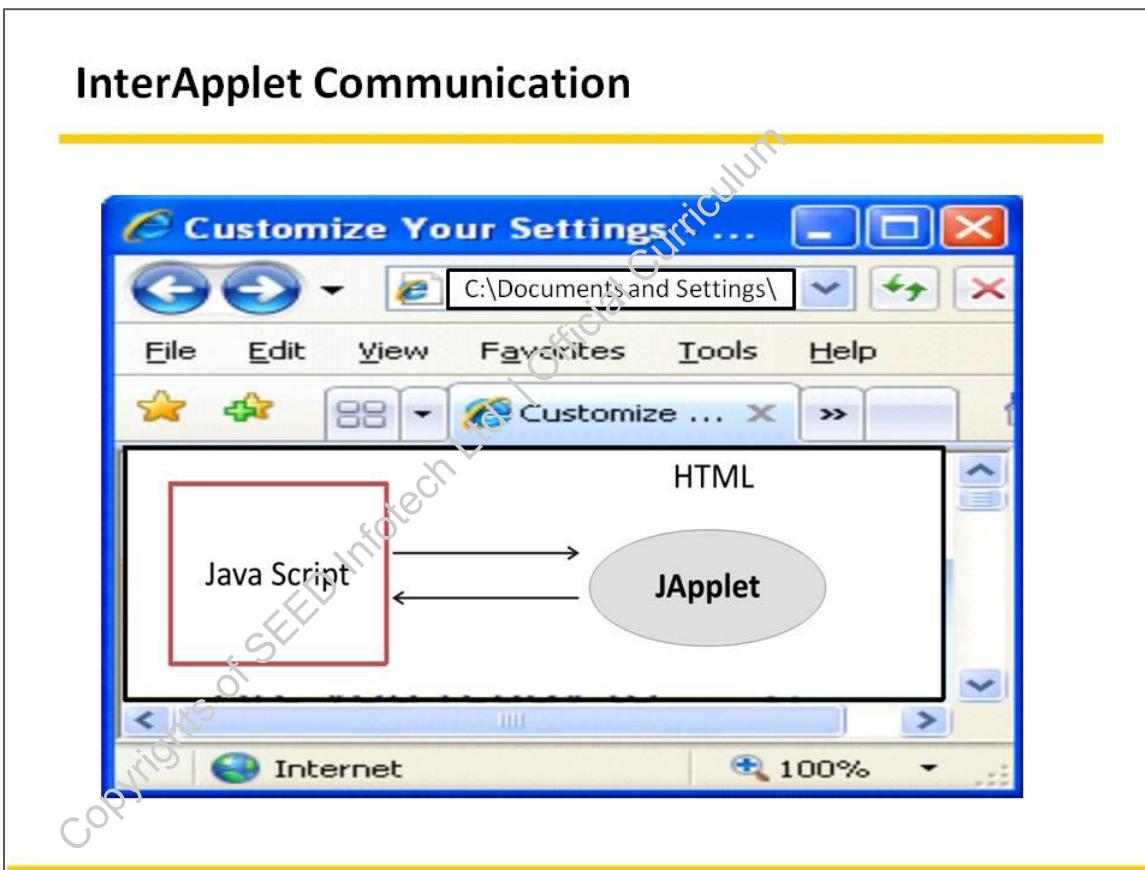
WebStart

- Java WebStart is a new technology that is platform-independent , secure and robust.
- The Java WebStart software allows you to download and run Java applications from the web.
- Java WebStart is general java program which requires main class to execute the application.
- WebStart application launch through browser.
- Why WebStart?
 - Provides an easy, one-click activation of applications .
 - Always running the latest version of the application .

Java WebStart is a mechanism which allows a user to download a Java application from the web and execute it locally on the client's machine. WebStart is similar to an Applet, but there are certain differences. A WebStart application needs a main method to start execution as opposed to an Applet which does not need it. The major reason for this change is the fact that a WebStart does not execute inside a browser. It has its own execution process, it needs the browser only for the first time when it is launched and downloaded on the client side.

A WebStart application can be used as a normal desktop application and normally exists such that a single click can activate the processing of the application. A WebStart application is also not closely bound with the run time environment as the application is loaded by the meta-data provided through the WebStart descriptor.

WebStart works on Java Network Launch Protocol (JNLP) and a jar file descriptor is required to launch the application. And the MIME type association of the browser is to be set to application/x-java-jnlp-file.



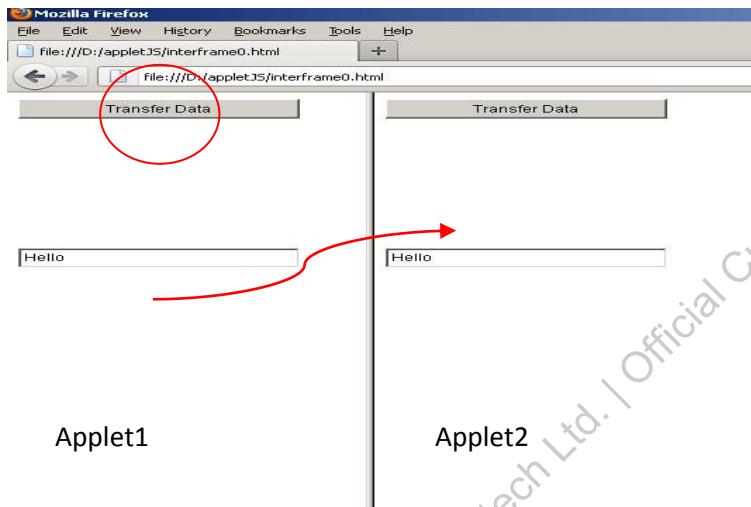
One should not use the `getApplet()` and `getApplets()` methods of the `AppletContext` class to find other applets. These methods only find applets that are running in the same JRE.

Applets must originate from the same directory on the server in order to communicate with each other.

An applet can communicate with other applets by using JavaScript functions in the parent web page. JavaScript functions enable communication between applets by receiving messages from one applet and invoking methods of other applets.

Code Example

consider an example of between javascript and an applet:



When use click on 'Transfer Data' button is clicked; the data will be transferred to second text field and vice versa.

```
<!-- interframe0.html
This is a simple html which contain 2 frames -->
<HTML><HEAD></HEAD>
<FRAMESET COLS="50%, *">
    <FRAME SRC="interframe1.html" NAME="f1" >
    <FRAME SRC="interframe2.html" NAME="f2">
</FRAMESET>
</HEAD>

<!-- interframe1.html
This html contains java script code.
-->
<HTML><HEAD></HEAD>
<SCRIPT>
function toOtherFrame (a, target)
{
    if (target == "f1")
        parent.f1.document.app1.fromOtherFrame (a);
    else
        parent.f2.document.app2.fromOtherFrame (a);
}
</SCRIPT>
```

```
</HEAD>
<BODY>
<APPLET CODE="InterFrameDemo.class" NAME="app1"
MAYSCRIPT HEIGHT=200
        WIDTH=200>
<PARAM NAME="target"
        VALUE="f2">
</APPLET></BODY></HTML>

<! - - interframe2.html
This html contains java script code.

<HTML><HEAD></HEAD>

<BODY>
<APPLET CODE="InterFrameDemo1.class"
        NAME="app2" MAYSCRIPT
        HEIGHT=200
        WIDTH=200>
        WIDTH=200>
<PARAM NAME="target"
        VALUE="f1">
</APPLET></BODY></HTML>

/*
   InterFrameDemo.java
   It accepts the text from first html page when user
   clicks on button the text transfer from one html to
   other html and vice versa.
When user click on button java script function get
called to perform this operation.
*/
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.net.URL;
```

```
import netscape.javascript.*;
public class InterFrameDemo extends Applet implements
ActionListener
{
    TextField tf;
    Button b;
    Label l;
    String target;
    public void init()
    {
        target = getParameter ("target");
        setLayout (new BorderLayout ());
        tf = new TextField(20);
        add ("South", tf);
        b = new Button ("Transfer Data");
        add ("North",b);
        b.addActionListener (this);
    }
    public void actionPerformed (ActionEvent ae)
    {
        if (ae.getSource() == b)
        {
            String js ="parent.f1.toOtherFrame (\\""
            +tf.getText () +"\", \\""
            + target + "\")";
            System.out.println ("to Javascript:" + js);
            JSObject win = (JSObject)
JSObject.getWindow(this);
            win.eval (js);
        }
    }
    public void fromOtherFrame(String s)
    {
        tf.setText(s);
    }
}
/*
```

InterFrameDemo1.java

It accepts the text from first html page when user clicks on button the text transfer from one html to other html and vice versa.

When user click on button java script function get called to perform this operation.

*/

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.net.URL;
import netscape.javascript.*;

public class InterFrameDemo1 extends Applet implements
ActionListener {
    TextField tf;
    Button b;
    Label l;
    String target;

    public void init() {
        target = getParameter("target");
        setLayout(new BorderLayout());
        tf = new TextField(20);
        add("South", tf);
        b = new Button("Transfer Data");
        add("North",b);
        b.addActionListener(this);
    }

    public void actionPerformed(ActionEvent ae) {
        if (ae.getSource() == b) {
            String js =
                "parent.f1.toOtherFrame(\"" +
                tf.getText() +
```

```
        "\",\" + target + "\")";  
        System.out.println("to Javascript:" + js);  
        JSObject win = (JSObject)  
JSObject.getWindow(this);  
        win.eval(js);  
    }  
  
    public void fromOtherFrame(String s) {  
        tf.setText(s);  
    }  
}
```



Tech App

Avoid using static variables to share data between applets.



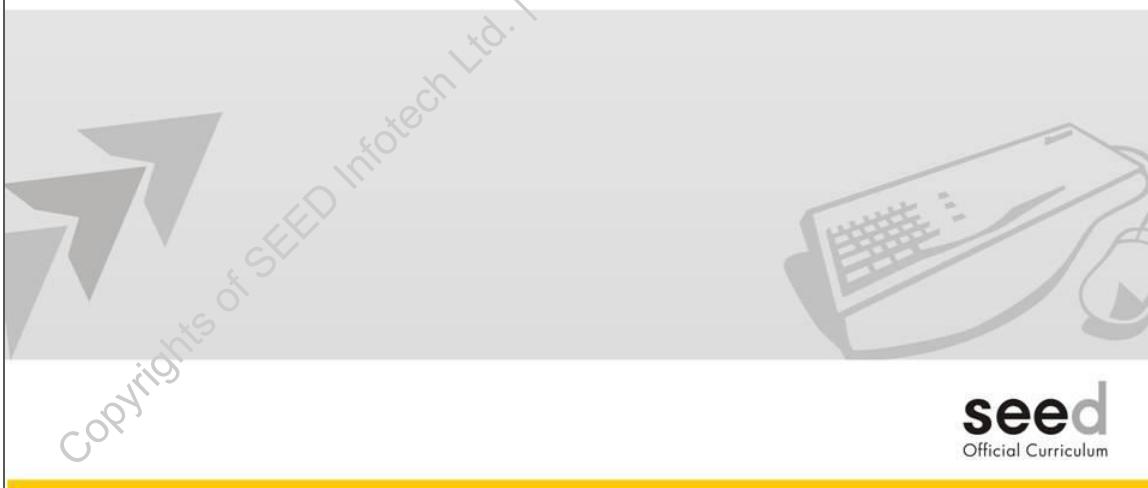
App Design

State what signed applet can do which a normal applet cannot.

Signed applets operate outside the security sandbox and have extensive capabilities to access the client. A signed applet will run outside the security sandbox only if the user accepts the applet's security certificate. If the user refuses to accept the certificate, the applet will run within the security sandbox similar to an unsigned applet.

Chapter-7

Input-Output(File IO)

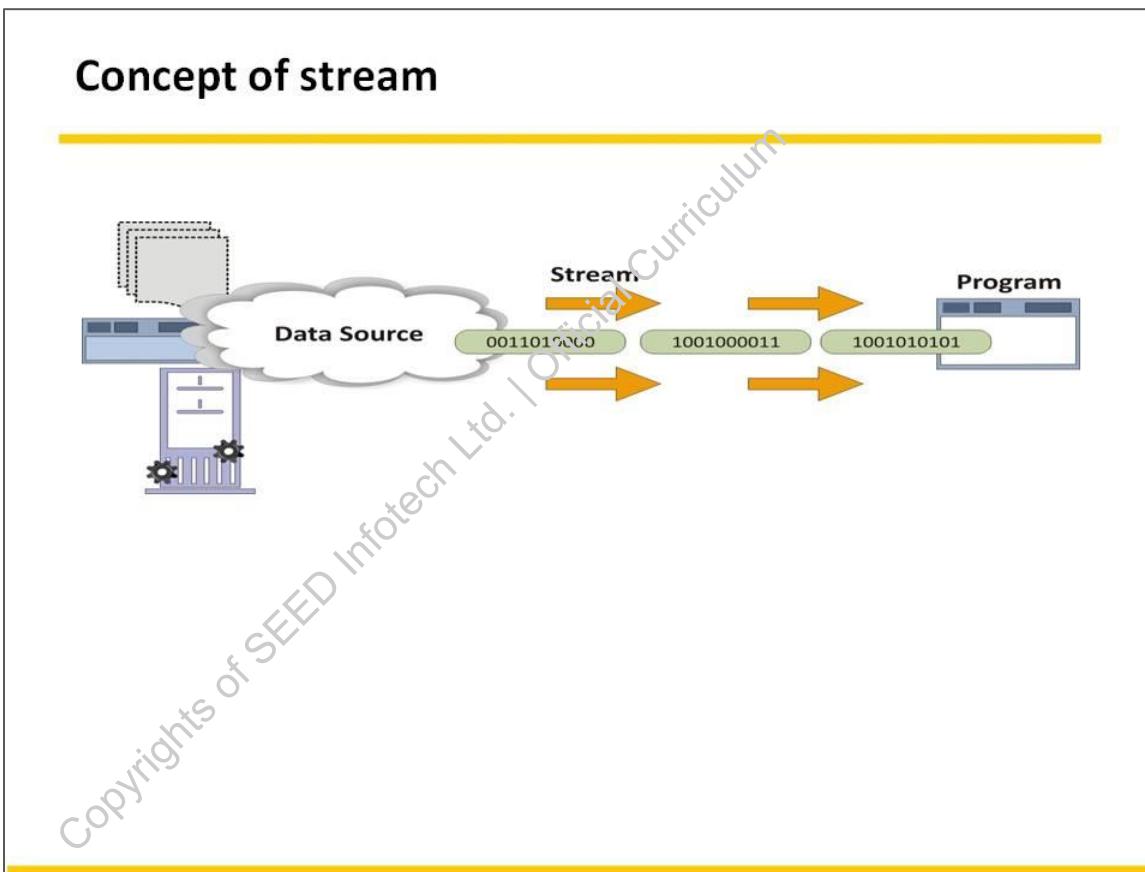


This chapter covers file handling using IO API. It includes streams, reader writer, object serialization, chaining, console input using System class, Scanner class and Console class. It also contains concept of Regular expression.

Objectives

At the end of this chapter you will be able to:

- Define Stream.
- Create a program to perform file operations like open, read, and write using `FileInputStream` and `FileOutputStream`.
- Create a program to perform file operations like open, read, and write using `RandomAccessFile`.
- Define chaining of streams.
- Identify the need of serialization.
- Use `System` class in your Java application.
- Use `Scanner` class in your Java application.
- Use `Console` class in your Java application.
- Identify the need of Regex.



Programs need ability to read and write data. This input output operation is performed on various types of physical devices like keyboard, floppy disk, Hard disk, USB drives etc. Various devices use different mechanism to read and write data. To simplify the hidden complexity programming languages created an abstraction called stream, a sequence of bytes. In real terms, these streams are nothing but buffers from where the data can be read or written. The device driver for a particular device then takes the responsibility of doing actual read or writes on a particular device. Use of such an abstraction helps languages to create programs that are independent of the physical device. Hence this can be even extended to remote communication using socket communication.

Java provides a similar abstraction in the form of stream classes, for reading an `InputStream` and for writing an `OutputStream`. There are various decorations or additional facilities provided as one goes down the hierarchy of I/O classes.

Stream

- **InputStream:** This depicts the flow of bytes from data source to the programs memory.
- **OutputStream:** This depicts the flow of bytes from the programs memory to the destination data store.
- Java views these streams in terms of objects that will perform different operations on the streams through their method calls.
- Two basic operations involved are:
 - Read from Input stream.
 - Write to output stream.

There are different types of streams:-

InputStream

It is an abstract class that defines methods to performing input operations i.e. Read a data from source.

```
public abstract class InputStream extends Object
```

This abstract class is the super class of all classes representing an input stream of bytes.

Applications that need to define a subclass of `InputStream` must always provide a method that returns the next byte of input.

So, this class is designed for byte stream classes when working with bytes or other binary objects.

Following is a set of all the classes which extend the `InputStream` class.

```
java.io.FileInputStream  
java.io.ByteArrayInputStream
```

```

java.io.SequenceInputStream
java.io.PipeInputStream
java.io.ObjectInputStream
FilterInputStream
└── java.io.DataInputStream
└── java.io.BufferedInputStream
└── java.io.LineNumberInputStream
    └── java.io.PushbackInputStream

```

InputStream API

Following are some of the important methods of the `InputStream` class:

Methods	Description
<code>InputStream()</code>	
<code>abstract int read()</code>	Reads the next byte of data from the input stream.
<code>int available()</code>	Returns an estimate of the number of bytes that can be read (or skipped over) from this input stream without blocking by the next invocation of a method for this input stream.
<code>void reset()</code>	Repositions this stream to the position at the time the <code>mark</code> method was last called on this input stream.
<code>void mark(int readlimit)</code>	Marks the current position in this input stream.
<code>long skip(long n)</code>	Skips over and discards <code>n</code> bytes of data from this input stream.

ByteArrayInputStream

ByteArrayInputStream is a sub class of InputStream. It has an internal buffer that contains bytes that may be read from the stream. An internal counter keeps track of the next byte to be supplied by the read method.

Following code snippet explains the simple methods of a ByteArrayInputStream class:

```
import java.io.*;
class ByteArrayInputStreamDemo
{
    public static void main(String args[]) throws
IOException {
    String tmp = "hello this is byte array input stream";
    byte b[] = tmp.getBytes();
    ByteArrayInputStream input1 = new
ByteArrayInputStream(b);
    int c=0;
    do
    {
        c = (byte) input1.read ();
        System.out.print ((char) c);
    }
    while (c!= -1);

}
}
```

OutputStream

This is an abstract class that defines methods to performing output operations i.e. writing a data to destination.

```
public abstract class OutputStream extends Object
```

This abstract class is a super class of all classes representing an output stream of bytes.

Applications that need to define a subclass of OutputStream must always provide a method that returns the next byte of output.

So, this class is designed for byte stream classes when working with bytes or other binary objects.

Following is a set of all the classes which extend the `OutputStream` class:

```
java.io.FileOutputStream
java.io.ByteArrayOutputStream
java.io.PipeOutputStream
java.io.ObjectOutputStream
FilterOutputStream
  └─ java.io.DataOutputStream
  └─ java.io.BufferedOutputStream
```

OutputStream API

Following are some of the important methods of the `OutputStream` class:

Method	Description
<code>OutputStream()</code>	
<code>abstract void write(int b)</code>	Writes the specified byte to this output stream.
<code>void flush()</code>	Flushes this output stream and forces any buffered output bytes to be written out.
<code>void close()</code>	Closes this output stream and releases any system resources associated with this stream.

ByteArrayOutputStream

`ByteArrayOutputStream` is a sub class of `OutputStream`. The buffer automatically grows as data is written to it.

Following code snippet explains the simple methods of a `ByteArrayOutputStream` class:

```
import java.io.*;
class ByteArrayOutputStreamDemo
{
```

```

public static void main (String args []) throws
IOException
{
    ByteArrayOutputStream f = new ByteArrayOutputStream
();
    String s = "This byte array output stream example";
    byte buf [] = s.getBytes ();
    f.write (buf);
    System.out.println (f.toString ());
    byte b [] = f.toByteArray ();
    for (int i=0; i<b.length; i++)
    {
        System.out.print ((char) b[i]);
    }
    f.reset ();
    for (int i=0; i<3; i++)
    f.write ('X');
    System.out.println (f.toString ());
}
}

```

FileInputStream

`FileInputStream` is used to obtain input bytes from file in a file system.

FileInputStream API

Following are some of the important methods of the `FileInputStream` class:

Method	Description
<code>FileInputStream(File file)</code>	Creates a <code>FileInputStream</code> by opening a connection to an actual file, the file named by the <code>File</code> object <code>file</code> in the file system.
<code>FileInputStream(String name)</code>	Creates a <code>FileInputStream</code> by opening a connection to an actual file, the file named by the path <code>name</code> in the file system.
<code>int read()</code>	Reads a byte of data from this input stream.

Code Example

Following code snippet explains the simple methods of a `FileInputStream` class and the way it works:

```
import java.io.*;
class FileInputStreamDemo
{
    public static void main (String args []) throws
IOException
    {
        FileInputStream input1 = new FileInputStream
("myfile.txt");
        int c=0;
        do
        {
            c = (byte) input1.read ();
            System.out.print ((char) c);
        }
        while (c! = -1);
    }
}
```

FileOutputStream

A file output stream is an output stream for writing data to a File:

FileOutputStream API

Following are some of the important methods of the `FileOutputStream` class and the way it works:

Method	Description
<code>FileOutputStream(File file)</code>	Creates a file output stream to write to the file represented by the specified <code>File</code> object.
<code>FileOutputStream(String name)</code>	Creates a file output stream to write to the file with the specified name.
<code>FileOutputStream(File file,</code>	Creates a file output stream to write to

boolean append)	the file represented by the specified File object.
write(int b)	Writes the specified byte to this file output stream.

Code Example

Following code snippet explains the simple methods of a FileOutputStream class:

```
import java.io.*;
class FileOutputStreamDemo
{
    public static void main (String args[]) throws
IOException
    {
        FileOutputStream f = new FileOutputStream
("myfile1.txt");
        int c=0;
        while(c!=-1) {
            f.write((char)c);

        }
    }
}
```

File class

- An abstract representation of file and directory pathnames.
- Models an OS dir entry, enabling you to access info about a file.
- Objects of file do not actually open a file or provide any file processing capabilities.
- File objects are used to do all operations related to files and directories.
- Interoperability with `java.nio.file` package.

To read and write into a file `FileInputStream` and `FileOutputStream` classes are used but to do this a file has to be passed as a parameter. So, Java gives us `File` class under `java.io` package.

`File` class gives facilities to manipulate files and directories. When new `File` object is created a path is created and not the actual file. The `File` class is not used to actually read or write data; it is used to work at a higher level, making new empty files, searching for files, deleting files, making directories, and working with paths.

File API

Following are some of the important methods of the `File` class.

Method	Description
<code>File(File parent, String child)</code>	Creates a new <code>File</code> instance from a parent abstract pathname and a child pathname string.

<code>File(String pathname)</code>	Creates a new File instance by converting the given pathname string into an abstract pathname.
<code>File(String parent, String child)</code>	Creates a new File instance from a parent pathname string and a child pathname string.
<code>File(URI uri)</code>	Creates a new File instance by converting the given file: URI into an abstract pathname.
<code>boolean canRead()</code>	Tests whether the application can read the file denoted by this abstract pathname.
<code>boolean canWrite()</code>	Tests whether the application can modify the file denoted by this abstract pathname.
<code>boolean createNewFile()</code>	Atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.
<code>boolean delete()</code>	Deletes the file or directory denoted by this abstract pathname.
<code>boolean exists()</code>	Tests whether the file or directory denoted by this abstract pathname exists.
<code>String getName()</code>	Returns the name of the file or directory denoted by this abstract pathname.
<code>boolean isHidden()</code>	Tests whether the file named by this abstract pathname is a hidden file.
<code>long length()</code>	Returns the length of the file denoted by this abstract pathname.
<code>URI toURI()</code>	Constructs a file: URI that represents this abstract pathname.

File and File Streams

Following examples shows how to use `File` and `FileInputStream` and `FileOutputStream` classes to perform read and write operations in a file.

```
//This program read data from one file and writes into  
another file.  
package com.seed.io;  
import java.io.*;  
class FileReadWrite  
{  
    public static void main (String args [])  
    {  
        FileInputStream fin = null;  
        FileOutputStream fos = null;  
        int x;  
        File f1=new File ("firstfile.txt");  
        File f2=new File ("secondfile.txt");  
        try  
        {  
            fin = new FileInputStream (f1);  
            fos = new FileOutputStream (f2);  
            do  
            {  
                x = (byte) fis.read ();      fos.write(x);  
            }  
            while (x! = -1);  
            fis.close ();  
            fos.close ();  
        }  
        catch(FileNotFoundException e)  
        {  
            System.out.println (" Cannot find file "+e);  
        }  
        catch(IOException e)  
        {  
            System.out.println (e);  
            System.exit (-1);  
        }  
    }  
}
```

RandomAccessFile class

- This class helps to find or write data anywhere in a file.
- A RandomAccessFile has a file-pointer.
- The File Pointer indicates the position of the next record that will be read or written.

This class is used for reading data from file and writing data to a file. The RandomAccessFile has a file pointer. This file pointer sets the current position in file RandomAccessFile implements DataInput and DataOutput Interfaces.

RandomAccessFile API

Following are some of the important methods of the RandomAccessFile class:

Method	Description
RandomAccessFile(File file, String mode)	Creates a random access file stream to read from, and optionally to write to, the file specified by the File argument.
RandomAccessFile(String name, String mode)	Creates a random access file stream to read from, and optionally to write to, a file with the specified name.

void close()	Closes this random access file stream and releases any system resources associated with the stream.
Filechannel getChannel()	Returns the unique FileChannel object associated with this file.
long getFilePointer()	Returns the current offset in this file.
long length()	Returns the length of this file.
int read()	Reads a byte of data from this file.
void seek(long pos)	Sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.
void write(int b)	Writes the specified byte to this file.

Code Example

Following code snippet explains the simple methods of a RandomAccessFile class:

```
//following code read data from one file and write in another file using RandomAccessFile.
package com.seed.io;
import java.io.*;
class RandomAccesDemo
{
    public static void main (String [] args) throws IOException
    {
        File f1=new File ("firstfile.txt");
        File f2=new File ("secondfile.txt");
        RandomAccessFile raf1=new RandomAccessFile (f1,"r");
        RandomAccessFile raf2=new RandomAccessFile (f2,"rw");
        int x=0;
        System.out.println ("Before setting pointer"+
"+raf1.getFilePointer ());
    }
}
```

```
raf1.seek (3);
System.out.println ("After setting pointer"+
"+raf1.getFilePointer ());
do
{
    x=raf1.read ();
    raf2.write ((char) x);
} while(x! =-1);
System.out.println ("Hello World!");
}
```

Chaining or Layering of Streams

- To use these file filters chaining of streams are required.

```
FileInputStream fis=new FileInputStream("c:\\abc.txt");
BufferedInputStream bis=new BufferedInputStream (fis);
DataInputStream dis=new DataInputStream(bis);
```

Java provides each derived class from input stream to operate with the data in a particular way. For example `FileInputStream` reads data from file as byte by byte and one wants to read the File in terms of sequence of records like `DataInputStream` and not bytes. To achieve this, functionalilites of both these need to be combined. When it is done is called layering or chaining of streams.

Here we want to open a file, store all the data in the file in a buffer and read the data from the buffer as and when required. To do so we are going to form a chain of streams as follows:

1. To read data from file `FileInputStream` is required.
2. Now to store the contents of file in a buffer `BufferedInputStream` is required.
3. Finally to read data from buffer (as and when required) use `DataInputStream`.
4. (Hint: use `read(byte[], int, int)` method in `DataInputStream` to read chunks of data.)

5. Now the FileInputStream will read all data from the file which will be stored in buffer by the BufferedInputStream and read by the DataInputStream as and when required.

In this way chaining/layering of streams required is used whenever required.



Tech App

LineNumberInputStream is an input stream filter that provides the added functionality of keeping track of the current line number. Implementation of this class is generally editors ,compilers etc.



Best Practice

Chaining of stream is a good example of Decorator Design Pattern.

For more information of Decorator Design pattern visit:

http://www.allapplabs.com/java_design_patterns/decorator_pattern.htm

Persisting Objects

```
//serial.java
try {
    Employee od=new Employee
("Rohan", 20, 10000);
    fos=new FileOutputStream
("serial.txt");
    oos=new
ObjectOutputStream(fos);
    oos.writeObject (od);
    oos.close ();
}
catch (Exception e)
{
    e.printStackTrace ();
}
```

```
//deserial.java
try{
Employee od;
fis=new FileInputStream
("serial.txt");
ois=new ObjectInputStream
(fis);
od= (Employee)
ois.readObject ();
od.display ();
ois.close ();
}
catch (Exception e)
{
    e.printStackTrace ();
}
```

A typical enterprise level application will have multiple components which are distributed to the different locations. Those locations might be on your networks or systems. In Java, everything is treated as an object. When two Java components want to communicate with each other, the data needs to be exchanged between components. When these java components want to communicate with each other over the network at that time there is a possibility of data loss, when you want to transfer data over the network first maintains the state of that object and transfer the object between components. Serialization is defined for this purpose, and Java components use this protocol to transfer objects.

Serialization is the process of saving an object's state. Deserialization is the retrieving those bytes into respective object. When we serialize or deserialize it is in the form of streams. This is also called as object serialization. To do Object Serialization we have two interfaces:

- **Serializable Interface**
- **Externalizable Interface**

To serialize an object it should be ensured that the class of the object implements `java.io.Serializable` interface.

In following example Employee information of employees needs to be serialized, so Employee class implements Serializable interface

```
package com.ser;
import java.io.*;
public class Employee implements Serializable
{
    public int eid,sal;
    public String ename;
    Employee() {}
        . . . //some code here
}
```

Object Serialization

- Object Serialization Steps:
 1. Create file output stream.
 2. Create object output stream
 3. Connect them
 4. Call writeObject()
- Object Deserialization Steps :
 1. Create file input stream.
 2. Create object input stream.
 3. Connect them
 4. Call readObject()

Following application serialized the employee information inside a file and desterilized it.

Code Example

```
//Employee.java
//This is an Employee class to which implements
Serializable interface.
package com.seed.io.ser;
import java.io.*;
public class Employee implements Serializable
{
    String enm;
    int no, sal;
    .
    .
    .
    //Assume required employee code is available here
}
//Serial.java
```

```
//This class serializes the employee object into a file.  
package com.seed.io.ser;  
import java.io.*;  
public class Serial  
{  
    public static void main(String args[])  
    {  
        FileOutputStream fos;  
        ObjectOutputStream oos;  
        try  
        {  
            Employee emp=new Employee("Rohan", 20, 10000);  
            fos=new FileOutputStream("serial.txt");  
            oos=new ObjectOutputStream(fos);  
            oos.writeObject(od);  
            oos.close();  
        }  
        catch(Exception e)  
        {  
            e.printStackTrace();  
        }  
    }  
}  
//DeSerial.java  
//This class deserializes the employee object from the same file.  
import java.io.*;  
public class Deserial  
{  
    public static void main(String args [])  
    {  
        FileInputStream fis;  
        ObjectInputStream ois;  
        try  
        {  
            Employee od;
```

```

        fis=new FileInputStream("serial.txt");
        ois=new ObjectInputStream(fis);
        od= (Employee) ois.readObject();
        od.display ();
        ois.close ();
    }
    catch(Exception e)
    {
        e.printStackTrace ();
    }
}
}

```

Why transient?

When the object is serialized default all the attributes in that object get serialized. Some time not required to serialize some attributes because might not have requirement to persist these attributes. In that scenario just declare such attributes are transient. If the attribute is transient than it will not persisted.

Code Example

Following code snippet declare name attribute as a transient:

```

import java.io.*;
public class Employee implements Serializable
{
    transient static int count;
    int no, sal;
    . . .
}

```



Tech App

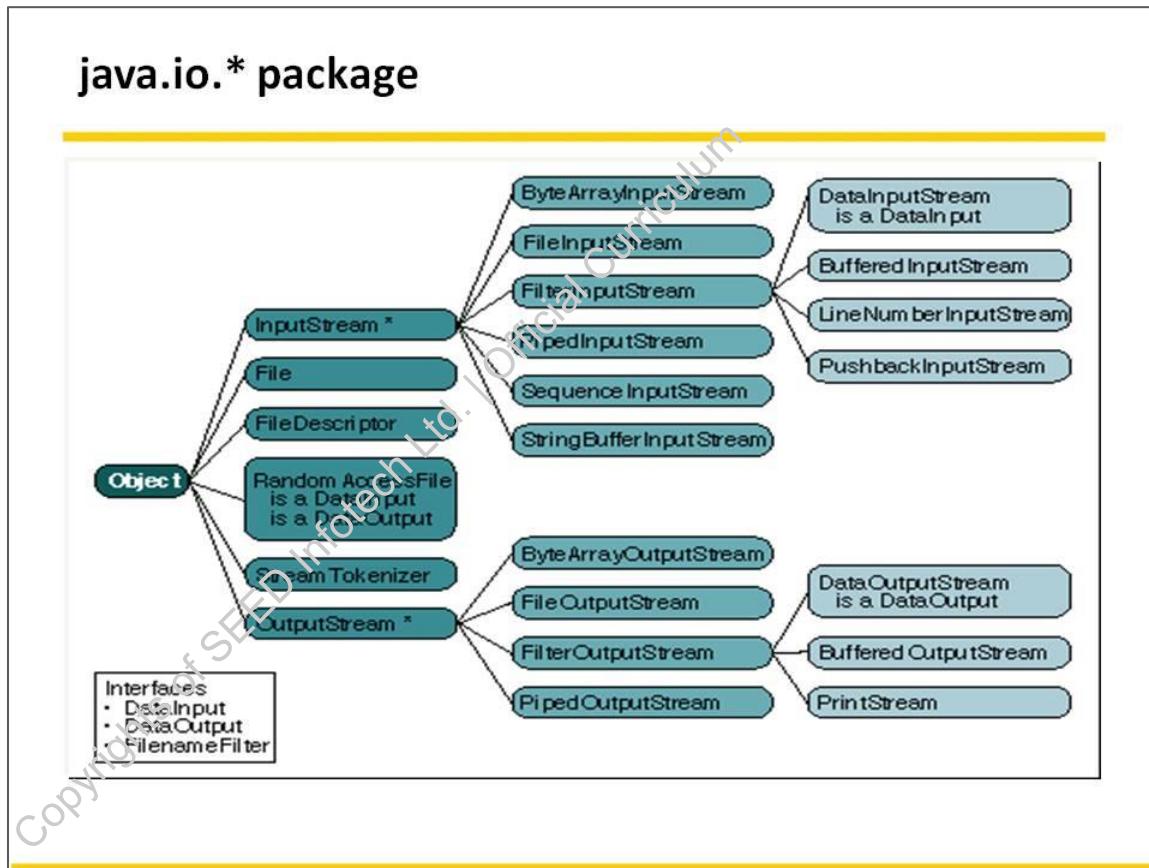
If an object contains contained object as an attribute then all contained objects have to implement Serialization interface otherwise NotSerializableException is thrown by application.



Tech App

Externalizable Interface- To perform custom marshalling and unmarshalling.for the extra information please refer

<http://www.javabeat.net/tips/11-using-the-externalizable-interface.html>



Java provides an extensive library populated with a set of classes and interfaces under `java.io` package.

The classes provided under the `java.io` package are used to deal with various I/O devices to perform different I/O operations such as reading or writing data to an I/O device or even deleting some data from some I/O device.

But problem related to these classes is that, these classes do not have much functionality i.e. a single class is not capable of performing a big operation. So they are layered. A number of features by layering one class over another.

The above diagram gives a detailed hierarchical view of the important classes and interfaces under the `java.io` package.

Java's stream-based I/O is built upon four abstract classes: `InputStream`, `OutputStream`, `Reader`, `Writer`. They are used to create several concrete stream subclasses. Although your programs perform their I/O operations through concrete subclasses, the top-level classes define the basic functionality common to all stream classes.

Difference between Stream classes and Reader/Writer classes are as follows:

Since byte-oriented streams are inconvenient for processing information stored in Unicode (Unicode uses two bytes per character), there is a separate hierarchy of classes for processing Unicode characters that inherit from the abstract Reader and Writer classes. These classes have read and write operations that are based on 2-byte Unicode characters rather than on single-byte characters.

InputStream and OutputStream are designed for byte streams. Reader and Writer are designed for character streams. The byte stream classes and the character stream classes form separate hierarchies. These 4 classes are also defined as root classes for input/output.

Technique for Using java.io Package

1. Choose a suitable input or output stream class.
2. Create the object of the chosen class.
3. While creating the object associate it with the source or destination in the constructor (depending on whether it is input or output object).
4. Once the object is obtained invoke a suitable read or write method using the object.

Though `java.io` package provides a set of classes to perform I/O operations, there are certain steps which should be followed for making use of these classes.

Following are some steps which will help to make proper and efficient use of these classes.

1. The device do you want to perform I/O operations.
2. Search the appropriate classes provided in the `java.io` package for performing I/O operations with the selected device.
3. Use the methods provided in that class the object of that class.
4. Find appropriate read and write methods that will help to read/write data from/to to the device.
5. Consider an example. In which file is a device used to perform I/O operations.
6. Java has provided the `FileInputStream/FileOutputStream` classes in `java.io` package.

7. To read data from the file we have to open that file using `FileInputStream` by creating an object of `FileInputStream`. Similarly, to write data to the file we have to open `FileOutputStream` on the file (in which we want to write data) by creating an object of `FileOutputStream`.
8. There is a `read()` method in `FileInputStream` that returns an int and a corresponding `write()` method in `FileOutputStream` which accepts an int. Constructive, interleaved use of these two methods will enable reading and writing.

System

- Three static I/O objects have already been created by the time main() method gains control.
- All 3 are public static members of System class.
 - System.in
 - System.out
 - System.err
- Systems associated with these objects provide communication channels between a program and a particular file or device.

Most of the time in our java code we make use of the `System` class but we never instantiate it. Why?

Actually we can't instantiate the `System` class. It follows the singleton design pattern.

(Hint: In singleton design pattern all the constructors of that class are private and we can't make an object of that class.)

The `System` class contains several useful class fields and methods.

The `System` class provides you with different facilities. Some of them are as follows:-

- Standard input, standard output, and error output streams.
- Access to externally defined properties and environment variables.
- A means of loading files and libraries.
- A utility method for quickly copying a portion of an array and so on.

Following code snippet explains the simple methods of a System class:

```
package com.seed.io;
import java.io.*;
class ReadFromConsole
{
    public static void main (String args[])
    {
        StringBuffer sb = new StringBuffer();
        char c;
        try
        {
            while ((ch = (char) System.in.read ()) != '\n')
            {
                sb.append(c);
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        String s = new String (sb);
        System.out.println(s);
    }
}
```

Scanner

- It is a new Java 5 feature to get input from the user.
- A simple text scanner which can parse primitive types and strings using regular expressions.
- A Scanner breaks its input into tokens using a delimiter pattern.
- By default, whitespace is the delimiter.

```
Scanner sc = new Scanner( System. in );
int i = sc.nextInt();
```

Java 5 introduces a new class called Scanner class. It gives you facility to take input from user in an efficient way. It can allow the read values from user in a various type.

A simple text scanner which can parse primitive types and strings using regular expressions. A Scanner breaks its input into tokens using a delimiter pattern, which by default matches whitespace. The resulting tokens may then be converted into values of different types using the various next methods.

Scanner API

Following are some important methods of Scanner class:

Method	Description
Scanner(File source)	Constructs a new Scanner that produces values scanned from the specified file.
Scanner(File source,	Constructs a new Scanner that produces

String charsetName)	values scanned from the specified file.
Scanner(InputStream source)	Constructs a new Scanner that produces values scanned from the specified input stream.
Scanner(InputStream source, String charsetName)	Constructs a new Scanner that produces values scanned from the specified input stream.
Scanner(Path source)	Constructs a new Scanner that produces values scanned from the specified file.
Scanner(Path source, String charsetName)	Constructs a new Scanner that produces values scanned from the specified file.
Scanner(Readable source)	Constructs a new Scanner that produces values scanned from the specified source.
Scanner(ReadableByteChannel source)	Constructs a new Scanner that produces values scanned from the specified channel.
Scanner(ReadableByteChannel source, String charsetName)	Constructs a new Scanner that produces values scanned from the specified channel.
Scanner(String source)	Constructs a new Scanner that produces values scanned from the specified string.
void close()	Closes this scanner.
Pattern delimiter()	Returns the Pattern this Scanner is currently using to match delimiters.
String findInLine(Pattern pattern)	Attempts to find the next occurrence of the specified pattern ignoring delimiters.
String findInLine(String pattern)	Attempts to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters.

boolean hasNext()	Returns true if this scanner has another token in its input. Note: <code>hasNext</code> methods have different versions according to our primitive types. Like : <code>hasNextInt()</code> : Returns true if the next token in this scanner's input can be interpreted as an int value in the default radix using the <code>nextInt()</code> method. <code>hasNextXXX()</code>
String next()	Finds and returns the next complete token from this scanner.
void remove()	The remove operation is not supported by this implementation of Iterator.
Scanner skip(Pattern pattern)	Skips input that matches the specified pattern, ignoring delimiters.
Scanner skip(String pattern)	Skips input that matches a pattern constructed from the specified string.

Code Example

Following code snippet explains the simple methods of a Scanner class:

```
// Enter the gross salary and read it either in integer or double format.
package com.seed.io;
import java.io.*;
import java.util.Scanner;
public class ScannerInput
{
    public static void main (String [] args)
    {
        Scanner in = new Scanner (System. in);
        System.out.println ("Enter your gross income: ");
    }
}
```

```
if (in.hasNextInt ())  
{  
    System.out.println ("You entered" + in.nextInt ());  
}  
else if (in.hasNextDouble ())  
{  
    System.out.println ("You entered" + in.nextDouble  
());  
}  
else  
    System.out.println ("Token not an integer or a real  
value.");  
}
```

Output

```
Enter your gross income:  
10000  
You entered 10000  
  
Enter your gross income:  
12000.00  
You entered212000.0  
  
Enter your gross income:  
java  
Token not an integer or a real value.
```

Console

- It is a new Java 6 feature to get standard streams.
- Console is generally used for secure password entry.
- Methods to access the character-based console device.
- It returns unique instance of Console with current Virtual Machine.

```
String input = System.console().readLine();
```

Java 6 is come with the new feature called `Console` class. Actually it is old command-line functionality where you have to handle data using `System.in` and `System.out`. The `java.io.Console` class is a simple extension of command-line and it is available through the `System.console()` method. Methods to access the character-based console device, if any, associated with the current Java virtual machine.

Console API

Following are some important methods of `Console` class:

Method	Description
<code>void flush()</code>	Flushes the console and forces any buffered output to be written immediately.
<code>Console format(String fmt,</code>	Writes a formatted string to this console's output stream using the specified format

Object... args)	string and arguments.
Console printf(String format, Object... args)	A convenience method to write a formatted string to this console's output stream using the specified format string and arguments.
Reader reader()	Retrieves the unique Reader object associated with this console.
String readLine()	Reads a single line of text from the console.
String readLine(String fmt, Object... args)	Provides a formatted prompt, then reads a single line of text from the console.
char[] readPassword()	Reads a password or passphrase from the console with echoing disabled
char[] readPassword(String fmt, Object... args)	Provides a formatted prompt, then reads a password or passphrase from the console with echoing disabled.
PrintWriter writer()	Retrieves the unique PrintWriter object associated with this console.

Code Example

Following code snippet explains the simple methods of a Console class:

Accepts the name and password using Console class and print name.

```
//ConsoleDemo.java
public class ConsoleDemo
{
    public static void main (String[] args)
    {
        Console console = System.console () ;//create console
        if(console != null)
        {
            //accept name from console
        }
    }
}
```

```
String name = console.readLine ("[Please Provide  
Your Name]: ");  
//accept password from console  
char[] passdata = console.readPassword ("[Please  
Input Your Password]: ");  
console.printf ("Hello"+ " "+ "Mr."+name);  
}  
}  
}
```

Output

```
[Please Provide Your Name]: Nik  
[Please Input Your Password]:  
Hello Mr.Nik
```

Regular Expression in Java

- regex means “Regular expressions” which is available in a `java.util.regex` package.
- A kind of language within a language.
- Designed to help programmers with the searching tasks.
- Regex engine search textual data using instructions that are coded into expressions.
- Regex language is used to create expressions.

In an application number of times text processing like word searching, email validations, xml integrity or date formats etc. is assumed to be done. All this often involves pattern matching. To do all these kinds of operations java uses `StringTokenizer` class with many `charAt`, `substring` methods to read through the characters or tokens to process the text. This often leads to complex coding.

So, a standard package called `java.util.regex` is used to enable a new concept called ‘Regular Expressions’.

This functionality use meta characters, which gives you regular expressions flexibility. Regular Expression performed in following some scenarios:

- Simple word replacement
- Email validation
- Removal of control characters from a file
- File searching

To use Regular Expression in your applications we require two main classes called:

Matcher and Pattern

Matcher

An engine that performs match operations on a character sequence according to Pattern requirement. A matcher is created from a pattern by invoking the pattern matcher method. A Matcher has three types of operations:

- The `matches` method attempts to match the entire input sequence against the pattern.
- The `lookingAt` method attempts to match input sequence, starting at the beginning, against the pattern.
- The `find` method scans the input sequence looking for the next subsequence that matches the pattern.

Following are some important methods of Matcher class:

Methods	Description
<code>int end()</code>	Returns the offset after the last character matched.
<code>boolean find()</code>	Attempts to find the next subsequence of the input sequence that matches the pattern.
<code>boolean find(int start)</code>	Resets this matcher and then attempts to find the next subsequence of the input sequence that matches the pattern, starting at the specified index.
<code>String group()</code>	Returns the input subsequence matched by the previous match.
<code>String group(int group)</code>	Returns the input subsequence captured by the given group during the previous match operation
<code>String group(String name)</code>	Returns the input subsequence captured by the given named-capturing group during the previous match operation.

boolean lookingAt()	Attempts to match the input sequence, starting at the beginning of the region, against the pattern.
boolean matches()	Attempts to match the entire region against the pattern.
Pattern pattern()	Returns the pattern that is interpreted by this matcher.
int start()	Returns the start index of the previous match.

Pattern

A compiled representation of a regular expression. A regular expression, specified as a string, must first be compiled into an instance of this class. The resulting pattern can then be used to create a Matcher object that can match arbitrary character sequences against the regular expression. All of the state involved in performing a match resides in the matcher; so many matchers can share the same pattern.

Following are some important methods of Pattern class:

Methods	Description
static Pattern compile(String regex)	Compiles the given regular expression into a pattern.
static Pattern compile(String regex, int flags)	Compiles the given regular expression into a pattern with the given flags.
int flags()	Returns this pattern's match flags.
Matcher matcher(CharSequence input)	Creates a matcher that will match the given input against this pattern.

String pattern()	Returns the regular expression from which this pattern was compiled.
static String quote(String s)	Returns a literal pattern String for the specified String.
String [] split(CharSequence input)	Splits the given input sequence around matches of this pattern.

Code Example

Following code snippet explains some important methods of Pattern and Matcher class.

Find out the specific pattern from given String and replace it with other String.

```
package com.seed.io.regex;
import java.util.regex.*;
class RegExpr
{
    public static void main(String[] args)
    {
        String str="Jon Jonathan FrankJon Ken Todd";
        Pattern pat=Pattern.compile ("Jon.*? ");
        Matcher mat=pat.matcher(str);
        System.out.println ("Original sequence: "+str);
        str=mat.replaceAll("Eric ");
        System.out.println("Modified sequence: "+str);
    }
}
```

Output

```
Original sequence: Jon Jonathan FrankJon Ken Todd
Modified sequence: Eric Eric FrankEric Ken Todd
```



Read about details of regex in Appendix of this book.

Additional Reading

AutoClosable interface



- A new interface `java.lang.AutoCloseable` which define a single method :
 - `public void close() throws Exception`
- `java.io.Closeable` is derived from `java.lang.AutoCloseable`.
- It supports to java's new Project Coin feature automatic resource management (ARM) blocks .
- Closes this resource, relinquishing any underlying resources.
- This method is invoked automatically on objects managed by the try-with-resources statement.

The try-with-resources is also called as ARM (Automatic Resource Management). A resource is as an object that must be close after the program is finished. ARM or try-with-resources checks that each resource is closed at the end of the statement. If the object implements interface called `java.lang.AutoCloseable`, which includes all objects which implement `java.io.Closeable` can be used as a resource.

Following code snippet is an example of ARM (Automatic Resource Management) implementation of AutoCloseable Interface.

```
try ( FileInputStream in = new FileInputStream  
("source.txt");  
     FileOutputStream out = new FileOutputStream  
("dest.txt");  
 )  
{  
     int c;  
     while ((c=in.read ()) != -1 )  
         out.write(c); }
```

Non-blocking IO (NIO)

- `java.nio.*` package introduced a Java “new” I/O library-The Java NIO called Non-blocking IO.
 - It supports a channel-based approach I/O operations.
 - It built on two fundamental items-channels and buffers.
 - Send buffer into the channel.
 - The channel either retrieves or loads data into the buffer.
 - Used in writing high-performance, large-scale applications.

The `java.nio` (new input/output) package, which is introduced in Jdk1.4 usually called NIO i.e. non-blocking IO.NIO, provides high-speed, block-oriented I/O operations. The NIO supports to some important features like:

- Buffers for data of primitive types.
- Character set encoders and decoders.
- A pattern-matching facility.
- Channels, a new primitive I/O abstraction.
- A file interface that supports lock.
- Asynchronous I/O.

In Java, I/O carried data using a stream. All I/O types pass single byte at a time through an object called a Stream. When stream transfer from source to destination its turning objects into bytes and then back into objects.

NIO is also having same role and purpose as original IO, but it follows different way to transfer data called block I/O instead of stream I/O.

NIO was created to allow programmers to implement high-speed I/O without writing custom native code. NIO performs time consuming activities so, naturally it increases speed.

A channel represents an open connection to an entity such as program component, file or a network socket that is capable of performing one or many distinct I/O operations such as reading and writing. NIO channels can be asynchronously closed. If a thread is blocked in an I/O operation on a channel, another thread can interrupt that blocked thread.



Additional Reading

Read about details of NIO in Appendix of this book.

StringTokenizer

- Tokenizing is a process of breaking the string into a small pieces.
- The string tokenizer class allows an application to break a string into tokens.
- Tokenizing is composed on two things :tokens and delimiters.

```
StringTokenizer st = new StringTokenizer("this is a test");
while (st.hasMoreTokens())
{
    System.out.println( st.nextToken());
}
```

Output:
this
is
a
test

StringTokenizer is a class which breaks the string into tokens. Tokens are based on delimiters. Delimitiers might be your comma, number, white space etc.

StringTokenizer API

Following are the some important methods of StringTokenizer:

Methods	Description
StringTokenizer(String str)	Constructs a string tokenizer for the specified string.
StringTokenizer(String str, String delim)	Constructs a string tokenizer for the specified string.
StringTokenizer(String str, String delim, boolean returnDelims)	Constructs a string tokenizer for the specified string.

int countTokens()	Calculates the number of times that this tokenizer's nextToken method can be called before it generates an exception.
boolean hasMoreElements()	Returns the same value as the hasMoreTokens method.
boolean hasMoreTokens()	Tests if there are more tokens available from this tokenizer's string.
Object nextElement()	Returns the same value as the nextToken method, except that its declared return value is Object rather than String.
String nextToken()	Returns the next token from this string tokenizer.
String nextToken(String delim)	Returns the next token in this string tokenizer's string.

Code Example

Following code snippet explains the simple methods of a StringTokenizer class:

```
//Note: tokenized the string using delim "=";
import java.util.StringTokenizer;
class StringTokenizerDemo
{
    static String in = "title=Java-Samples;" +"author=John
J;";
    public static void main(String args[])
    {
        StringTokenizer st = new StringTokenizer (in, "=");
        while(st.hasMoreTokens())
        {
            String key = st.nextToken();
        }
    }
}
```

```
String val = st.nextToken ();
System.out.println (key + "\t" + val);
}
}
}
```

Output

```
title Java-Samples
author John J
```



Tech App

StringTokenizer is a deprecated class.

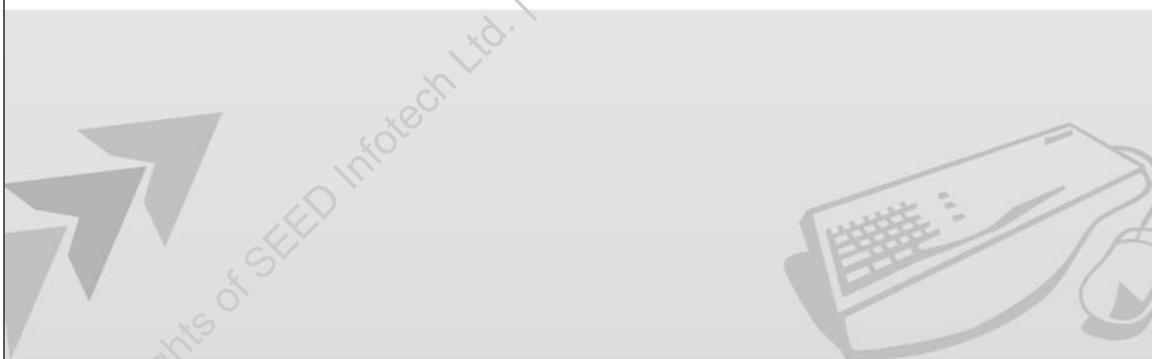


Tech App

Similar to StringTokenizer; StreamTokenizer is a class that takes entire stream as token. The StreamTokenizer can recognize identifiers, numbers, quoted strings, and various comment styles.

Chapter - 8

Java Sockets



This chapter covers socket programming using Java APIs. Related concepts like TCP/IP architecture, sockets, and ports are included along with socket programming examples using TCP and UDP socket classes.

Objectives

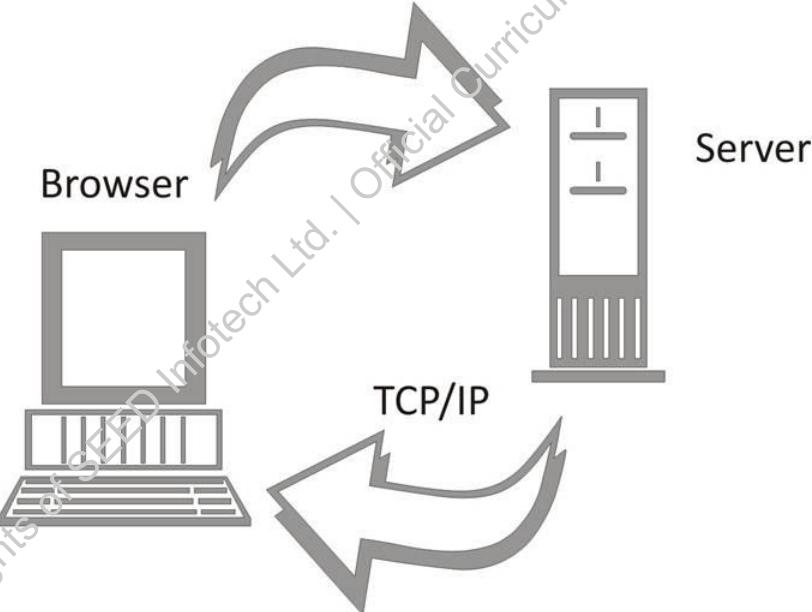
At the end of this chapter you will be able to:

- Describe OSI model.
- Compare OSI with TCP/IP architecture.
- State IP address and classes.
- List various protocols in TCP/IP stacks.
- Use `InetAddress` class in socket programming.
- Identify the need of sockets in point to point communication.
- Define Port and list well-known ports.
- Define Socket.
- List algorithm for building remote communication using sockets.

- Construct a program communication using TCP based Socket API.
- Use TCP/IP commands.
- Differentiate between TCP-socket and UDP-socket.
- Construct a program using UDP based Socket API.

Copyrights of SEED Infotech Ltd. | Official Curriculum

And TCP/IP was born!



In the era of client-server based applications development, applications were split into two parts, front end and back end. Front end was usually a GUI based programming language like Visual Basic, Power Builder etc. and back end was a database server like Oracle, SQL server etc. A robust communication mechanism was needed between these two to avail the advantages of client-server based computing.

With the advent of internet and its gaining popularity, issues became more complex. To address that HTTP (Hyper Text Transfer Protocol) was created. The data which would be sent in the form of text needed to be formatted for presentation purpose. Otherwise reading it would not be easy and interesting. This is how HTML (Hyper Text Markup Language) came into being.

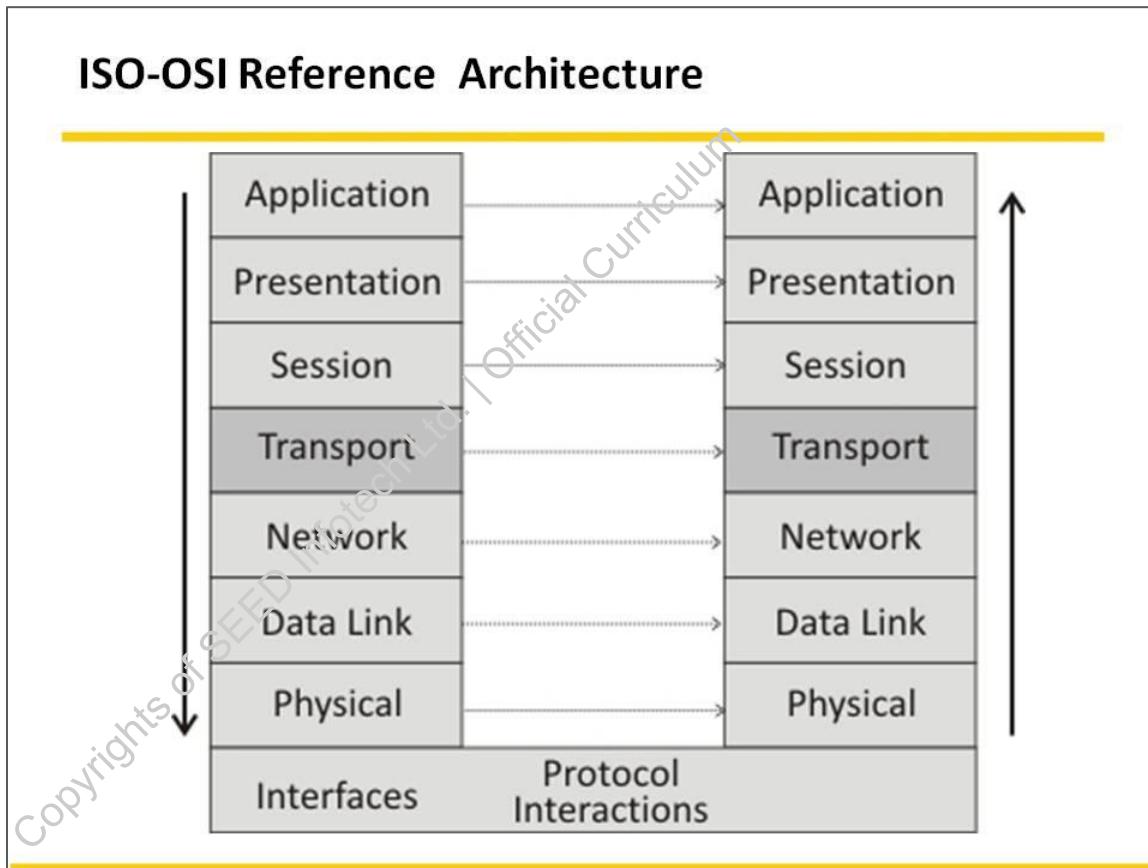
To carry data in the form of text, there has to be a vendor neutral carrier that would provide a good transport mechanism. This is where TCP/IP was introduced. In this architecture, HTTP is an application level protocol over TCP/IP stack. HTTP is the service user and TCP/IP is the service provider.

The basic objective of designing TCP/IP was to develop a protocol that was independent of everything – hardware, OS, vendor, etc. As a result, TCP/IP can be implemented on top of virtually any hardware networking technology.

In an internet based applications, the browser is the http client that sends an http request and the http web server responds to this request and page is displayed. The response is sent over a text packet which is carried over TCP/IP.

More than a quarter century after the internet was born, this model was developed. Therefore it has all the best operational functionalities that can be.

Applications based on TCP/IP are divided into two sections' layers; frontend, backend. Normally the frontend applications are GUI based, like Visual Basic (VB), PowerBuilder. The backends are typically databases.



A distributed application is one in which multiple components or programs reside on different machines and then they communicate with each other to give the user the complete functionality. To develop communication between various components of distributed application, ISO-OSI model was created as reference architecture. It divides the communication process into seven groups, called layers with each layer having its own standard protocols.

Layer	Description
Physical layer	This layer consists of all hardware that will be required to connect the two machines.
Data link layer	This layer does creation of smaller packets of data and error handling between two directly connected nodes.
Network layer	This layer decides the path which a packet can take. This is called routing.
Transport layer	This layer handles end-to-end acknowledgement and makes data transfer more reliable.

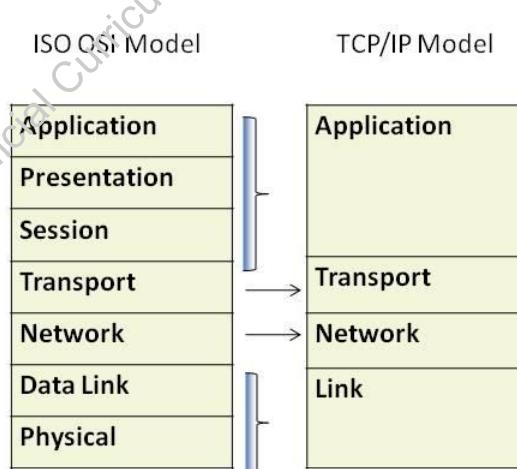
Session layer	This layer maintains session information which is helpful for session-aware applications. For example, restartable download of file
Presentation layer	This layer takes care of different data formats sent between source and destination to preserve the meaning.
Application layer	This layers hosts all the applications created using rest of the 6 layers like email, file transfer etc.



Focus on understanding of peculiar functionalities of each layer.

TCP/IP Architecture

- TCP/IP is global data communication model
- TCP/IP is a 4 layer protocol stack that can be mapped to five of seven layers of OSI Model



The four layers of the TCP/IP model can be mapped to the seven layers of the ISO OSI model as shown above.

TCP/IP Protocol Stack

Layer	Description
Link layer	The physical and data link layers in the OSI model are combined and called the link layer. This layer constitutes the physical equipment such as twisted pair cables, equipment and system for signaling such as Ethernet, ISDN, Modems, Routers, arc net, fiber optics, RS232, etc.
Network layer	The data is packaged, addressed and routed to network destinations. The Internet Layer defines IP address system and the routing schemes for navigating packets from one IP address to another. Some examples of Internet Layer protocols are IPv4, IPv6, etc.

Transport layer	The standards of data transport are decided in this layer. The Transport Layer has the connection protocols such as TCP, UDP. This layer not only enables opening and maintaining connections but also ensures that the data packets are received properly.
Application layer	Session management, presentation layer, byte swapping, etc is done here. This is the topmost layer where the protocols such as SMTP, FTP, DHCP, POP3, SMTP, HTTP, DNS etc operate.

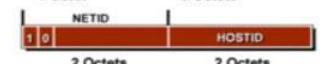


Interview Tip

Focus on understanding of peculiar functionalities of each layer and the protocols designed to work at each of them.

IP Addressing

- IP addresses have been divided into the following classes:

Class	First Octet Range	Max Hosts	Format
A	1-126	16M	
B	128-191	64K	
C	192-223	254	
D	224-239	N/A	
E	240-255	N/A	

When the client and server interact with each other, the locations of both (or addresses) have to be known to each other. They must have a unique identification number which can be used to identify them. TCP/IP uses a mechanism called IP addresses for this.

IPv4

Here an IP address is 32 bit long and is composed of 4 octets each of 8 bits. Therefore, there can be 2^{32} combinations of numbers as addresses of machines. It is represented as a combination of decimal numbers separated by dots. For example, 192.168.2.13

All the addresses are divided into classes as follows

Class A

In this class, 0 to 23 bits are used to give addresses to the host machines. So there can be 2^{24} machines. The value in the most significant bit is fixed as 0 and the remaining 2⁷ bits in the last octet are the number of networks. Each of these

networks can have theoretically 2^{24} machines. These addresses are allocated to large sized organizations.

Class B

To avoid conflicts in the IP addresses, the two most significant bits are fixed as 1 and 0 and the remaining 2^{14} bits are used to give the network ID. 0 to 15 bits are used to name the host machines. So there can be 2^{16} machines in each of the 2^{16} networks. These addresses are allocated to medium sized organizations.

Class C

In this class, the first octet, that is 0 to 7 bits are used for the host ID and the last three octets are used for network ID. The last 3 most significant bits are fixed as 110 and the remaining 2^{21} bits are used for network ID. This means that there will be 2^8 machines in each of the networks. These addresses are allocated to small sized organizations.

IPv6

IP v6 is a more recent protocol, offering a much larger address pool than IPv4. It is represented as a 128 bit number. IPv6 is designed to solve the problems of IPv4. It does so by creating a new version of the protocol which serves the function of IPv4, but without the same limitations of IPv4. The differences between IPv6 and IPv4 are in five major areas: addressing and routing, security, network address translation, administrative workload, and support for mobile devices. IPv6 also includes an important feature: a set of possible migration and transition plans from IPv4.

Special IP address

Following are some of the special IP addresses reserved for peculiar usage.

IP address	Significance
127.*.*.* Typically 127.0.0.1	This is called loopback address. When used as target IP address, it represents the same machine and packet is looped back. Used for testing purpose
192.3.4.0	0 in the place of host-id represents the network address. These are used by routers

192.3.4.255	255 in the place of host-id indicates it is a broadcast address.
-------------	--



Focus on understanding of IP address structure and its need. You should be able to find the class of given IP address.

Domain Name System (DNS)

To locate anything uniquely it is fine to have numbers while dealing with computers. For humans it is difficult to remember and use number lists. To make this task easy, domain name system was created. Here each IP address is mapped with a meaningful name or string like www.seedinfotech.com, microsoft.com or java.sun.com. These strings are called domain names. When any website is accessed using the name like www.seedinfotech.com, the DNS system maps the IP address associated with it and then communication happens using the IP address.



For every new website you host on the internet, a unique IP address should be sought from the interNIC. These unique IP addresses identify your website uniquely on the internet. You need to get unique domain name and an IP address. List websites which give this.



Hosts file was used before DNS for mapping names to IP addresses. Study how you can use hosts file to create this mapping.

The InetAddress Class

- java.net.InetAddress class represents IP address
- It converts numeric addresses to names & names to addresses
- Used by Socket & ServerSocket to identify hosts
- No public InetAddress () Constructors-Arbitrary addresses may not be created
- Addresses created must be checked with DNS

```
public static
InetAddress
getByName(String
host) throws
UnknownHostException
```

```
public static
InetAddress[]
getAllByName(String
host) throws
UnknownHostException
```

```
public static
InetAddress
getLocalHost()
throws
UnknownHostException
```

InetAddress class represents the IP address. It does not have a constructor. To create object one of its static methods needs to be invoked. Most of these methods use DNS Server configured for the system.

Some of the important methods of InetAddress class

Method	Description
static InetAddress getAllByName(String host)	Given the name of host, it returns array of its IP addresses based on configuration.
static InetAddress getByAddress(byte[] addr)	Returns the InetAddress object given the raw IP address.
static InetAddress getByAddress(String host, byte[] addr)	Creates IP address based on the host name given IP address,. No name service is checked.

static InetAddress getByName (String host)	Returns IPaddress of a host, given host name.
static InetAddress getLocalHost ()	Returns a hash code for this IP address.

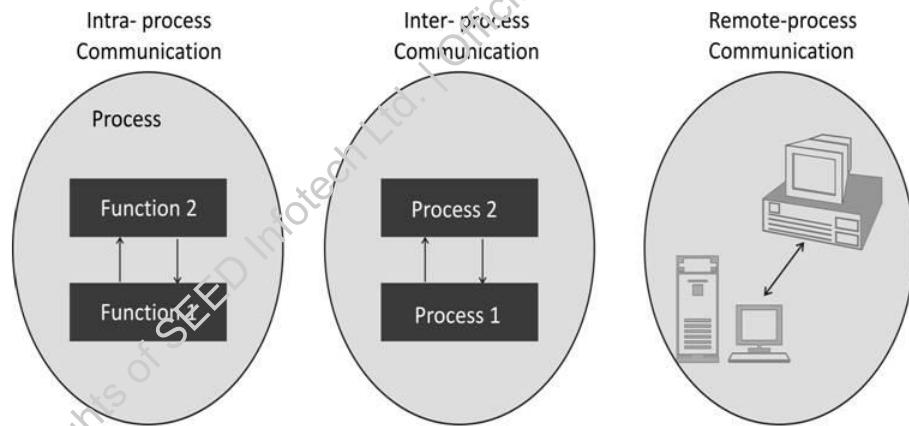


Best Practice

Constructor is a standard method to create an object. Sometimes a static method is created for this purpose to exercise more control on object creation like Singleton pattern. This static method is called Factory method design pattern.

Program to Program Communication

- Communication can be of 3 types:



At the heart of distributed systems is program to program communication. There are three different ways in which this can be done based on location of interacting programs or components.

Intra-process communication

When functions within the same process interact with each other, it is said to be intra-process communication. For example, a function within a program calls another function. This is done by using stack as a mechanism.

Inter-process communication

When two processes or programs running on the same computer interact with each other, it is inter-process communication. For example, copying text from one program and pasting it into another using clipboard.

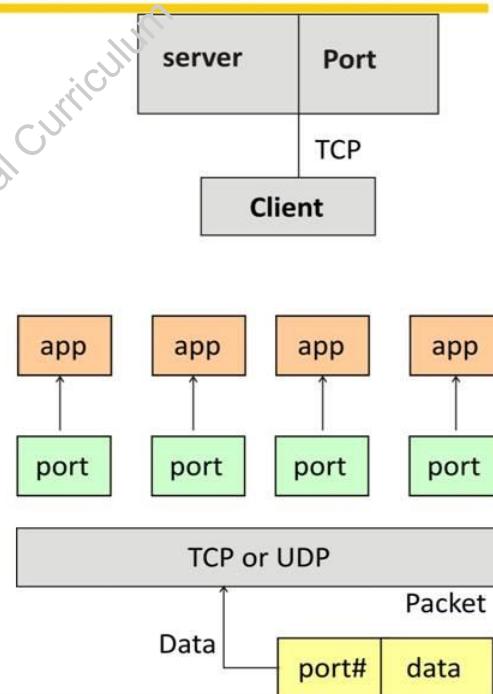
Remote process communication

When two programs on two different computers want to communicate with each other it is remote process communication or RPC. Here the problem becomes

more complex as one need to identify a program running on a particular computer in the whole distributed system. IP address being unique can be used to identify a particular computer on the system. Still a mechanism is needed to locate a particular program running on that computer identified by its IP address. This mechanism is called port.

Port

- Port is a Logical number assigned to process running on server
- +ve number
- Reserved ports
 - 21 – FTP
 - 23 – Telnet
 - 25 – SMTP
 - 80 – HTTP
 - 109 - POP



Port is a 16 bit number used to uniquely identify a program running on a particular machine. It is same as delivering a postal letter to a specific person at the given address. Port numbers range from 0 to 65,535. Ports are represented by 16 bit numbers. Port numbers from 0 to 1023 are reserved ports that are restricted for specific functionality and cannot be used by the developers. Some ports are used for commonly used applications and are called as well known ports. It is recommended that these ports also should be avoided by developers.

For example,

- Port 21 is reserved for File Transfer Requests in an application using FTP protocol.
- Port 80 is reserved to receive Hypertext requests only.

Having knowledge of ports and socket and related commands like netstat would help you in application deployment related issues. For example, your server is not getting started due to some port related issues such as your previous server instance is running on the same port and you are trying to start a new one on the same.



Tech App

Socket Communication

- Socket [IP address + Port Number]
- Bidirectional Stream.
- Endpoint for Transport layer connection.
- Used for implementation of Remote Procedure Call (RPC).

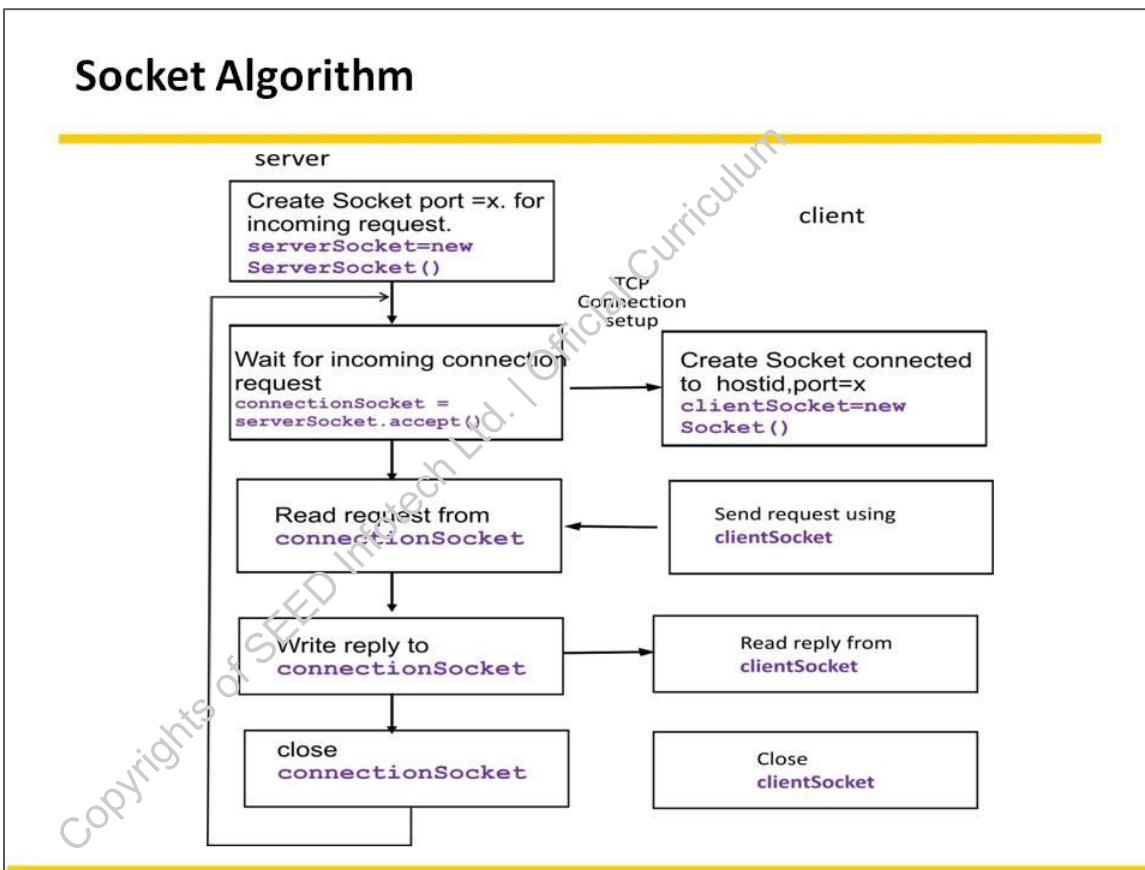
In a remote procedure call (RPC) one program called “client” invokes a procedure or method of other program situated on the remote computer. To facilitate these communications two end points need to be defined. A specific computer can be identified uniquely by its IP address and a program running on that computer by a port number. This combination uniquely identifies a program running on a specific computer. It is called a socket. Socket is an abstraction provided to the application programmer. It works like a connection or stream connected to the remote computer on which data can be exchanged in both the directions.

Socket is a convenient abstraction and represents a connection point into a TCP/IP network, much like the electrical sockets, which provide a connection point for electrical appliances. When two computers want to converse, each uses a socket. One computer is termed the server - it opens a socket and listens for connections. The other computer is termed the client - it calls the server socket to start the connection. To establish a connection, all that's needed is a server's destination address and port number. Concept of socket first emerged in early UNIX programming and became common primitive for communication. It is very easy to

plug-in socket and become part of communication which follows the standard protocol.

Each computer in a TCP/IP network has a unique address and ports represent individual connections within that address. Each port within a computer shares the same address, but data is routed within each computer by the port number. When a socket is created, it must be associated with a specific port - this process is known as binding to a port.

Copyrights of SEED Infotech Ltd. | Official Curriculum



There are two types of sockets: connection-oriented and connectionless. Connection-oriented is always based on TCP and connectionless sockets are based on UDP. TCP sockets guarantee data arrival in specific order. Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request. Sockets implement stream based communication. Stream is sequence of bytes. Sockets are required to be established on both the ends of communication in order to send or receive data bytes depending on the role played by the application at the respective end.

Socket Algorithm

Steps for writing server:

1. Create ServerSocket: create an instance of ServerSocket object and initialized the port.
2. Wait for connection: server invokes the accept() method on a serversocket object. This method waits for incoming connection request.

3. Get I/O stream: once the connection established then Input and Output stream corresponding to the socket object has to be obtained.
4. Read and Write data: read and write operation in between client Socket and ServerSocket using streams.
5. Close the connection: Once the client stops sending request close the application by calling `close()` method.

Steps for writing client:

1. Create Socket: Create an instance of `Socket` class. The socket at client side only needs to know the host and the port where the server is listening.
2. Get I/O stream: Once the connection is established then input and output stream corresponding to the socket object has to be obtained. It's communicating with server.
3. Read/Write data: read and write operation in between client `Socket` and `ServerSocket` using stream.
4. Close the connection: Finally close the connection by calling `close()` method.

Socket API

Following table lists some of the important methods from **Socket** and **ServerSocket** classes:

TCP Sever Socket

<code>public ServerSocket()</code>
Creates an unbound server socket.
<code>public ServerSocket(int port)</code>
Creates a server socket, bound to the specified port.
<code>public ServerSocket(int port, int backlog)</code>
Creates a server socket and binds it to the specified local port number, with the specified backlog.

`accept()`

Listens for a connection to be made to this socket and accepts it.

TCP Client Socket

`Socket()`

Creates an unconnected socket

`Socket(InetAddress address, int port)`

Creates a stream socket and connects it to the specified port number at the specified IP address.

`Socket(InetAddress address, int port, InetAddress localAddress, int localPort)`

Creates a socket and connects it to the specified remote address on the specified remote port.

`Socket(String host, int port)`

Creates a stream socket and connects it to the specified port number on the named host.

`Socket(String host, int port, InetAddress localAddress, int localPort)`

Creates a socket and connects it to the specified remote host on the specified remote port

`void bind(SocketAddress bindpoint)`

Binds socket to a local address

`void close()`

Closes the socket

`void connect(SocketAddress endpoint)`

Connects this socket to the server

```
void connect(SocketAddress endpoint, int timeout)
```

Connects this socket to the server with specified timeout value

```
InetAddress getInetAddress()
```

Returns the address to which this socket is connected

```
InputStream getInputStream()
```

Returns an input stream for this socket

```
int getLocalPort()
```

Returns the local port to which this socket is bound

Code Example

Let us create a chat application using TCP socket communication:

```
//Code sample for chat
```

TCP Server

```
package com.seed.net;
//import statements
class Chatserver
{
    ServerSocket server;
    Socket connection;
    DataInputStream dis;
    DataOutputStream dos;
    DataInputStream disi;
    public Chatserver()
    {
        try
        {
            server=new ServerSocket(2000, 1,
InetAddress.getLocalHost());
            //create instance of ServerSocket object
        }
    }
}
```

```
System.out.println("Server started.....");
connection=server.accept ();
/*accept method called on server sockets object
which will give you Socket
connection.*/
System.out.println ("Request received...");
}
catch(Exception e)
{
    System.out.println(e);
}
}

/*User define method talk() for read and write
operation using I/O stream.*/
public void talk() throws IOException,
UnknownHostException
{
try
{
dis=new
DataInputStream(connection.getInputStream());
dos=new
DataOutputStream(connection.getOutputStream());
disi=new DataInputStream(System.in);
while(true)
{
    System.out.println(dis.readUTF());
    dos.writeUTF(dis.readLine());
}
}
catch (Exception e)
{
    e.printStackTrace();
}
finally
{
    connection.close();
}
```

```
        }
    }
public static void main(String[] str)
{
try
{
Chatserver cs=new Chatserver();
cs.talk();
}
catch(Exception e)
{
e.printStackTrace();
}
}
```

TCP Client

```
package com.seed.net;
//import statements

class Chatclient
{
Socket connection;
DataInputStream dis;
DataOutputStream dos;
DataInputStream disi;
public Chatclient() throws Exception
{
connection =new Socket (InetAddress.getLocalHost (),
2000);
//create instance of socket object which accepts the
host and port as arguments.
System.out.println ("Request sent....");
}
/*User define method talk() for read and write
operation using I/O stream.*/
public void talk()
```

```
{  
    String str=new String(" ");  
    try  
    {  
        dis=new  
DataInputStream(connection.getInputStream());  
        dos=new  
DataOutputStream(connection.getOutputStream());  
        disi= new DataInputStream(System.in);  
        while(true)  
        {  
            str=new String(dis.readUTF());  
            dos.writeUTF(str);  
            System.out.println(dis.readUTF());  
        }  
    }  
    catch(Exception e)  
    {  
        e.printStackTrace();  
    }  
    finally  
    {  
        connection.close (); //closing connection  
    }  
}  
public static void main(String args[])  
{  
    try  
    {  
        Chatclient cc=new Chatclient();  
        cc.talk();  
    }  
    catch(Exception e)  
    {  
        e.printStackTrace();  
    }  
}
```

}



Tech App

All the remote program communication takes place based on the socket communication. All the remote applications like chat, email, websites (HTTP), database interaction is done with the help of socket programming.

Hands-on with TCP/IP

- Following are a few useful commands:
 - ipconfig
 - Gives the IP address of machine
 - ping
 - Indicates whether a machine is available on the network
 - ftp
 - Mainly used for file transfer
 - telnet
 - Mainly used to communicate with a remote login service

TCP/IP commands

Following table lists some of the commonly used commands for using TCP/IP. These are important for the developer for deployment of applications.

Command Name	Ipconfig
Description	This is a common command to identify a particular machine. A machine's IP address can be obtained by typing it at the command prompt.
Syntax	Ipconfig
Example	Ipconfig

Command Name	Ping
Description	<p>The ping utility verifies connection to a remote computer through TCP/IP stack.</p> <p>If no reply is received from the device being pinged, it indicates a network failure between the local and remote nodes or firewall blocking ping request.</p>
Syntax	Ping < IP address>
Example	Ping 120.3.224.2

Command Name	Netstat
Description	Netstat is a useful tool for checking network and Internet connections. Netstat displays the active TCP connections and ports. This can be used to find out if any of the servers are already running on some specific ports.
Syntax	netstat [-a] [-b] [-e] [-f] [-n] [-o] [-p proto] [-r] [-s] [-t] [-v] [interval]
Example	netstat -a

Command Name	ftp
Description	FTP is an acronym for File Transfer Protocol. It is the simplest way to exchange files between computers on the Internet. It is commonly used to download shareware or other files to a personal computer from other networks. To use FTP, an FTP client has to be acquired. A web browser can also make FTP requests to download programs found on a web page.
Syntax	ftp [-v] [-d] [-i] [-n] [-g] [-s:FileName] [-a] [-

	w: <i>WindowSize</i>] [-A] [<i>Host</i>] get filename – to download a file put filename – to upload a file
Example	ftp –put d:\filename.txt

Command Name	telnet
Description	telnet is a user command used for accessing remote computers. Using this command one can log on to the remote server.
Syntax	telnet [host [port]]
Example	telnet 172.16.3.95



Interview Tip

Application of TCP/IP commands is important. Practice these commands and master them. This is an important peripheral knowledge for a software developer or tester.



Tech App

Knowing TCP/IP commands and architecture is important from application deployment and testing perspective. Though this would not be required for a developer for creating programs, it is very essential for deployment and debugging in enterprise level applications which use application servers, databases and more complex interaction between systems.

Socket Programming using UDP

- No connection establishment between Client and Server before transfer of data.
- No fixed route for UDP packets is pre-decided.
- Sender attaches IP address and Port of destination.
- Server extracts IP address and Port of the sender from received datagram.
- Transmitted data may be received out of order or lost.
- UDP provides unreliable transfer of group of bytes("datagrams") between client and server.

DatagramSocket and DatagramPacket classes provide the developer a facility to create a connectionless communication using UDP protocol. Following table lists some of the important methods from DatagramSocket and DatagramPacket classes.

DatagramSocket

```
public DatagramSocket( ) throws SocketException  
public DatagramSocket(int port) throws  
SocketException
```

The above constructors create a datagram socket and bind it to the specified local port. If port is not specified or is set to zero, the new socket is bound to any locally available port except the operating system reserved ports. Any port number is allocated after verification by the SecurityManager.

```
void bind(SocketAddress addr)
```

Binds this DatagramSocket to a specific address & port.

```
void connect(InetAddress address, int port)
```

Connects the socket to a remote address for this socket.

```
void receive(DatagramPacket p)
```

Receives a datagram packet from this socket.

```
void send(DatagramPacket p)
```

Sends a datagram packet from this socket.

DatagramPacket

```
DatagramPacket(byte[] buf, int length)
```

Constructs a DatagramPacket for receiving packets of length length.

```
InetAddress getAddress()
```

Returns destination InetAddress, typically used for sending.

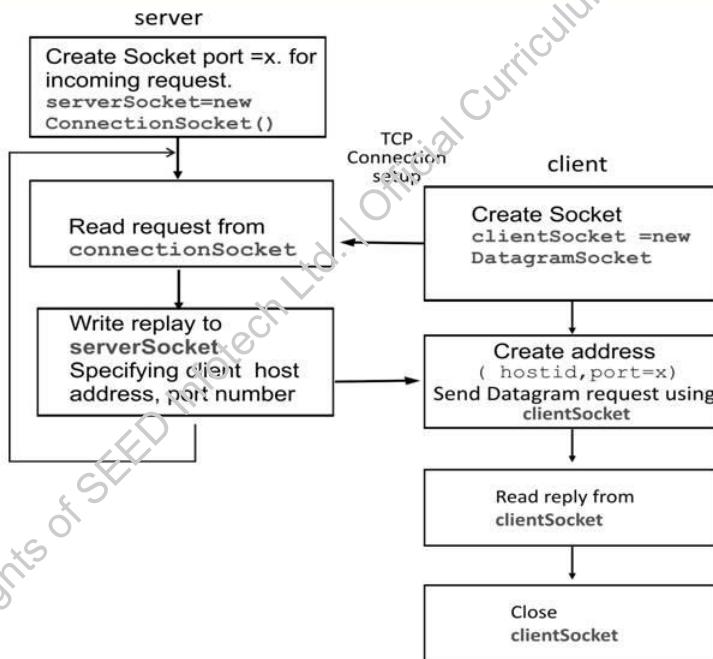
```
int getPort()
```

Returns the port number

```
byte[] getData()
```

Returns data contained in datagram(after receiving)

UDP Algorithm



Code Example

Following source code implements the algorithm to create simple communication using UDP based sockets:

UDP Client

```

package com.seed.net;
import java.io.*;
import java.net.*;
class UDPClient
{
    public static void main(String args[]) throws
Exception
    {
        BufferedReader inFromUser =new BufferedReader(new
InputStreamReader(System. in));
        DatagramSocket clientSocket = new
DatagramSocket(9876);
    }
}

```

```
// create instance of DatagramSocket which binds
specific port
InetAddress IPAddress = InetAddress.getByName
("hostname");
byte [] sendData = new byte [1024];
byte [] receiveData = new byte [1024];
String sentence = inFromUser.readLine ();
sendData = sentence.getBytes ();
DatagramPacket sendPacket = new
DatagramPacket(sendData,sendData.length,IPAddress,
9876);
/* create instance of DatagramPacket which datagram
packet for sending packets of
length to the specified port number on the
specified host.*/
clientSocket.send(sendPacket);
// Sends a datagram packet from this socket.
DatagramPacket receivePacket = new
DatagramPacket(receiveData,receiveData.length);
clientSocket.receive(receivePacket);

// Receives a datagram packet from this socket.
String modifiedSentence = new
String(receivePacket.getData());
System.out.println("FROM SERVER:" + modifiedSentence);
clientSocket.close();
}
```

UDP Server

```
package com.seed.net;
import java.net.*;
import java.io.*;
public class ServerUDP
{
```

```
public static void main(String args[]) throws
Exception
{
    try
    {
        DatagramSocket serverSocket = new
DatagramSocket(9876);
        // create instance of DatagramSocket which binds
specific port
        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];
        while(true)
        {
            receiveData = new byte[1024];
            DatagramPacket receivePacket = new
DatagramPacket(receiveData,
            receiveData.length);

            //Create instance of DatagramPacket
            System.out.println ("Waiting for Client request");
            serverSocket.receive(receivePacket);
            InetAddress IPAddress = receivePacket.getAddress();
            int port = receivePacket.getPort();
            System.out.println ("From: " + IPAddress + ":" +
port);
            String data = new
String(receivePacket.getData(),0,receivePacket.getLength());
            System.out.println(data);
            DatagramPacket sendPacket = new

DatagramPacket(sendData,sendData.length,IPAddress,port)
;
            serverSocket.send(sendPacket);
            // Sends a datagram packet from this socket.
        }
    }
}
```

```
        catch(SocketException ex)
        {
            System.out.println("UDP Port 9876 is occupied.");
            System.exit(0);
        }
    }
```

TCP Vs UDP Sockets

	TCP	UDP
Connection	Connection-oriented	Connection less
Reliability	More reliable	Less reliable
Speed	Slow	Fast
Applications	FTP, HTTP, telnet	RIP, SNMP

TCP and UDP protocols constitute the transport layer for TCP/IP. Transmission control protocol (TCP) is connection-oriented protocol. That is this protocol first establishes a connection with the target computer and then sends the data. Each data packet sent is confirmed by the receiver through acknowledgement. This makes the protocol more reliable. When speed is more critical, these overheads of establishing connections etc. make use of TCP unviable. In such cases UDP (User Datagram Protocol) is used which is connectionless protocol and has less overheads. Though UDP is less reliable as compared to TCP, it is faster.

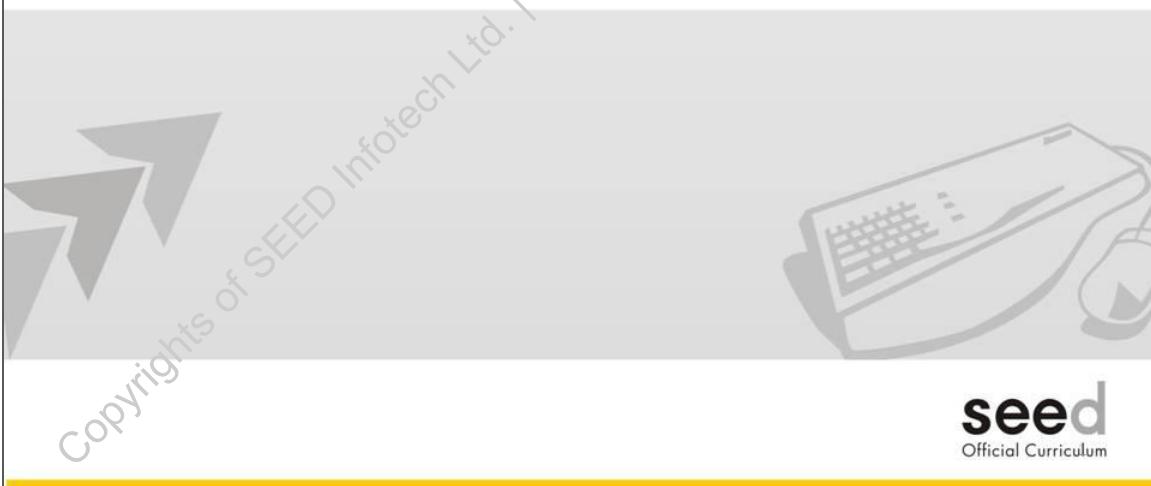
Java API provides Socket classes implementing both of these protocols. Designer has to choose the appropriate socket implementation based on the need of the situation.



It is not that UDP is seldom used due to its unreliability. Since it does not have overhead of connection making, it is very fast. Many advanced messaging infrastructures like TIBCO use UDP protocol for its speed and less overhead.

Chapter - 9

Java Database Connectivity(JDBC)



This chapter covers database connectivity using JDBC API. It includes SQL, Parameterized SQL, Stored Procedure, JDBC Drivers, Database Transactions, Result set metadata and Database metadata.

Objectives

At the end of this chapter you will be able to:

- Compare File versus Database as storage medium.
- List features provided by databases.
- List Types of SQL.
- Identify the need for JDBC.
- List various technologies used to access data from databases.
- State the algorithm for reading data from a database.
- Construct a Java program to execute an SQL query.
- List different types of JDBC drivers with their usage.
- Identify various parts of JDBC URL and construct it as per need.

- Use SQL to update the database records.
- Use JDBC construct to ensure ACID properties of transactions.
- Construct and use parameterized SQL.
- Invoke a given stored procedure.
- Use `ResultSetMetaData` class to fetch additional information about the result set like column names.
- List important methods of `DatabaseMetaData` class and their usage.

Copyrights of SEED Infotech Ltd. | Official Curriculum

Storage: Files Vs Database

- A database is a system which helps to organize, store, and retrieve large amounts of data easily.



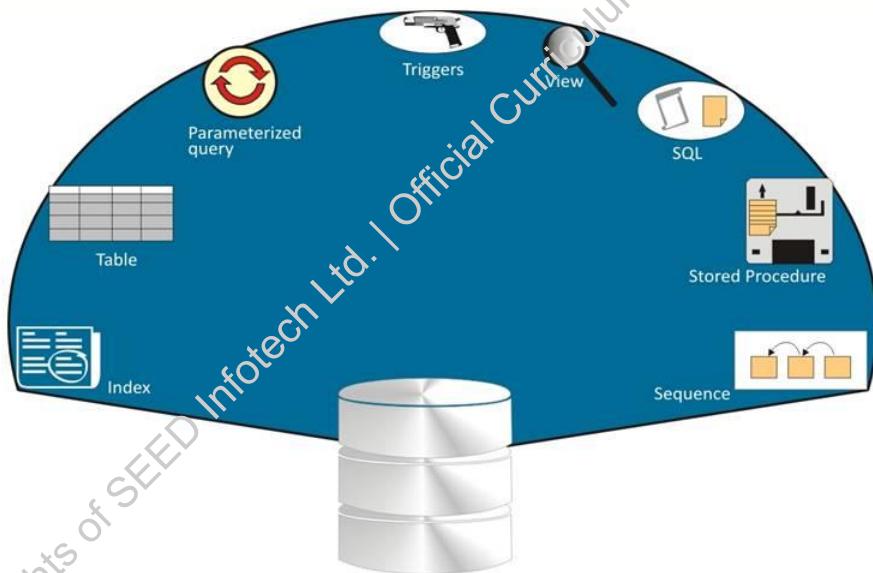
	File	Database
Data Redundancy	High	Low
Data Integrity	No	Yes
Data Ownership	File wise	Data item
Security Level	File	Data item
Multi-access	Difficult	Easier

Data is the most important part for any business application. Essentially, programs are created to manage this business data. Consider an example of data about all the employees in an organization. It is maintained in the files in different departments like HR, Accounts and the department to which the employee belongs to. In this case, same data is duplicated at multiple places. Extra computer storage is needed to store these duplicate copies of data. Additionally, when the data is stored at different locations, it is difficult to make the changes at all the locations at the same time. This leads to a problem called data integrity. Data integrity refers to the correctness of data maintained. For example, an employee leaves the organization and the accounts department is unaware about it, the salary processing will take place for that employee. Since each department being an owner may modify their own copy making the data inconsistent. Hence control over a group of records or data cannot be made uniform at the organizational level. File system supports security at file level. Access can be given to an entire file or denied to the entire file. This acts as a severe limitation when one wants to control data at record level.

A database is a system which helps to organize, store, and retrieve large amounts of data. All these limitations are overcome by using a database in which the data is stored at a central location. All applications can have uniform access to this data. Data duplication or redundancy is greatly reduced. Data integrity is maintained as the changes are done at one place in the database. All the applications can be given selective access to the data items depending on the requirement. As the data can be accessed at data item level rather than file level, security can be controlled at data item level. Modern business applications use databases to store the data instead of files. Relational database management systems or RDBMS are widely used for this purpose. Examples of leading RDBMS softwares are Oracle, SQL Server, IBM DB2, Sybase, MySQL etc.

Copyrights of SEED Informational Curriculum

Relational Database Management System



- Relational Databases or RDBMS stores data in the form of rows and columns.

3

A database is a system which helps to organize, store and retrieve large amounts of data easily. It is a set of related records and a set of programs to access this data. It is an entire system that helps to enter, store and manage data, so it is called Database Management System (DBMS). Filing cabinet is used to file the papers. New papers can be added to it and old unwanted papers can be removed. Changes can be made to the existing papers. If any particular paper is required, it can be retrieved. For easy and fast retrieval, index can be maintained. Sequence can also be given to these papers for easy maintenance and retrieval. DBMS is exactly like this filing cabinet which is electronic.

Relational Databases or RDBMS is a type of database management system which stores data in the form of rows and columns. There are many database objects like stored procedures, index, triggers, sequence, view, etc. as shown in the above diagram.

- **Structured Query Language (SQL):** SQL is used to perform these operations such as addition, deletion, updating and retrieval of data from the database.

- **Parameterized Query:** An SQL can be treated like a function in a programming language and passed parameters to it. This special type of SQL queries are called parameterized query. They execute faster than SQL.
- **Stored Procedure:** Stored procedure by definition is a segment of code which contains declarative or procedural SQL statements.
- **Index :** Index is an additional data created in the database which allows faster access to the rows through use of pointers. Indexing of a table typically speeds up the data access from that table. It is a sorted list of rows accompanied by location of the row.
- **Trigger:** Trigger is a PLSQL program unit associated with a specific table which gets fired automatically whenever any user or application tries to modify the data from the concerned table in a predefined way. Trigger can be used to apply business rules or to log all the changes in a particular data table.
- **Sequence:** Sequence helps to give a unique identity to particular entity.
- **View:** View is a logical or virtual table, based on a table or another view. View does not have its own data but shows data from the base tables. Views can be used to present relevant data from the same table to different users in different format (view).



Focus on Knowledge about various database objects and their usage.

Interview Tip

Structured Query Language (SQL)

- Language to query the database
- Types of SQL
 - Data Definition Language (DDL)
 - Creating tables and other database objects, etc.
 - Data Manipulation Language (DML)
 - Inserting, Updating, Deleting the records
 - Data Control Language (DCL)
 - Granting privileges, revoking them

4

SQL is used to perform these operations such as addition, deletion, updating and retrieval of data from the database.

SQL has three sub parts.

- **Data Definition Language (DDL)** is a set of SQL commands used to create or modify and delete database structures or objects such as tables, views etc. and not actual data.
Examples of DDL commands are CREATE, ALTER, DROP.
- **Data Manipulation Language (DML)** is a set of SQL commands which are used to query, insert, update, and delete data stored in the database.
Examples of DML commands included are SELECT, INSERT, UPDATE, DELETE.
- **Data Control Language (DCL)** is a set of commands for controlling access to the data.
Examples of DCL commands included are GRANT, REVOKE.



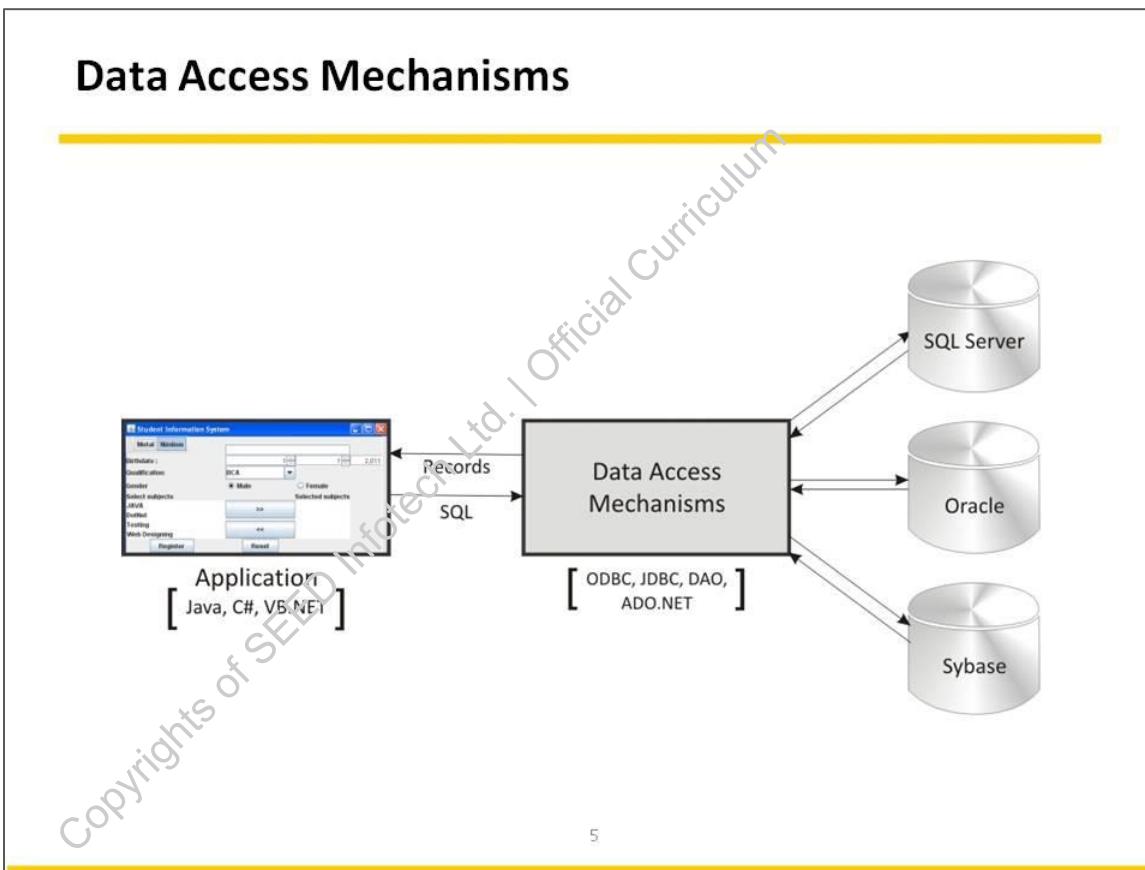
Group Exercise

Employee Table Structure

Name	Type
EMPLOYEE NO	NUMBER(4)
ENAME	VARCHAR2 (50)
SAL	NUMBER (5)

Write SQL Queries for following.

1. Select all employee names from employee table.
2. Delete employee with employee no = 30.
3. Add an employee with employee no 31, name joy and salary 20000.



SQL is used to access the database. Application can be written in any language, for example, Java, C#, VB.NET, C++, VB 6.0, etc. Consider an example of online reservation system, banking application, etc. These applications need to access the data or sometimes change the data in the database. Data access mechanisms are required to do this task. They can be ODBC (Open Database Connectivity), JDBC (Java Database Connectivity), DAO (Data Access Object), ActiveX Data Object (ADO) and ActiveX Data Object on .NET platform (ADO.NET).

These mechanisms provide support in the form of different objects that interact with lower level APIs (Application Programming Interfaces) to interact with the respective database. These lower level APIs are called drivers. They provide an interface to access the underlying database.

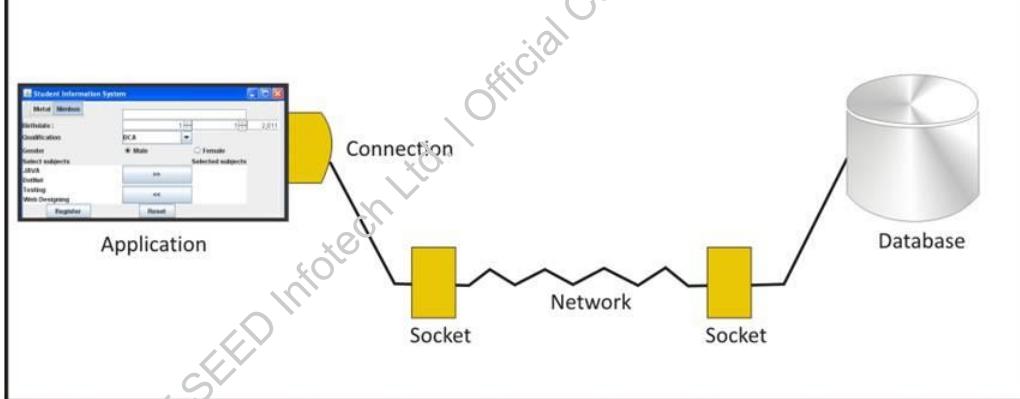
Accessing the Data: Steps

- 1 Create a SQL query.
- 2 Establish a connection.
- 3 Associate SQL with the connection.
- 4 Fire the query.
- 5 Get the result (set of rows)
- 6 Process the result.
- 7 Close the connection.

6

To access the data from database above are the typical steps. Let us understand each of these steps in more details. We already know how to create an SQL query.

Connection and JDBC Drivers



7

A Connection object represents a connection with a database. All operations executed on database are invoked through API of Connection object. It represents a session or a communication channel. Multiple connections can be opened with the same/different databases simultaneously. Connection object and its API completely hides complexity of socket communication between the Application program and the database. Connection class APIs are discussed in details in subsequent slides.

Types of JDBC Drivers

Driver is a program which allows the application to establish connection with the database by hiding complexity of handling network connection/sockets. Essentially Connection object uses the JDBC driver which is a Java class to make the connection and communicate with the database.

The JDBC drivers are of 4 types based on how they are created.

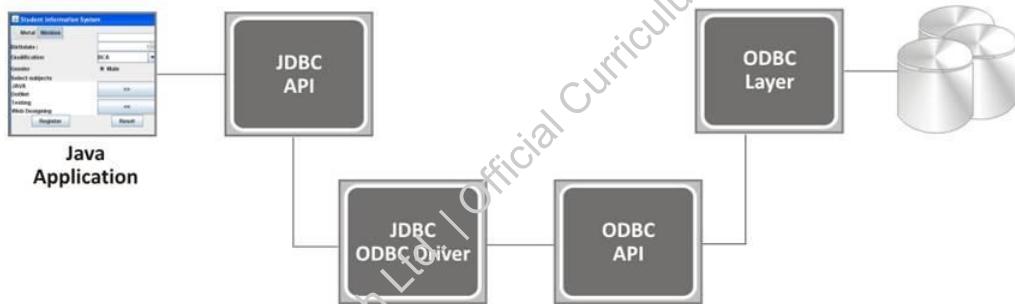
JDBC-ODBC Bridge	(Type 1)
Native-API partly Java Driver	(Type 2)
Net-Protocol All-Java Driver	(Type 3)
Native Protocol All-Java Driver	(Type 4)



To get complete list of JDBC drivers use the following link

<http://industry.java.sun.com/products/jdbc/drivers>

JDBC-ODBC Bridge (Type 1)



- Suitable only for prototyping purposes.
- Limited to functionality of ODBC driver.
- Not suitable for higher volume of transactions.
- Inherits all the limitations of ODBC implementation.
- Slower in performance.

8

Type 1 driver leverages ODBC driver implementation. It translates all JDBC calls to ODBC (open database connectivity) calls and sends them to the ODBC driver. It is a Java wrapper developed over ODBC API. It should be used only for prototyping and is not recommended for production environment.

Advantage

- Easily available as part of JDK.

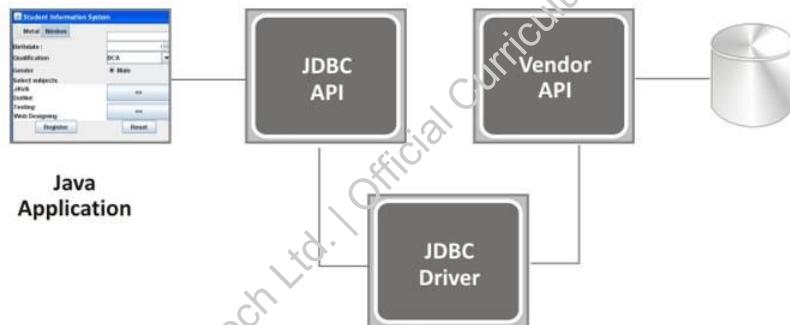
Disadvantages

- Limited to functionality of ODBC driver.
- Not suitable for higher volume of transactions.
- Slower in performance.

Example

- `sun.jdbc.odbc.JdbcOdbcDriver`

Native-API partly Java Driver (Type 2,



- Written partly in Java and partly in native code.
- Some platform-specific code in addition to Java library
- Uses native 'C' language lib calls for Conversion.

9

Type 2 drivers use a mixture of Java implementation and vendor-specific native APIs to provide data access. Essentially a Java Native Interface (JNI) implementation is developed around the native driver code. This mandates a requirement of installation of native code API on every client that runs Java application. Native part of the code performs faster and efficient communication with the underlying database system.

Advantage

- Type 2 drivers typically offer better performance than the JDBC-ODBC Bridge as the layers of communication (tiers) are less than that of Type 1 and also it uses Native API which is Database specific.

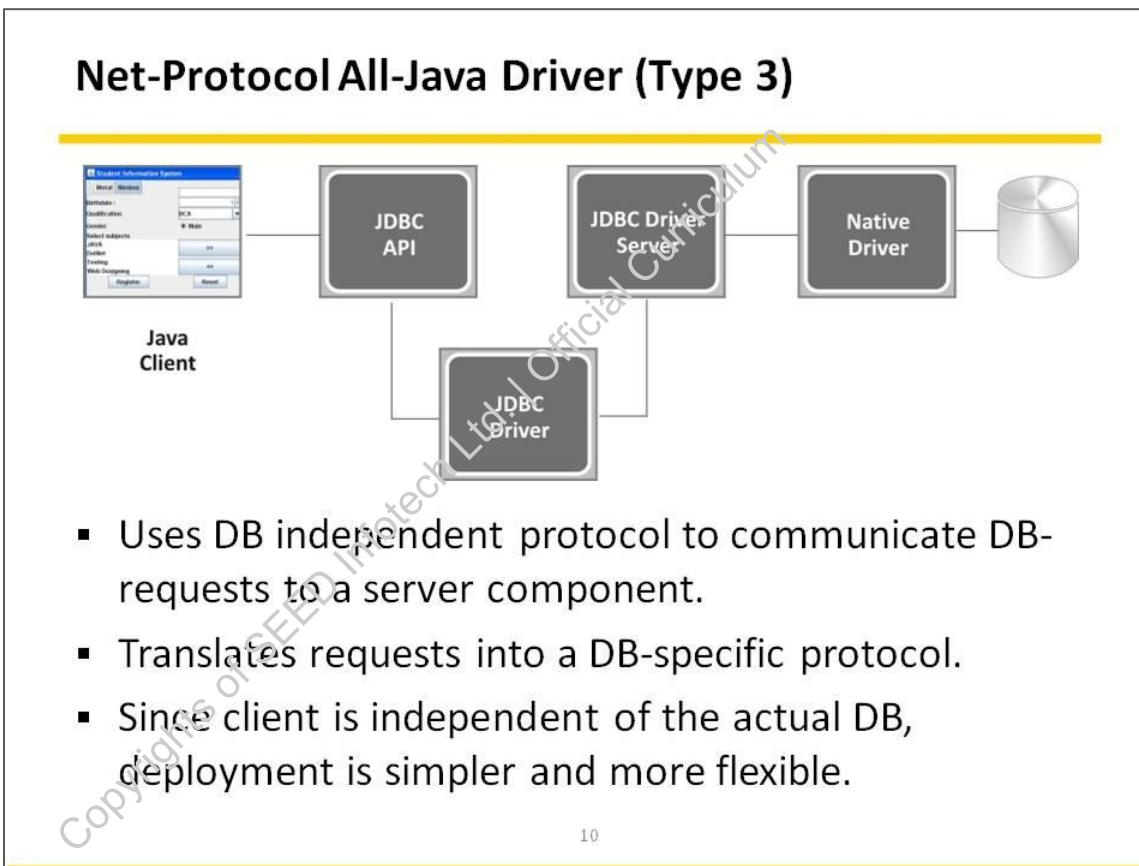
Disadvantages

- Native API must be installed in the Client System and hence type 2 drivers cannot be used for the Internet.
- Like Type 1 drivers, it is not written in Java Language which forms a portability issue.

- If the database is changed the native API also has to be changed.
- Mostly obsolete now.
- Usually not thread safe.

Example

- `com.ibm.db2.hcc.DB2Driver`



Advantages

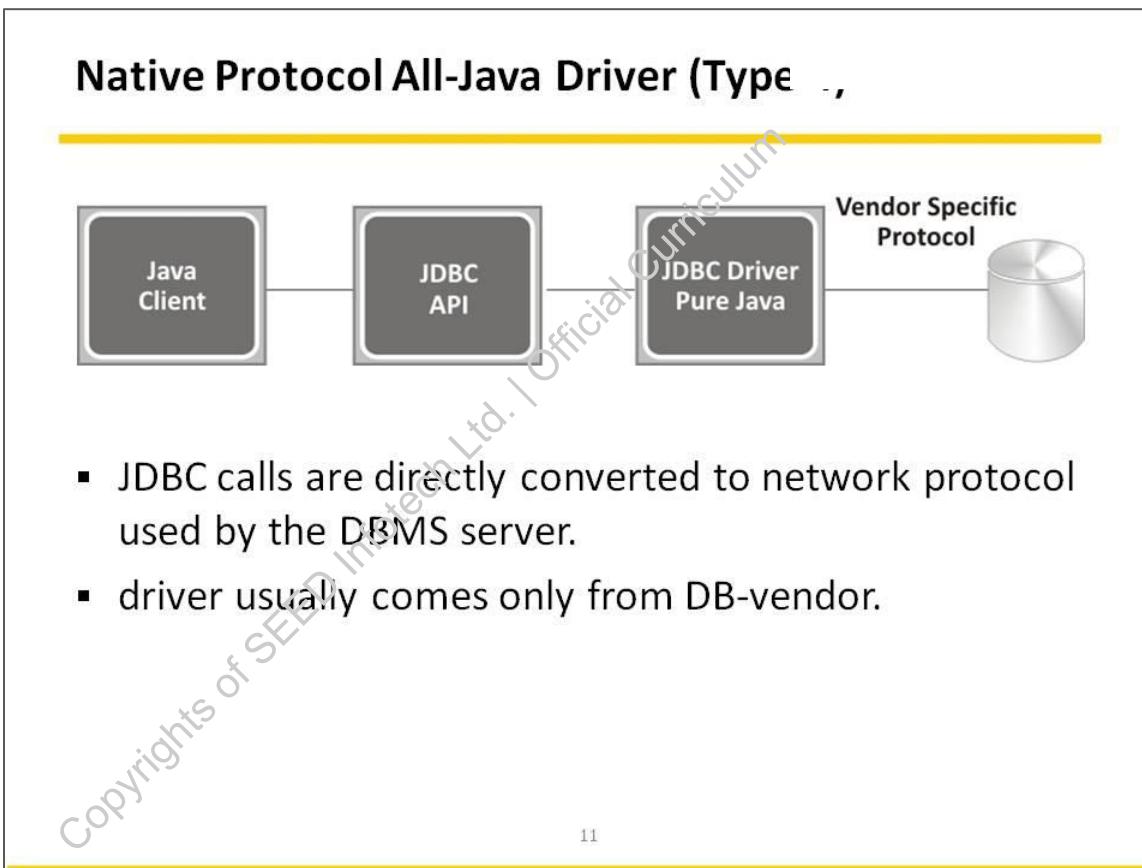
- This driver is server-based. So, the vendor database library is not required for client machines. It is fully written in Java and hence portable. It is suitable for the web.
- There are many opportunities to optimize portability, performance, and scalability.
- The net protocol can be designed to make the client JDBC driver very small and fast to load.
- It typically provides support for features such as caching (connections, query results, and so on), load balancing, and advanced system administration such as logging and auditing.
- It is very flexible and allows access to multiple databases using one driver.
- It is the most efficient amongst all driver types.
- It essentially converts JDBC calls into vendor specific protocols thereby eliminating need of native code installation at each client.

Disadvantages

- It requires another server application to be installed and maintained. Traversing the record set may take longer, since the data comes through the backend server.
- In this case clients connect to database servers via an intermediate server component that acts as a gateway for multiple database servers.

Example

- `com.informix.jdbc.lfxDriver`



Type 4 drivers typically make direct socket connections to the databases. They convert the JDBC API calls to network calls using vendor-specific networking protocols. Type 4 drivers generally offer better performance than their Type 1 and Type 2 counterparts. These are preferred over other drivers, since there are no additional libraries or middleware to install. Almost all major database vendors provide Type 4 JDBC drivers.

Advantages

- Type 4 drivers are completely written in Java. So we can achieve platform independency. Number of translation layers is very less.
- Performance is good.
- Need not to install special software on the client or server. Further, these drivers can be downloaded dynamically.

Disadvantages

- With type 4 drivers, the user needs a different driver for each database

Example

- `oracle.jdbc.driver.OracleDriver`

JDBC URL: Locate the database

- Needed by drivers to locate, access and get other valid information about the databases
- Typical Syntax

`jdbc : [subprotocol] : [subname] [attributes]`

- JDBC URL examples

`jdbc : odbc : dataSourceName`

`jdbc : db2 : database_name`

12

A Java application may use multiple databases to store data. These databases can be accessed using different database drivers. The parameters required to obtain connection may be different. In this situation, how to identify a specific driver is a question. A JDBC URL is the answer. A database connection URL is a string that DBMS JDBC driver uses to connect to a database. It can contain information such as where to search for the database, the name of the database to connect to, and configuration properties. The exact syntax of a database connection URL is specified by your DBMS and JDBC driver documentation.

The standard syntax for JDBC URLs is shown below.

`jdbc : [subprotocol] : [subname] [attributes]`

JDBC URL examples

ODBC Driver	<code>jdbc: odbc: dataSourceName</code> <code>jdbc:odbc:test</code>
DB2 Driver	<code>jdbc:db2:database_name</code> <code>jdbc:db2:MyDB</code>
Informix Driver	<code>jdbc:Informix-sqli://<host>:<port>/<database_name>:INFORMIXSERVER=<sid></code> <code>jdbc:Informix-sqli://localhost:1025/MyInformixDB:INFORMIXSERVER=MyDBServer</code>
Oracle thin Driver	<code>jdbc:oracle:thin:@host:port:service</code> <code>jdbc:oracle:thin:@localhost:1521:oracle10g</code>

Getting Connection

```
try {  
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  
    Properties p = new Properties();  
    p.put("user","dba");  
    p.put("password","sql");  
    String url = "jdbc:odbc:mytable";  
    Connection c =  
    DriverManager.getConnection(url,p);  
}
```

13

Class.forName takes a string class name and loads the necessary class dynamically at run time. When a driver is loaded, it registers itself with DriverManager using registerDriver() method. The getConnection() method of DriverManager class attempts to establish a connection to the given database URL. The DriverManager attempts to select an appropriate driver from the set of registered JDBC drivers.

Using Driver class

Code Example

```
String driverName="oracle.jdbc.driver.OracleDriver";  
Driver driver= (Driver) Class.forName  
(driverName).newInstance ();  
Properties p=new Properties();  
p.put ("user","java");  
p.put ("password","java");  
String  
url="jdbc:oracle:thin:@192.168.1.100:1521:oracle9i";
```

```
con=driver.connect (url, p);
```

Getting connection using System.properties

Code Example

```
Properties props = new Properties ();
FileInputStream in = new FileInputStream
("database.properties");
props.load (in);
in.close();
String drivers = props.getProperty ("jdbc.drivers");
if (drivers!= null) System.setProperty("jdbc.drivers",
drivers);
String url= props.getProperty("jdbc.url");
String username = props.getProperty ("jdbc.username");
String password = props.getProperty ("jdbc.password");
```

Executing a Simple SQL Query

```
String url = "jdbc:odbc:MyDataSource"
Connection con = DriverManager.getConnection(
    url);
Statement stmt = con.createStatement();
String sql = "SELECT Last_Name FROM EMPLOYEES";
ResultSet rs = stmt.executeQuery(sql);
while(rs.next())
{
    System.out.println(rs.getString("Last_Name"))
}
}
```

14

Statement

The Statement object returns a `java.sql.ResultSet` object that encapsulates the results of query execution. This is an interface implemented by driver vendors. The result set can be scrolled using a cursor for reading the data values in the `ResultSet`.

Using Statement

1. Load the JDBC driver.
2. Get Connection.
3. Call `CreateStatement`.
4. Call `executeQuery` with SQL query as parameter.
5. Iterate over `ResultSet` to retrieve data.
6. Close the Statement object.
7. When you are finished using a Statement, call the method `Statement.close` to immediately release the resources it is using. When you call this method, its `ResultSet` objects are closed.

Closing the connection

Before JDK 7

```
try { // your database operation statements go here  
}  
finally { if (con != null) con.close(); }
```

With JDK 7

```
try (Statement stmt = con.createStatement()) {  
} catch (SQLException e) {  
}
```

15

If you have carefully noticed, how Connection object is handled. It is very important to close the connection once your work is done. If connection object is kept open and its reference is lost, there will be a resource leak i.e. connection object continues to occupy memory but is not useable. If this continues for a longer period, database would run out of connections and application would crash. It is utmost important to manage resources like connection object.



Best Practice

Prior to JDK 7 and JDBC 4.1

When you work with the connection is finished, you need to make sure that Connection is closed properly. Following code snippet shows how to do it in a correct way. Call to `close()` method will release the connection with the database or will return the connection object to the pool is dependent on the implementation.

```
try { // your database operation statements go here
```



JDK7

```
}
```

```
finally { if (con != null) con.close(); }
```

In JDK 7 and JDBC 4.1

In JDBC 4.1, which is available in Java SE release 7 and later, you can use a try-with-resources statement to automatically close Connection, Statement, and ResultSet objects, regardless of whether an SQLException has been thrown. An automatic resource statement consists of a try statement and one or more declared resources.

```
try (Statement stmt =
con.createStatement()) {
} catch (SQLException e) {
}
```

Executing DML Statement

```
string url = "jdbc:odbc:MyDataSource"
Connection con = DriverManager.getConnection(
    url);
Statement stmt = con.createStatement();
String sql="insert into employee(empno,
    empname,designation,salary)

    values(5,'rrr','manager',11000)";
int updateRowCount=stmt.executeUpdate(sql);
```

16

To execute a DML type of SQL a different method is provided called `executeUpdate()`. It executes the given SQL statement which may be INSERT, UPDATE or DELETE and returns the count of affected rows in the database.

Maintaining Data integrity: Commit

```

Begin Transaction
  unit operation 1
  unit operation 2
  unit operation n
if Successful then
  Commit
Else
  Rollback

```



```

Begin Transaction
  Cancel ticket Train1
  Book ticket Train2
if (Successful) then
  Make changes permanent
Else
  do not cancel the
  ticket Train1

```

17

Transactions

The day to day (real-life) applications like electronic money transfer, reservation systems or accounting system involve a series of operations to complete a task. This series of operations needs to be treated as an atomic unit - 'a logical transaction' and we also need to have the facility for undoing changes under certain circumstances.

For example, in a railway reservation system request to book a ticket for a particular train against the cancellation of ticket for some other train. It is also specified that the existing ticket is to be cancelled only if we are able to book it for the other train.

1. This task involves two operations.
2. Cancellation of a ticket for one train and
3. Booking of a ticket for some other train.

These operations can be accomplished by executing a set of commands on the reservation database. If one operation in a set of operations fails it is not

acceptable here. Here the set of operations should either succeed or fail as a group and not as individual operations.

The transaction processing support provides a solution for such situations. A series of changes made using SQL commands like INSERT, UPDATE or DELETE can be grouped together using transactions.

In a transaction, when manipulating the data in the database, the changes made by us are not normally made permanent in the database until we give a command to do so.

In a transaction, the changes made are not normally made permanent in the database until we confirmed them. If changes are not confirmed, the database maintains its original state i.e. changes made during the database operations are undone.



Read more about ACID properties of transactions.

Additional Reading

Transactions in JDBC

The code block for transactions in JDBC is shown in the example below. By default the property called AutoCommit is true, hence every single DML statement is treated as a transaction and committed to the database. Hence when setAutoCommit(false) is called, it disables it and allows the transaction scope to contain more than one DML statements. Here this statement is equivalent to Begin Transaction. All the statements inside the try block are part of the transaction. If everything goes well, last statement to execute is Commit which indicates success of the transaction.

In case of any error, an exception would be thrown and catch block will be executed rolling back the entire transaction.

```
public void moneyTransfer (Account from, Account to,  
int amount) {  
    try {  
        con.setAutoCommit (false);  
        from.balance=from.balance-amount; // withdraw from  
        one Account
```

```

pst=con.prepareStatement ("update account set
balance=? Where
accno=?");
pst.setInt(1,from.balance);
pst.setInt (2, from.accountId);
int i=pst.executeUpdate ();
to.balance=to.balance+amount; // deposit in other
account
pst=con.prepareStatement ("update account set
balance=? where
accno=?");
pst.setInt (1,to.balance);
pst.setInt (2,to.accountId);
int ii=pst.executeUpdate ();
con.commit (); // if both are successful keep the
new state

} catch (Exception b) {
try {
con.rollback (); // if one of them fails undo
the other one
} catch (SQLException e) {
e.printStackTrace ();
}
}
}

```



Transaction handling is an important part of database interactions while creating business applications. Understand the transaction mechanism in details.

JDBC Connection API

- `close()`
- `commit()`
- `void setAutoCommit(boolean b)`
- `rollback()`
- `Statement createStatement()`
- `CallableStatement prepareCall(String sql)`
- `PreparedStatement`
`prepareStatement(String sql)`

18

Following are important APIs of Connection interface:

Method	Description
<code>public void close() throws SQLException</code>	Releases a Connection's database and JDBC resources immediately instead of waiting for them to be automatically released.
<code>public void commit() throws SQLException</code>	Makes all changes made since the previous commit/rollback permanent and releases any database locks currently held by the Connection. This method should be used only when auto-commit mode has been disabled.
<code>public void</code>	Sets this connection's auto-

<pre>setAutoCommit(boolean autoCommit) throws SQLException</pre>	<p>commit mode. If a connection is in auto-commit mode, then all its SQL statements will be executed and committed as individual transactions. Otherwise, its SQL statements are grouped into transactions that are terminated by a call to either the method commit or the method rollback</p>
<pre>public void rollback() throws SQLException</pre>	<p>Drops all changes made since the previous commit/rollback and releases any database locks currently held by this Connection. This method should be used only when auto-commit has been disabled.</p>
<pre>public Statement createStatement() throws SQLException</pre>	<p>Creates a Statement object for sending SQL statements to the database. SQL statements without parameters are normally executed using Statement objects. If the same SQL statement is executed many times, it is more efficient to use a PreparedStatement object.</p>
<pre>public CallableStatement prepareCall(String sql) throws SQLException</pre>	<p>Creates a CallableStatement object for calling database stored procedures. The CallableStatement object provides methods for setting up its IN and OUT parameters, and methods for executing the call to</p>

	a stored procedure
public PreparedStatement prepareStatement(String sql) throws SQLException	Creates a PreparedStatement object for sending parameterized SQL statements to the database. A SQL statement with or without IN parameters can be pre-compiled and stored in a PreparedStatement object. This object can then be used to efficiently execute this statement multiple times.

Copyrights of SEED Infotech Ltd. | Official Curriculum

Parameterized SQL

```
string SQL =  
    "select * from Employees where First_Name=?";  
  
PreparedStatement pstat =  
    con.prepareStatement(sql);  
  
pstat.setString(1, "John");  
  
ResultSet rs = pstat.executeQuery();  
  
pstat.clearParameters();
```

19

Parameterized SQL is an SQL in which you can pass the parameters at runtime while you are executing it. Every database has a different way of implementing parameterized query. Since they are cached by the database, they improve the performance as compared to a non-parameterized query.

When database engine receives an SQL query it parses the query to find out any syntax errors, then execution plan is prepared to access the data in most efficient way. These steps are executed each time an SQL query is received. Databases usually have a statement cache which stores the execution plan. This arrangement facilitates reuse of execution plans for the statements which have been executed previously, merely by substituting different parameters for each successive execution.

PreparedStatement

PreparedStatement is Java way of performing parameterized query. The PreparedStatement object contains an SQL statement that has already been compiled. This means execution can be faster than that of Statement objects. It

may have one or more IN parameters whose value is not specified when the SQL statement is created. The statement has a question mark ("?") as a placeholder for each IN parameter. A value for each question mark must be supplied by the appropriate setXXX() method before the statement is executed, where XXX stands for data type. Being a subclass of Statement, PreparedStatement inherits all the functionality of Statement.



Use PreparedStatement for better performance instead of Statement object in Java applications.

Using PreparedStatement

1. Call `prepareStatement()` to create the object.
2. Set values for all the input parameters using `setXXX()` method. The first argument to the `setXXX()` methods is the ordinal position (index number) of the parameter to be set, and the second argument is the value to which the parameter is to be set. Call appropriate methods from `executeQuery`, `executeUpdate`.
3. Use `clearParameters` before using the same `PreparedStatement` object for successive call with different parameters.
4. In addition, it adds a whole set of methods which are needed for setting the values to be sent to the database in place of the placeholders for IN parameters. Also, the three methods `execute`, `executeQuery`, and `executeUpdate` are modified so that they take no argument. The Statement forms of these methods (the forms that take an SQL statement parameter) should never be used with a `PreparedStatement` object.
5. Before a `PreparedStatement` object is executed, the value of each '?' parameter must be set. This is done by calling a `setXXX` method, where XXX is the appropriate type for the parameter. For example, if the parameter has a Java type of long, the method to use is `setLong()`.

Stored Procedure

```
CREATE DEFINER='root'@'localhost' PROCEDURE
`sp_product` (IN pid INT,OUT price INT)
BEGIN
    SELECT price from product where product_id=pid;
    SET price=price*2;
    Update product set price=price where
product_id=pid;

END
```

20

By definition stored procedure is a segment of code which contains declarative or procedural SQL statements. It overcomes limitations of SQL. SQL has following limitations:

- SQL is a non-procedural language.
- SQL does not allow usage of variables and constants.
- Developer cannot control the flow of execution using SQL.
- SQL does not have ability to process data row by row in a loop.

Most of the RDBMS systems have their own programming language to create such procedures. Oracle's PL/SQL and SQL Server's Transact-SQL are examples of the same. These objects being tightly coupled with the underlying database are not easily portable across databases. Nonetheless they are faster as compared to SQL or parameterized SQL.

Above examples shows a simple stored procedure which accepts product ID and doubles its price. This stored procedure is already created. Let us see how to invoke it from A Java application. Note: How to create a stored procedure for a particular database is out of scope of this course.

Invoking Stored Procedure

```
CallableStatement cstmt =  
con.prepareCall( "{ call sp_product(?,?) } " );  
cstmt.setInt(1,12);  
cstmt.registerOutParameter(2,Types.FLOAT);  
  
cstmt.execute();  
  
System.out.println(cstmt.getFloat(2));
```

21

Use `prepareCall()` method to create `CallableStatement` object. JDBC provides callable statement interface as a mechanism to call stored procedures. `prepareCall()` method of `Connection` object is used to create a `CallableStatement`. JDBC defines standard escape syntax to access stored procedures.

```
{call procedure_name[ (?, ?, ...) ] }
```

In the above syntax, each question mark (?) represents a placeholder for a procedure or a return value. Note that the parameters are optional. It is responsibility of JDBC driver to convert this escape syntax into the database's own stored procedure syntax.

1. Set input Parameters

Substitute input parameters by calling `setXXX()` methods. These are numbered from 1 to n, left to right.

2. Register OUT and IN/OUT parameters

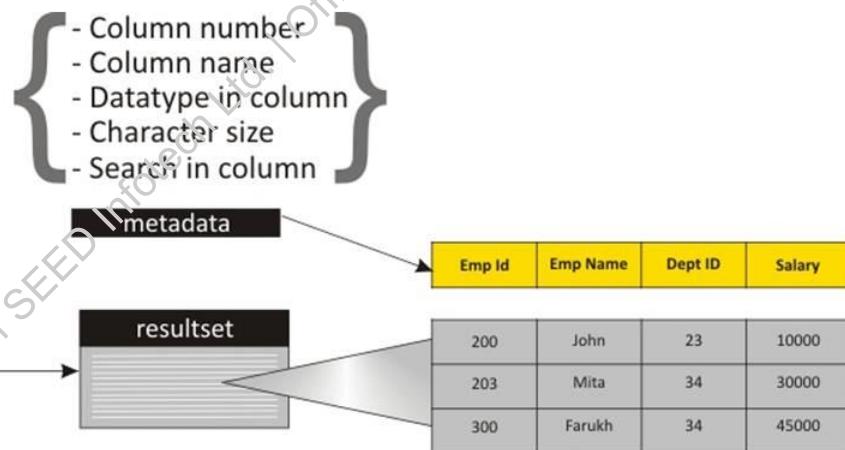
Register all the OUT and IN/OUT parameters using registerOutParameter() method. This method takes type of output as second argument.

3. Call execute() to Invoke the stored procedure.

Extract the OUT parameter values using the getXXX() methods of CallableStatement.

ResultSet MetaData

- ResultSet MetaData describes the structure of a `resultset` object



22

To retrieve the records or fields from a `resultSet` object, one needs to know the structure, sequence, column names, data type of columns etc. In most of the situations this information is known to the developer. Sometimes, it is necessary to get this information dynamically at runtime. `ResultSetMetaData` is used for this purpose. `ResultSetMetaData` interface provides information about the structure of a particular `ResultSet`. It can be obtained by calling `getMetadata()` method on any result set object. `ResultSetMetaData` makes it possible for the user to display or manipulate the data in the `resultset` without knowing its structure in advance. For example, a generic method can be developed which would accept any result set object as a parameter and display the contents of `ResultSet` in a Grid/Table fashion. The beauty of this mechanism is that there is no hard-coding of column names or their types.

Following table shows key methods of `ResultSetMetaData` interface:

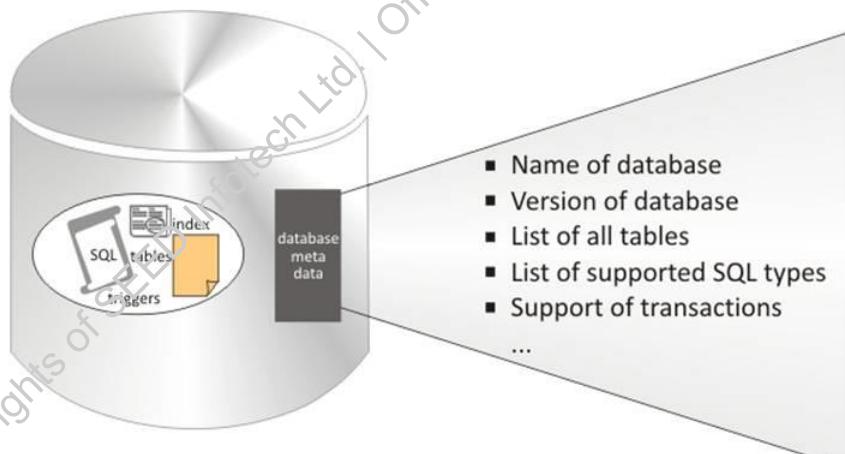
Method	Description
<code>int getColumnCount ()</code>	Returns the number of columns in this <code>ResultSet</code> object.
<code>String getColumnLabel (int index)</code>	Gets the designated column's suggested title for use in printouts and displays.
<code>String getColumnTypeName (int index)</code>	Note that type names returned by <code>getColumnTypeName ()</code> are database-specific (e.g., Oracle refers to a string value as a <code>VARCHAR</code> ; Microsoft Access calls it <code>TEXT</code>).



Database metadata is required for creating application like dynamic query builder.

Database MetaData

- Database Metadata provides information about the database.



23

Business applications are developed with prior knowledge of the database and the table structure. Sometimes, it is necessary to know information about the table structures, features supported by database and so on dynamically at runtime. This is like SYS Table space provided by Oracle. This table space stores information about the data hence called meta-data. JDBC provides DatabaseMetaData interface to dynamically discover information about database configuration, system table structure, its capabilities, and so on. Using this interface one can create an application to read and manipulate data or other objects like indexes, and so on. Independent of a particular database. This interface contains large number of methods. Some of the important ones are listed below:

Method	Description
<code>String getDatabaseProductName()</code>	Retrieves the name of this database product.
<code>ResultSet getTables(...)</code>	Retrieves a description of the tables

	available in the given catalog
ResultSet getIndexInfo (...)	Retrieves a description of the given table's indices and statistics.
int getMaxConnections ()	Retrieves the maximum number of concurrent connections to this database that are possible.
ResultSet getTablePrivileges (...)	Retrieves a description of the access rights for each table available in a catalog.



Tech App

Database metadata can be used to create application which is similar to Microsoft Access or TOAD.

Appendix-A

Regular Expression

Regular Expressions constructs

Characters	
x	The character x
\\	The backslash character
\t	The tab character ('\u0009')
\n	The newline (line feed) character ('\u000A')
\r	The carriage-return character ('\u000D')
\f	The form-feed character ('\u000C')
\a	The alert (bell) character ('\u0007')
\e	The escape character ('\u001B')
Character classes	
[abc]	a, b, or c (simple class)
[^abc]	Any character except a, b, or c (negation)
[a-zA-Z]	a through z or A through Z, inclusive (range)
[a-d[m-p]]	a through d, or m through p: [a-dm-p] (union)
[a-z&&[def]]	d, e, or f (intersection)
[a-z&&[^bc]]	a through z, except for b and c: [ad-z] (subtraction)
[a-z&&[^m-p]]	a through z, and not m through p: [a-lq-z](subtraction)

Predefined character classes

.	Any character (may or may not match line terminators)
\d	A digit: [0-9]
\D	A non-digit: [^0-9]
\s	A whitespace character: [\t\n\x0B\f\r]
\S	A non-whitespace character: [^\s]
\w	A word character: [a-zA-Z_0-9]
\W	A non-word character: [^\w]

POSIX character classes (US-ASCII only)

\p{Lower}	A lower-case alphabetic character: [a-z]
\p{Upper}	An upper-case alphabetic character:[A-Z]
\p{Digit}	A decimal digit: [0-9]
\p{Alpha}	An alphanumeric character:[\p{Alpha}\p{Digit}]
\p{Punct}	Punctuation: One of !"#\$%&'()*+,-./;:<=>?@[\\]^_`{ }~
\p{Blank}	A space or a tab: [\t]
\p{XDigit}	A hexadecimal digit: [0-9a-fA-F]
\p{Space}	A whitespace character: [\t\n\x0B\f\r]

Boundary matchers

^	The beginning of a line
---	-------------------------

\$	The end of a line
\b	A word boundary
\B	A non-word boundary
\A	The beginning of the input
\G	The end of the previous match
\Z	The end of the input but for the final terminator, if any
\z	The end of the input

Greedy quantifiers

$X^?$	X , once or not at all
X^*	X , zero or more times
X^+	X , one or more times
$X^{\{n\}}$	X , exactly n times
$X^{\{n,\}}$	X , at least n times
$X^{\{n,m\}}$	X , at least n but not more than m times

Reluctant quantifiers

$X^{??}$	X , once or not at all
$X^{*?}$	X , zero or more times
$X^{+?}$	X , one or more times
$X^{\{n\} ?}$	X , exactly n times

$X\{n,\}?$	X , at least n times
$X\{n,m\}?$	X , at least n but not more than m times
Possessive quantifiers	
$X?+$	X , once or not at all
$X*+$	X , zero or more times
$X++$	X , one or more times
$X\{n\}+$	X , exactly n times
$X\{n,\}+$	X , at least n times
$X\{n,m\}+$	X , at least n but not more than m times
Logical operators	
XY	X followed by Y
$X Y$	Either X or Y
(X)	X , as a capturing group

Appendix-B

NIO (Non-Blocking IO)

The `java.nio.*` package introduced a java “new” I/O library-The Java NIO called Non-blocking I/O. It supports a channel-based approach to I/O operations. It is built on two fundamental items, channels and buffers. The buffer is pushed into the channel. The channel either retrieves or loads data into the buffer. It is used in writing high-performance, large-scale applications. In Java I/O data is transferred from source to destinations one byte at a time. Data is not cached anywhere. NIO transfers data from source to destination using blocks. So, data is always read into a buffer. Naturally, NIO applications are faster than I/O applications. NIO applications are performance based applications.

The main difference between stream and block is, when a thread invokes a `read()` or `write()` method. That thread is blocked until it reads to data or data is fully written. The thread is not performing anything in the meantime. Generally, Java I/O’s streams are blocking streams, hence until full data is read or written into the data stream no task can be performed, naturally it affects the performance. In case of NIO (Non-blocking I/O) when a thread invokes a `read()` or `write()` method the data is read from channels. In addition, the trhead reads the data which is currently available, rather than getting blocked until data becomes available for reading. This frees up the thread to perform any other operation while the data is being collated in the buffer. Same condition applies to writing the data as well. This approach ensures that the read/write calls do not become blocking calls.

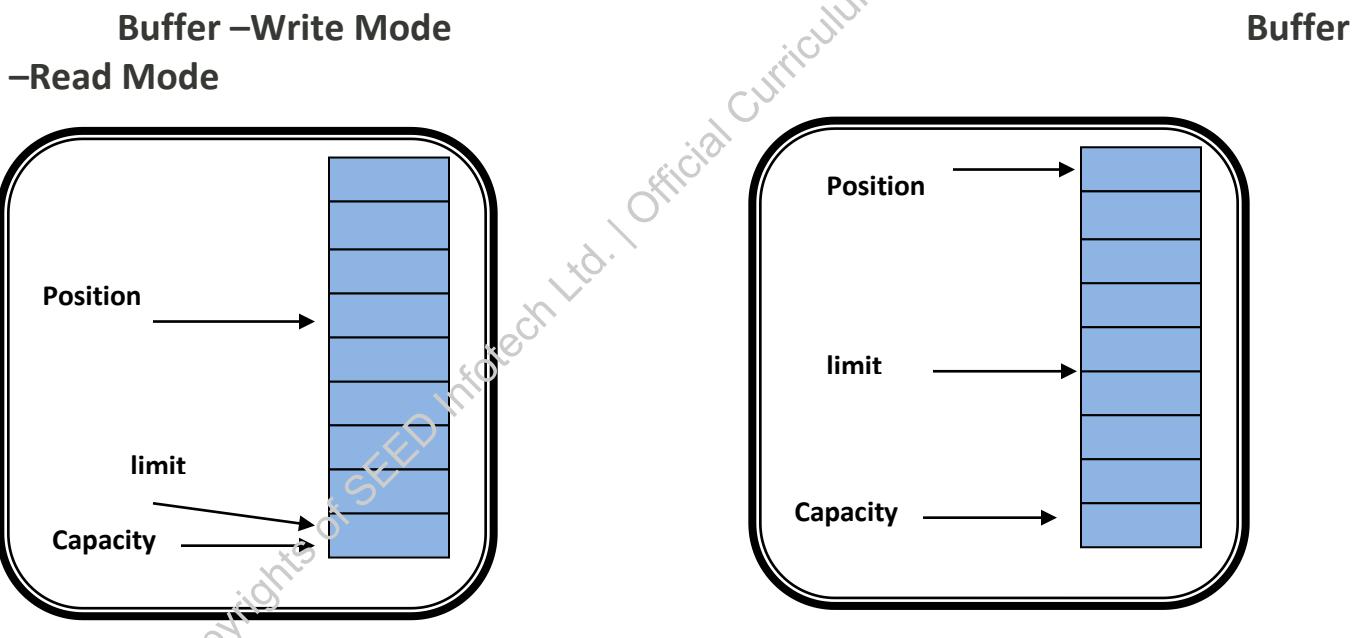
NIO-Buffer

A buffer is a container for data of a specific primitive type. A buffer is a linear, finite sequence of elements of a specific primitive type. The essential properties of buffer are its capacity, limit and position.

Capacity: A number of elements it contains. It can never be negative and can never change.

Limit: index of the first element that should not be read or written. Never negative and is never greater than its capacity. A limit is a variable which specifies how much data at a time to get (while writing) and put (while reading).

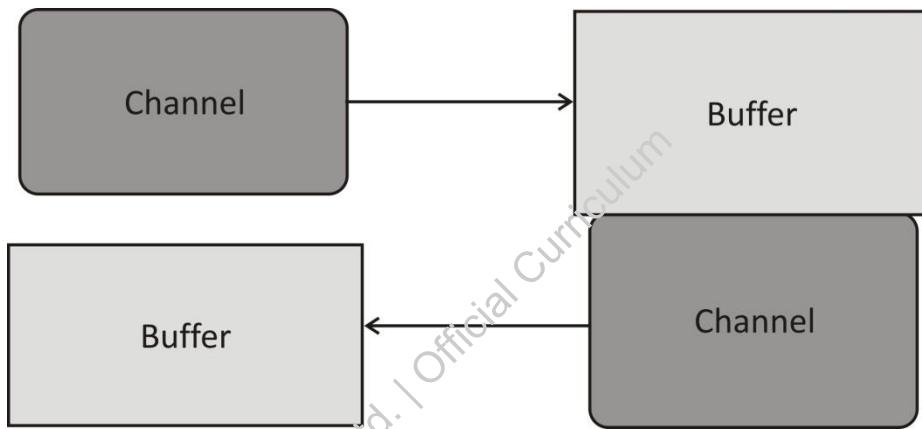
Position: The index of the next element to be read or written. It is never negative and never greater than the limit. The position variable keeps track of how much data has been written or read.



NIO-Channel

A channel is an object or entity through which data can be read or written. It is similar to the I/O streams. Data can't be read or written directly through Channels. Data is transferred from channels to buffers and the applications reads the bytes from the buffer.

A channel represents an open connection to an entry such as program component that is capable of performing one or more distinct I/O operations such as reading or writing. NIO channels can be asynchronously closed. If a thread is blocked in I/O operation on a channel, another thread can close that channel.



FileLock

File locking is now possible using NIO package. A file lock is either exclusive or shared. A shared lock prevents other concurrently-running programs. No processes or an application accessing the file is a exclusive lock. Java provides `lock()` and `tryLock()` methods in `FileChannel` class for getting a lock over the file object.

A File lock is either exclusive or shared. A shared lock prevents other concurrently-running programs. An exclusive lock prevents other programs from acquiring an overlapping lock of either type.

Client-Server Application Development Using Java

Lab Manual and Appendix

▶ ▶ ▶ **Contents**

Sr. No.	Chapter Name	Page No.
1.	Introduction to Lab Manual	364
2.	Java Swing (Basic Controls)	366
3.	Java Swing(Menu, Dialog, GridBag)	370
4.	Java Swing(Containers, MVC)	374
5.	Java Multithreading Basics	379
6.	Java Thread Communication	380
7.	Java Applets	383
8.	Input-Output(File IO)	386
9.	Java Sockets	389
10.	Java Database Connectivity(JDBC)	393
11.	Appendix C	395
12.	Appendix D	406

Introduction to Lab Manual

This lab manual aims at giving complete understanding of the core concepts of the topic and applying them in the exercises.

The lab manual consists of a set of lab exercises defined chapter wise. Each exercise has a definite objective defined. These objectives map with the terminal objectives defined at the beginning of each chapter. The problem statement is defined with clear instructions.

Some exercises have specific configuration or pre-condition mentioned. Advanced lab exercises are also given for some topics.

Appendix A given at the end of lab manual contains stepwise instructions to use eclipse IDE.

Configuration

jdk 1.7 should be installed. The lab exercises should be coded using Java language. Specific configuration related to a particular exercise is mentioned. If it is not mentioned then the above configuration should be considered.

Structure of Lab Manual

Lab manual consists of different sections. The explanation of each structure is given below.

Objective

It states what you will achieve after completing a particular application. At the end of each lab exercise you should keep a track whether the objectives of that session are achieved.

Configuration

It is optional section and is present for specific lab exercises. If absent, then the configuration mentioned earlier should be considered.

Pre-condition

You should have understood the concepts explained in a particular chapter in the courseware thoroughly to solve the exercises mentioned.

But some lab exercises will have pre-condition explicitly mentioned.

Problem Statement

Problem statement for each lab exercise is given. It defines clear instructions to achieve the defined objective.

How to use the Lab Manual?

Whenever a programmer has to transform a problem statement into a program which a computer can execute, you should split the activity in the following manner:

- Read the problem statement carefully. Hint is given for some problem statements to help you to solve the problem.
- Preparation
 - Decide the User interface (CUI or GUI) before hand.
 - Write the algorithm or steps to be followed to solve the problem. You can also draw flowcharts if required.
 - The program is made modular by writing functions. So pen down expected function prototypes and arguments on paper.
 - Also decide how one module (function) would communicate with other module (function).
- Give a dry run to the algorithm written.
- Write the code.
- Execute the code.

Coding Practices

The maintenance of code is easy if the code is written using coding practices. You should follow following coding practices while solving the lab exercises in this lab manual:

- Use meaningful names for variables, functions, file.
- Use uniform notation throughout your code.
- Code should be properly indented.
- Code should be commented. While documenting your code, write the purpose of your piece of code. What task is assigned to a method, what arguments are passed to it, what it returns should be clearly stated. Write a clear comment if you have added any statements for testing purpose.

Writing a right kind of well-documented software is an art along with your technical skills. Enough necessary documentation should be done. This makes the code easily maintainable.

Chapter 1 - Java Swing (Basic Controls)

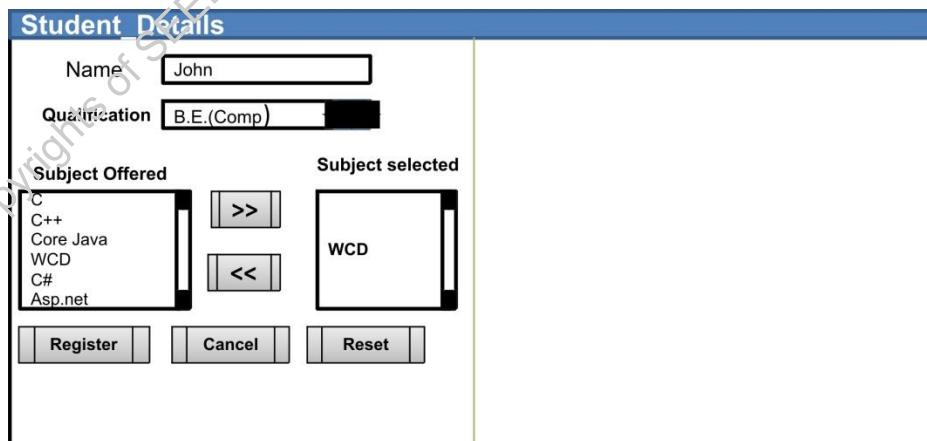
Lab Exercise - 1

Objectives

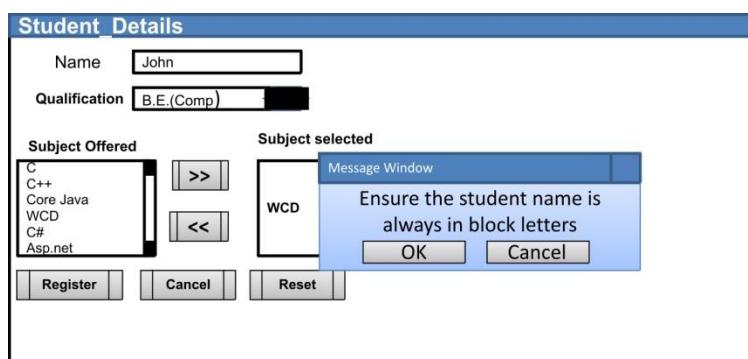
- Construct an application UI using basic controls like Button, TextField, ComboBox, radio buttons, password fields, check boxes, lists etc.
- Develop an application to perform some operation on button click.

Problem Statement

Construct a Student registration form for ABC institute. It should accept the details of a student. Use appropriate Swing controls.



- a. Name field should not be left blank.
- b. Ensure the student name is always in block letters. It should display message box if data is not entered in required format or it can be converted in upper case irrespective of the case user enters it.



- c. Courses should populate in list box when the form is loaded.

- d. Course selected in the list box should be added to adjacent list box on the click of the button. The other button should help to delete the item in the second list box if the items need to be de-selected.
- e. Data for qualification of the student should be populated in the combo box at design time.

Lab Exercise - 2

Objective

- Develop an application to perform some operation on event triggered by a UI component.

Problem Statement 1

Create a GUI application to accept salary details from user like basic salary, HRA, DA, PF, I Tax to find gross salary and net salary.

Salary_Details

Name	<input type="text" value="John"/>
BasicSalary	<input type="text" value="3000"/>
HRA	<input type="checkbox"/>
DA	<input type="checkbox"/>
Total Salary <input type="text"/>	
PF	<input type="checkbox"/>
I Tax	<input type="checkbox"/>
Salary in Hand <input type="text"/>	
calculate	

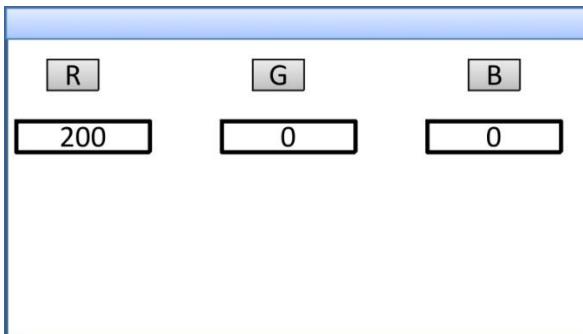
Salary calculation is based on following steps:

- a. Enter basic salary of the employee.
- b. Calculate HRA; HRA is 10% of the basic salary.
- c. Calculate DA; DA is 15% of the basic salary.
- d. Total Salary = BasicSalary + HRA + DA

- e. Calculate PF; PF is 12% of Total Salary
- f. Calculate I Tax; I Tax is 2% of Total Salary
- g. When user of this system clicks on calculate button; net salary in Hand text field should show salary figure.
- h. Salary in Hand = Total Salary - (PF + I Tax)
- i. Net salary in hand text field is not editable.

Problem Statement 2

Create a GUI application which accepts three numbers from user and when user clicks on frame. It should show background color of JFrame.



- a. Text field range is 0 to 255. Text field should not accept greater than 255 and less than 0.
- b. Text field should not accept any character.

Hint: Use Mouse Event.

Expected Result:



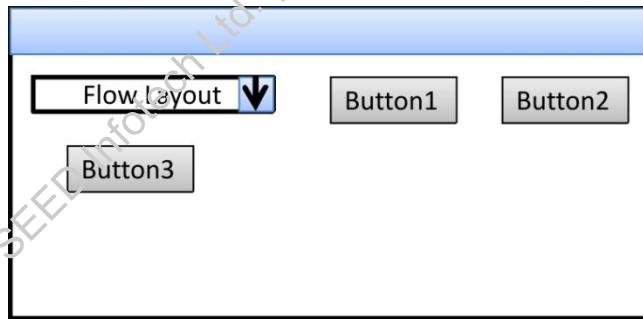
Lab Exercise - 3

Objective

- Arrange the components using simple layouts like Border, Grid and Flow layout.

Problem Statement

Create a GUI application which arranges same the components in Border Grid and Flow layout.



- a. GUI frame should contain Combo box and 3 buttons.
- b. When user selects ‘Flow Layout’ all the 4 components should be arranged in a FlowLayout.
- c. When user selects ‘Grid Layout’ all the 4 components should be arranged in GridLayout.
- d. When user selects ‘Border Layout’ all the 4 components should be arranged in BorderLayout.

Chapter 2 - Java Swing (Menu, Dialog, Grid Bags)

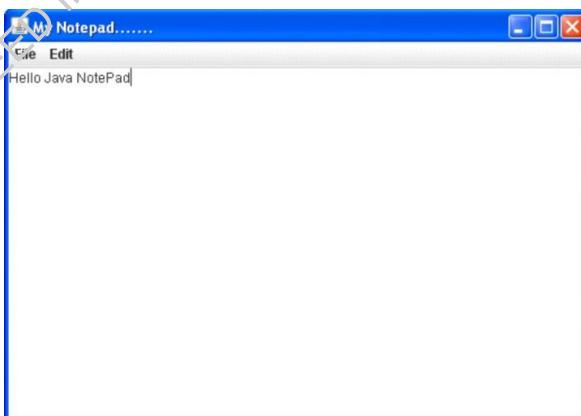
Lab Exercise - 4

Objectives

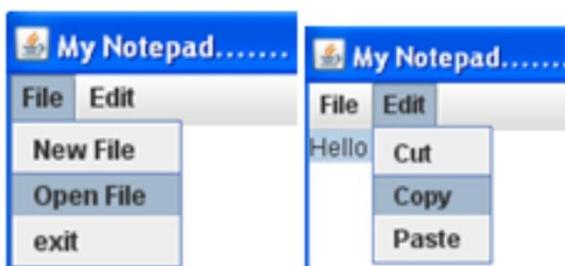
- Integrate built-in dialogs like File, Color Dialog etc. as part of the application (e.g. Text Editor).
- Use controls like Menu, Popup Menu and Dialog.

Problem Statement

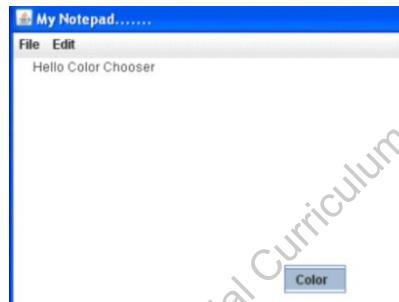
Design the Text Editor as shown below:



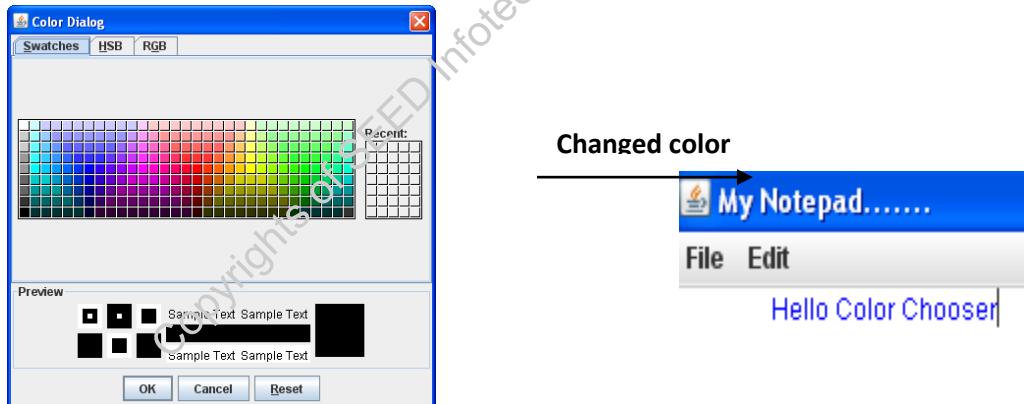
- a. File, Edit menu should look like following:



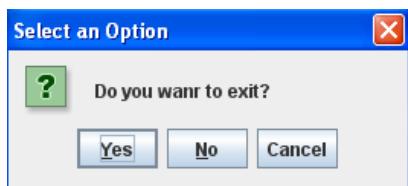
- b. The popup menu should be displayed on right button click. Popup menu displays option "Color".



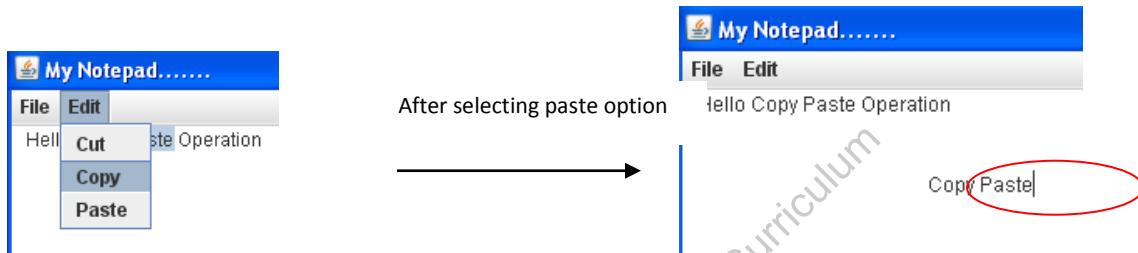
- c. ColorChooser dialog should be displayed at the click of "Color" option.
- d. After selecting the text from editor and colour from the dialog, text colour should be changed.



- e. Option 'New' should open a new client area.
- f. Option 'Open' should display Open File Dialog box. The file should be opened in the client area.
- g. Option 'Exit' should display dialog box. With 'Yes, No, Cancel' permission. If user selects 'Yes' then Notepad should be closed.



- h. Option 'Copy' from Edit menu should enable copying of text into clipboard.
- i. Option 'Paste' from Edit menu should enable copying text from clipboard.
- j. Option 'Cut' from Edit menu should enable cutting text from the document and copying into clipboard.



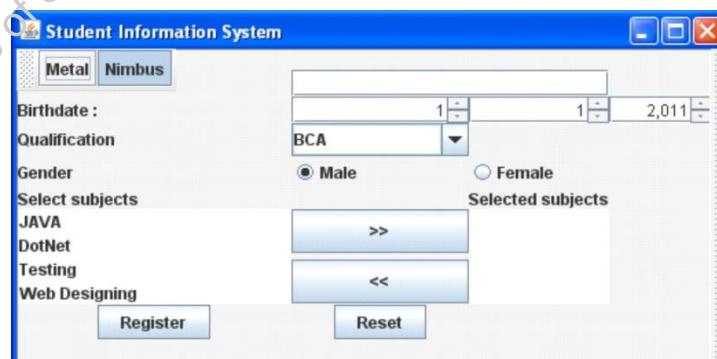
Lab Exercise - 5

Objective

- Create UI using GridBagLayout.

Problem Statement

Design 'Student Information Form' using GridBagLayout



The Form contains attributes like:

- Name: use text field
- Birthdate: use JSpinner
- Qualification: use combo box
- Gender: use radio buttons
- Select subjects and selected subjects: list box
- Four button components.

When Student clicks on register button the dialog box displays "Hello <student_name>!!! You have been successfully registered".

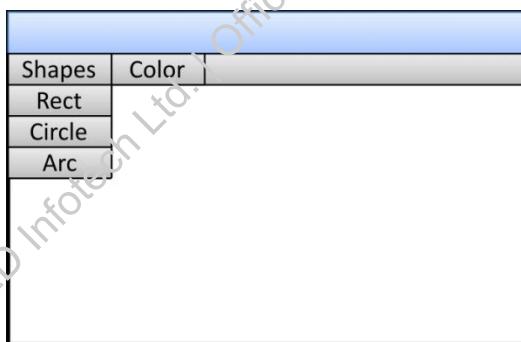
Lab Exercise - 6

Objective

- Working with a 2D graphics.

Problem Statement

Design ‘Mini PaintPad’ Application.



- a. Option ‘Rect’ should draw a Rectangle.
- b. Option ‘Circle’ should draw a Circle.
- c. Option ‘Arc’ should draw an Arc.
- d. Option ‘Color’ should color chooser dialog.
- e. Select color to fill the shapes.

Chapter 3 - Java Swing (Events, Containers, MVC)

Lab Exercise - 7

Objectives

- Use Adapter classes for event handling.

Problem Statement

Create an application which displays tooltip when mouse cursor hovers over a component.



1. Create a simple Swing JFrame and JTextField.
2. User hovers mouse on JTextField.
3. It should show tool tip message.

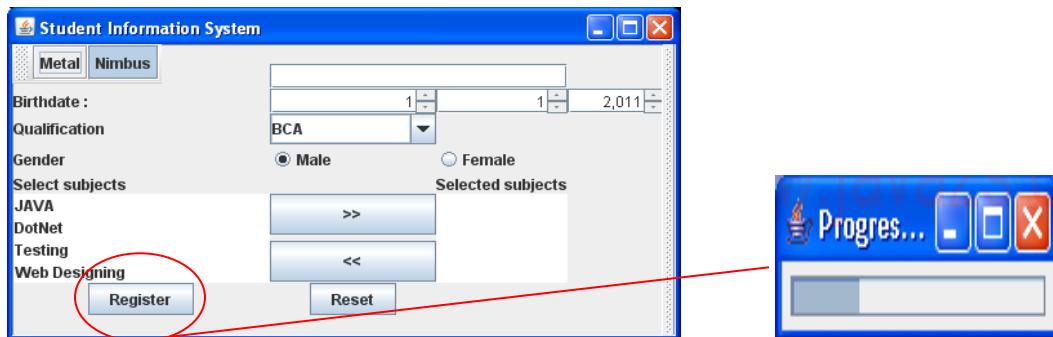
Lab Exercise - 8

Objective

- Use display controls like JProgressBar, JToolTip in an application.

Problem Statement

Simulate Data storage of the 'Student Information Form' using JProgressBar.



When user clicks on ‘Register’ button, progress of data saving should be simulated using JProgressBar.

When progress completes, it should show dialog message “Successful Registration”. Provide tool tip while student enters the information.

Lab Exercise - 9

Objective

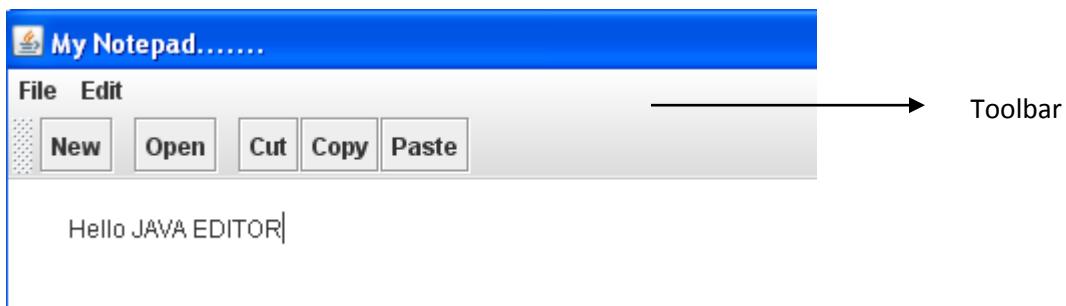
- Use general purpose containers like JToolBar, JTabbedPane, and JSplitPane.

Pre-conditions

- Use Text Editor Assignment (Chapter 2, Lab Exercise 1, and Problem statement 1).
- Use Student Information System Assignment (Chapter 1, Lab Exercise 1 and Problem statement)

Problem Statement 1

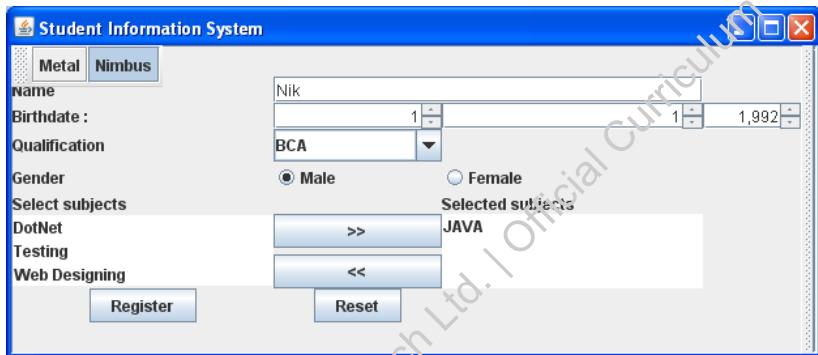
Use the same problem statement and construct the following:



- a. The editor should have a tool bar with tools like ‘new, open, cut, copy, paste’.
- b. Performing cut, copy and paste operations like the ‘Text Editor’ assignment.

Problem Statement 2

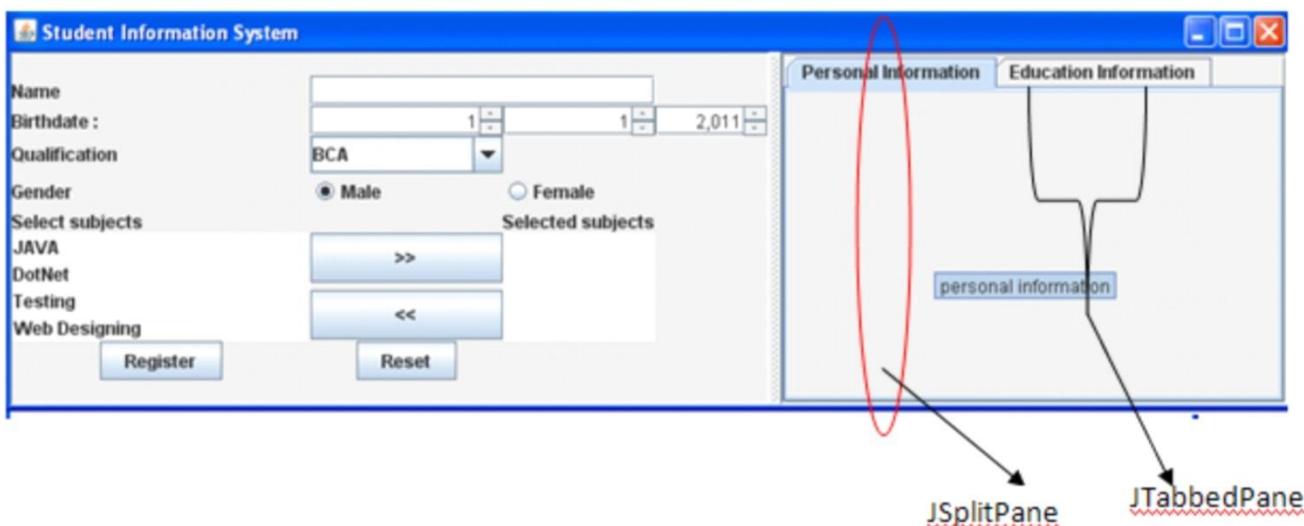
Use the same problem statement and construct the following:



Enter all the information and click on 'Register Button'.

The information should be displayed in tabs.

- Personal information should be displayed in 'personal information tab'.
- Education Information should be displayed in 'educational information tab'.
- Separate the form and tabs using split pane.



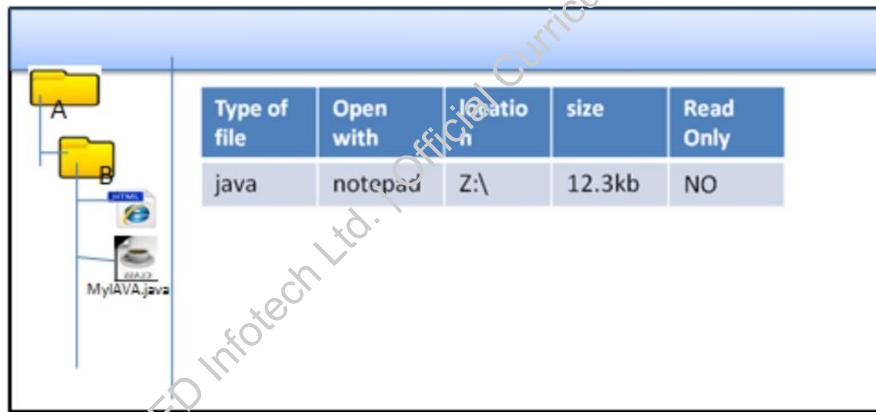
Lab Exercise - 10

Objective

- Use advance UI controls like JTree, JTable, and JSpinner.

Problem Statement

Create a GUI application where all the files should be displayed in tree format. When user selects any specific file the data should be displayed in Table form.



Lab Exercise - 11

Objective

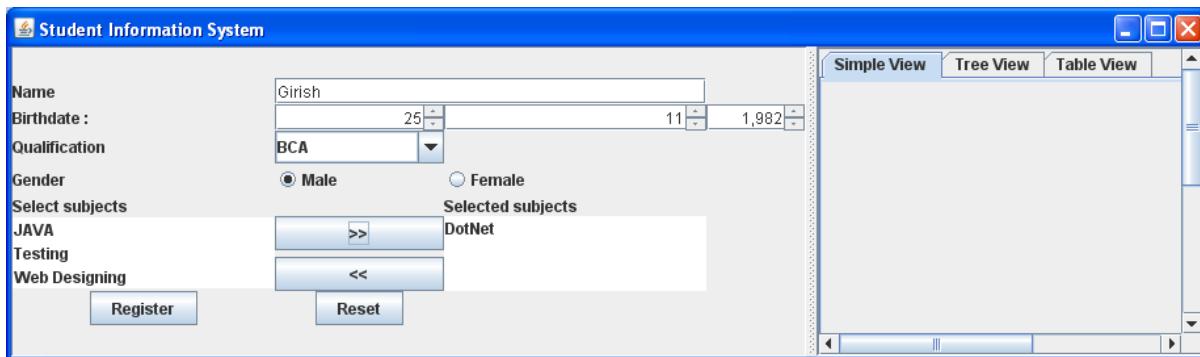
- Understand the concept of Model-View-Controller in Swing.

Pre-condition

- Use Student Information system Assignment (Chapter 1, Lab Exercise 1 and Problem statement)

Problem Statement

Create a GUI application to perform following activities:



Show the student information in the three different views

- Simple view
- Tree view
- Table view

Use MVC to show all three views.

Expected Result:

The screenshot displays a user interface for viewing student information across three tabs: Simple View, Tree View, and Table View. The Tree View tab is currently selected.

Simple View: Shows the student's name and birthdate.

Tree View: Shows a hierarchical tree structure of student information. The root node is "Student Information", which branches into "Girish", "Birthdate" (with value "25/11/1982"), "Gender" (with value "Male"), and "Subjects" (with value "DotNet").

Table View: Shows the student's details in a tabular format.

Name	Birthdate	Gender	Sub
Girish	25/11/1982	Male	[DotNet]

Chapter 4 - Java Multithreading Basics

Lab Exercise - 12

Objectives

- Create Thread using Thread class.
- Create Thread using Runnable interface.

Problem Statement1

Develop simple application as shown below. This application should provide concurrent functionality of incrementing by one and a multiplication table.

Accept two numbers from user. First number should be incremented by 1 while a multiplication table for second number should be generated.

Expected output:

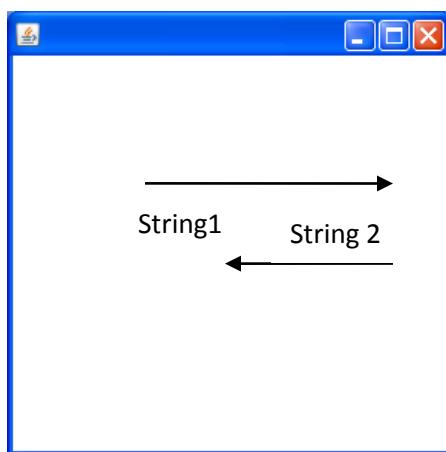
Accept 2 numbers from user: 2 3

Thread-1: 2 3 4 5 6 7 8 9 10 11 ...

Thread-2: 3 6 9 12 15 18 21 24 ...

Problem Statement2

Create an application that scrolls two banners horizontally.



Create a GUI that extends JFrame and implements Runnable. Draw two strings using paint() method and move them horizontally in opposite direction.

Chapter 5 –Java Thread Communication

Lab Exercise - 13

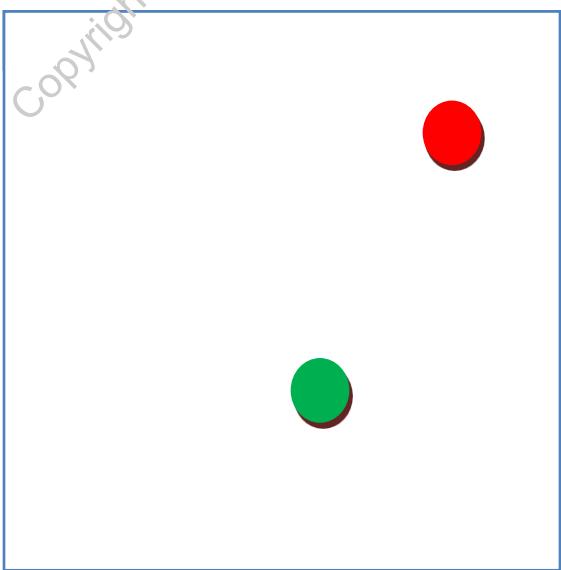
Objective

- Monitor Object methods like `wait()`, `notify()`, `notify All()`.

Problem Statement

Move 3 balls on the screen (red, blue and green) with different speeds. Red is fastest followed by green and blue is the slowest. Whichever ball reaches 300 pixel locations to the right from starting position waits for other balls to arrive. When all three balls have reached the same location, they restart their movement with corresponding speeds.

Expected Result:



Lab Exercise - 14

Objective

- Create an application that demonstrates usage of Lock and Reentrant Lock.

Problem Statement

Create a banking domain application simulating concurrent deposit and withdraw.

Hint:

Following is the signature of the method:

```
public int withdraw (Account account, int amount)
{
    try {implicitLock.lock();}
    finally {implicitLock.unLock();}
}

public int deposit (Account account, int amount) { . . .
. }
```

Lab Exercise - 15

Objectives

- Construct synchronised code.
- Differentiate between synchronized method and block.

Problem Statement 1

Create a banking domain application simulating concurrent deposit and withdrawal. Use synchronized method for the transaction.

Hint:

The signature of the method is as follows:

```
public synchronized int withdraw (Account account, int
amount)
{ . . . }

public synchronized int deposit (Account account, int
amount)
{ . . . }
```

Problem Statement 2

Create a banking domain application simulating concurrent deposit and withdraw

Use synchronized block.

Hint:

The signature of the method as below:

```
public int withdraw (Account account, int amount)
{synchronized (this){. . .}
}
public int deposit (Account account, int amount)
{synchronized {. . .}
}
```

and amount is say 2000/-rupees

Chapter 6 - Java Applets

Lab Exercise - 16

Objective:

- Describe Applet Life Cycle.

Configuration:

- Check if IE is installed.

Problem Statement

Create an Applet to check the life cycle methods of applets.



- a. Write `paint()` method to display 'Hello Applet'.

Lab Exercise - 17

Objective

- Use parameters with Applet Tag.

Pre-condition:

Refer Lab Exercise-1, Problem Statement-1.

Problem Statement

Refer 1st assignment to do following changes.

- a. Create HTML file and set the parameters in HTML using param tag
 - i. Size= 20

- ii. Font= Arial
- b. 'Hello Applet' should be in
 - i. Text size 20
 - ii. Text font Arial

Pass these values as parameters.



Lab Exercise - 18

Objective:

- Construct a application to share data between two applets using JavaScript.

Problem Statement

Selecting a colour from one applet should change color of a component in second applet.

Applet1

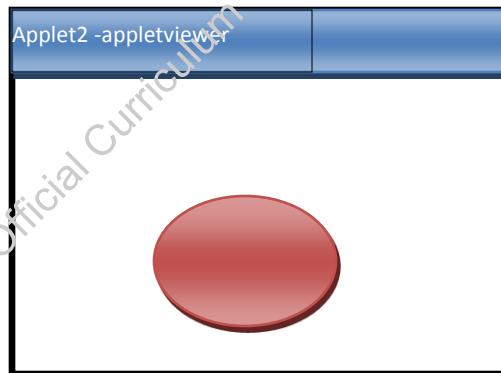
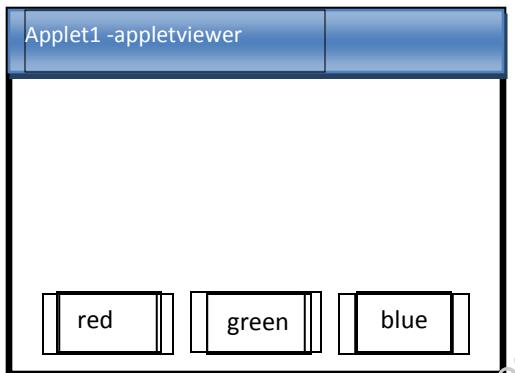
Applet1 contains 3 buttons

Button names: red, green, blue

Applet2

Applet2 contain Oval

Expected Output:



- a. Clicking a specific button should change the colour of the oval (e.g., clicking button red should make the oval red).

Chapter 7 - Java Input-Output

Lab Exercise - 19

Objective

- Perform file and directory operations.

Pre-condition:

- file1.txt and file2.txt should be present in a current directory with some pre-existing text data.

Problem Statement 1

Create an application to check availability of ‘file1.txt’. File information should also be retrieved.

Problem Statement 2

Read the contents from ‘file1.txt’ and overwrite the contents of ‘file2.txt’.

Problem Statement 3

Read and write file information using RandomAccessFile.

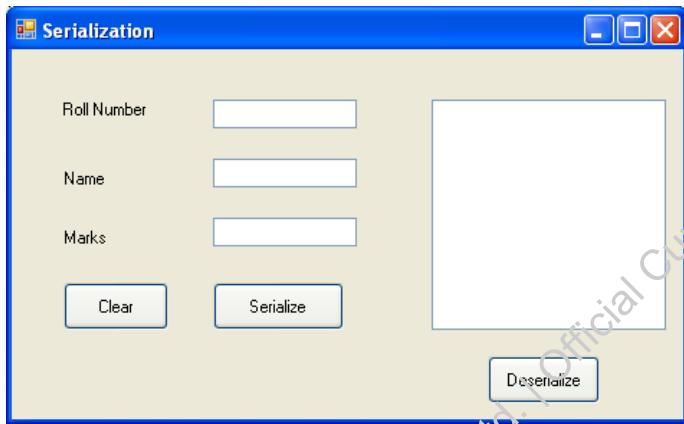
Lab Exercise - 20

Objective:

- Serialize the object using Object Serialization.
- **Pre-condition:**
- emp.txt should be present in a current directory.

Problem Statement

Design the user interface as shown below:



- On clicking “Clear” button, the contents of all text boxes should be cleared and the focus should be set on the first text box.
- On clicking “Serialize” button, the student object should be serialized to a binary file.
- On clicking “Deserialize” button, the student object should be de-serialized and the details should be displayed in list box.

Lab Exercise - 21

Objective

- Construct an application that uses console for I/O operations.
- Construct an application that filters data based on pattern matching using Regular Expressions (Regex).
- **Pre-condition**
- emp.txt and console.txt should be present in a current directory.

Problem Statement 1

Accept the input from console and write in a file ‘console.txt’ using System class.

- Accept an input like “Hello this is System class demo” and store it in the console.txt file.
- Print this message on console.

Problem Statement 2

Accept the input from console and write in a file ‘console.txt’ using Scanner class.

- a. Accept an input like “Hello this is Scanner class demo” and store it in the console.txt file.
- b. Print this message on console.

Problem Statement 3

Accept the input from console and write in a file ‘console.txt’ using Console class.

- c. Accept an input like “Hello this is Console class demo” and store it in the console.txt file.
- d. Print this message on console.

Problem Statement4

Read Employee names whose names like start with character range (Aa-Dd) from emp.txt file.

- a. emp.txt file contain 10 employee names.
- b. Read only those names that start in pattern ‘Aa-Dd’.
- c. Write these contents on Console.

Chapter8 - Java Sockets

Lab Exercise - 22

Objective

- To create applications that will enable remote communication using Sockets.

Problem Statement

Create an application which accepts file name and transfers it to a server if it exists. The server transfers file metadata to the client where it is displayed to the client.

e.g.

Client.java

Perform following activities:

- a. Check whether file exists or Not
- b. If exists than show file information.

Like read-only or not, Size of the file, location of that file.

Show this information to User.

Hint:

a. Client.java -----→Server.java

Enter File Name: file name received

Enter a file name in a client window so that server accept or receive the file name

b. Client.java-----→Server.java

c. Server will check file exist or not and if file is available than search all the file information. Client.java←-----Server.java

d. Server searches the file information and shows this information to client window.

Use TCP/IP connection.

Lab Exercise - 23

Objective:

- Hands-on with TCP/IP commands.

Problem Statement

Execute the following TCP/IP commands on command prompt and check the result.

- a. ipconfig
- b. ping
- c. netstat
- d. ftp
- e. telnet

Expected Result:

```
Z:\>ping 120.3.224.2

Pinging 120.3.224.2 with 32 bytes of data:

Reply from 120.3.224.2: bytes=32 time=246ms TTL=109
Reply from 120.3.224.2: bytes=32 time=271ms TTL=109
Reply from 120.3.224.2: bytes=32 time=277ms TTL=109
Reply from 120.3.224.2: bytes=32 time=261ms TTL=109

Ping statistics for 120.3.224.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 246ms, Maximum = 277ms, Average = 263ms

Z:\>ipconfig

Windows IP Configuration
```

Ethernet adapter Local Area Connection:

```
Connection-specific DNS Suffix. :  
IP Address. . . . . : 172.16.3.96  
Subnet Mask . . . . . :  
255.255.248.0  
Default Gateway . . . . . : 172.16.7.1
```

Z:\>netstat -b

Active Connections

Proto	Local Address	Foreign Address
State	PID	
TCP	SAKETK:1216	bryant.panchsheel.seedinfotech: netbios-ssn ESTA
BLISHED	4	
	[System]	

Lab Exercise - 24

Objective:

- Construct a program using UDP socket
- **Pre Condition:** Refer Lab Exercise-1, Problem Statement-1

Problem Statement

Refer same description of Lab Exercise-1, Problem Statement-1 and show information to user. Use UDP connection.

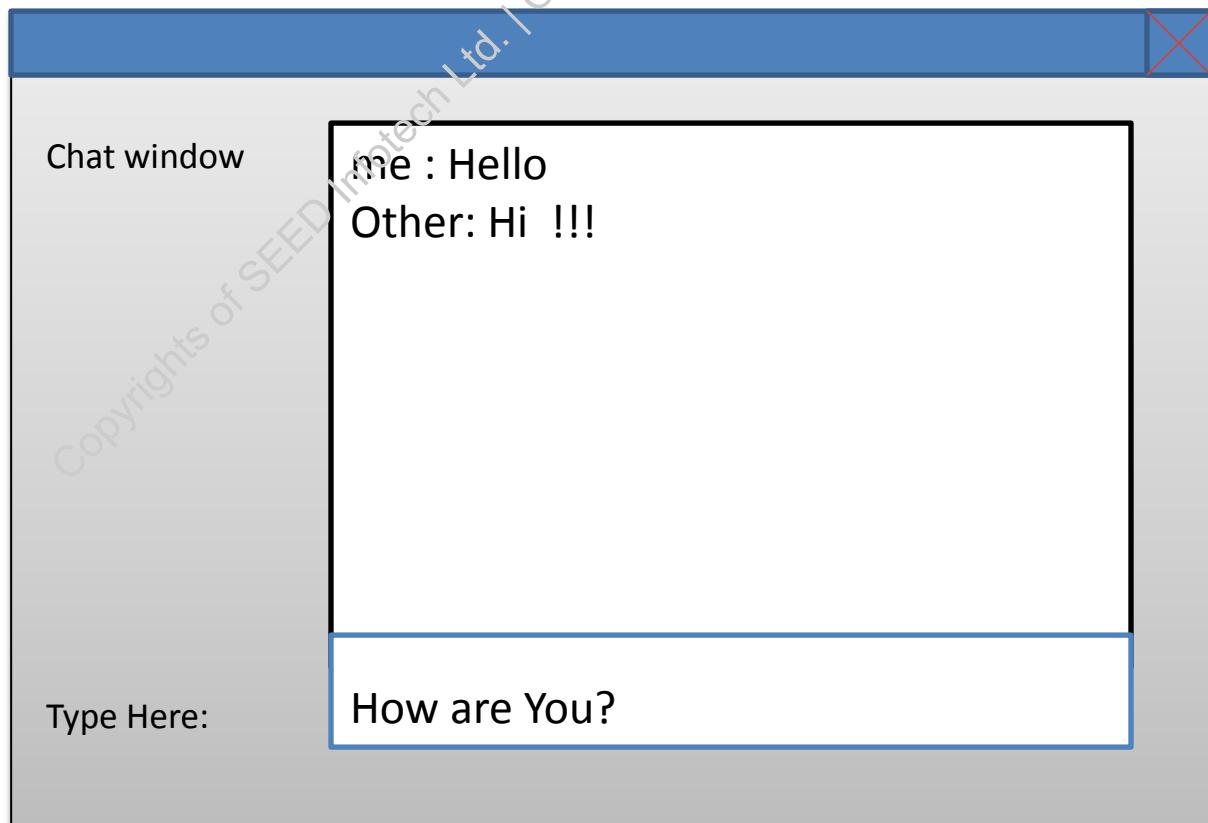
Lab Exercise -25

Objective:

- Create a chat server application using swing.

Problem Statement

Create a GUI screen which contains TextArea and TextField. Text Area is for displaying chat history and Text field for sending a new message.



Chapter 9 - JDBC

Lab Exercise - 26

Objective

- Access data from the tables in the MySQL database and perform operations like insert, update and delete using JDBC.

Configuration: MySQL should be installed.

Pre-condition

- employee table should be present in the database. The table should have fields employee_id, name, salary and deptno.

Problem Statement 1

Display all employee information at a command prompt from the employee table.

Problem Statement 2

Insert the following information inside employee table.

employee_id=123, ename=Prashant, salary=12000,
deptno=12

Problem Statement3

Update the employee salary to ₹15000 whose employee_id is 123.

Problem Statement4

Delete all the records from the employee table.

Lab Exercise -27

Objective

- Perform transactions using JDBC.

Pre-condition

- Account table should be present in the database. The table should have fields accno, acc_holder_name, balance.
- account table should contain at least 2 records:

Accno	acc_hoder_Name	balance
1001	Prashant	20000
1099	Anuradha	30000

- For SQL statements to create the structure, refer 'script.txt'.

Problem Statement

Perform the account transaction between Prashant's account and Anuradha's account.

- a. Transfer the ₹2000 from Prashant's account to Anuradha's account.
- b. If transfer operation fails then rollback the transaction.
- c. If transfer amount successfully transfer from Prashant's account to Anuradha's account than commit the transaction.

Hint

The signature of the method as below:

```
public void transfer (Account from, Account to, int
amount)
{ . . . }
```

Lab Exercise -28

Objective

- Invoke the stored procedure present in the SQL Server database.

Pre-condition

- Stored procedure named "account_rout" to update the record from the table. It should take accno as a parameter.
- For code of the procedure refer 'script.txt' .

Problem Statement

Create a procedure for ACCOUNT table to update the balance.

Hint

Update the balance by ₹2000.

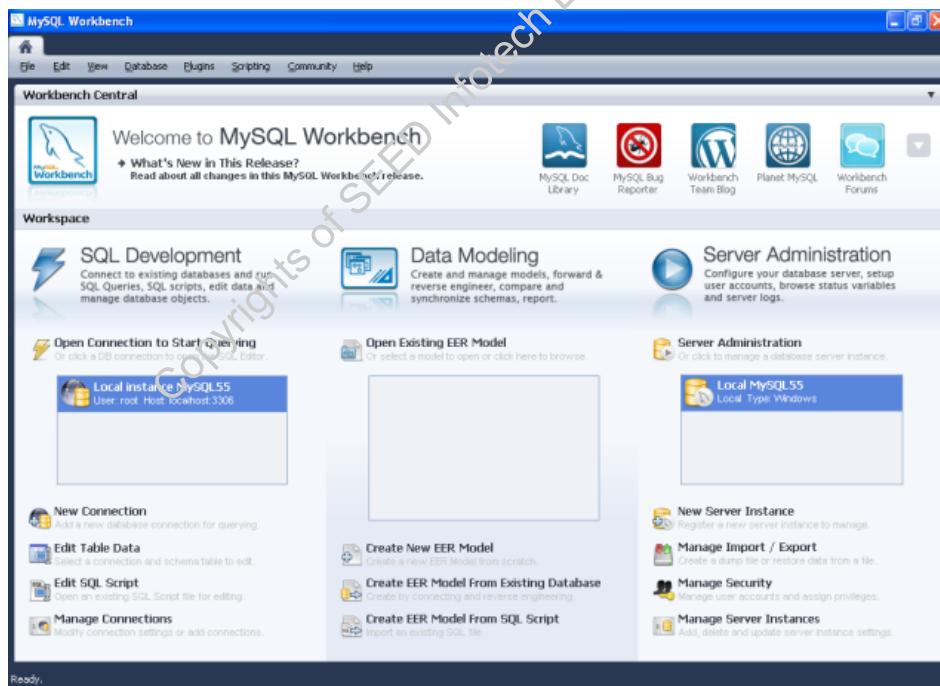
Appendix-C

This document is a guide to use MySQL database.

1. To Open database and establish the connection.

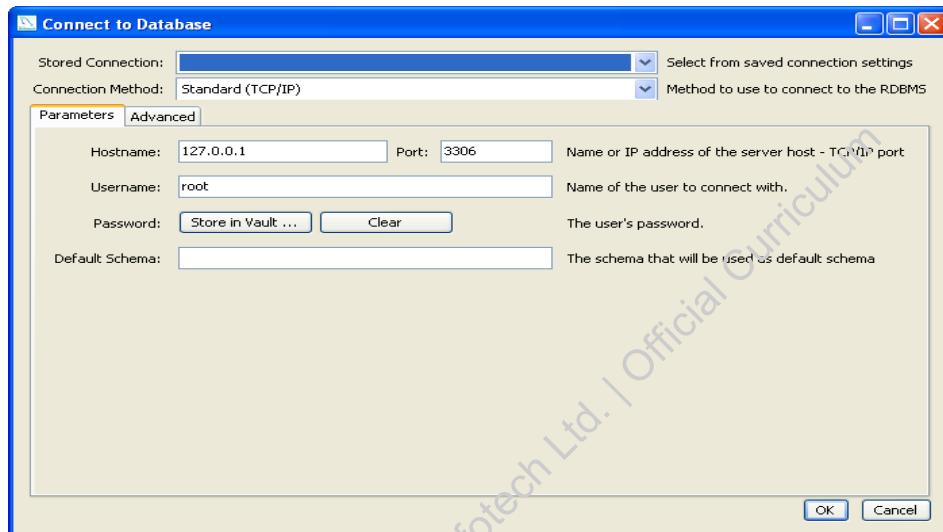
Step 1

Open MySQL database and click on 'open connection to Start Querying' for login purpose.



Step 2

Connect to Database Window opens.



Connect to Database:

Enter database specific information as follows and click on ‘OK’.

- Hostname: (name or IP address of server host) e.g. 127.0.0.1
- Port: (TCP/IP Port) e.g. 3306
- Username: (name of the user to connect with) e.g. root
- Password: (the users password) e.g. myroot
- After that click on OK button.
-

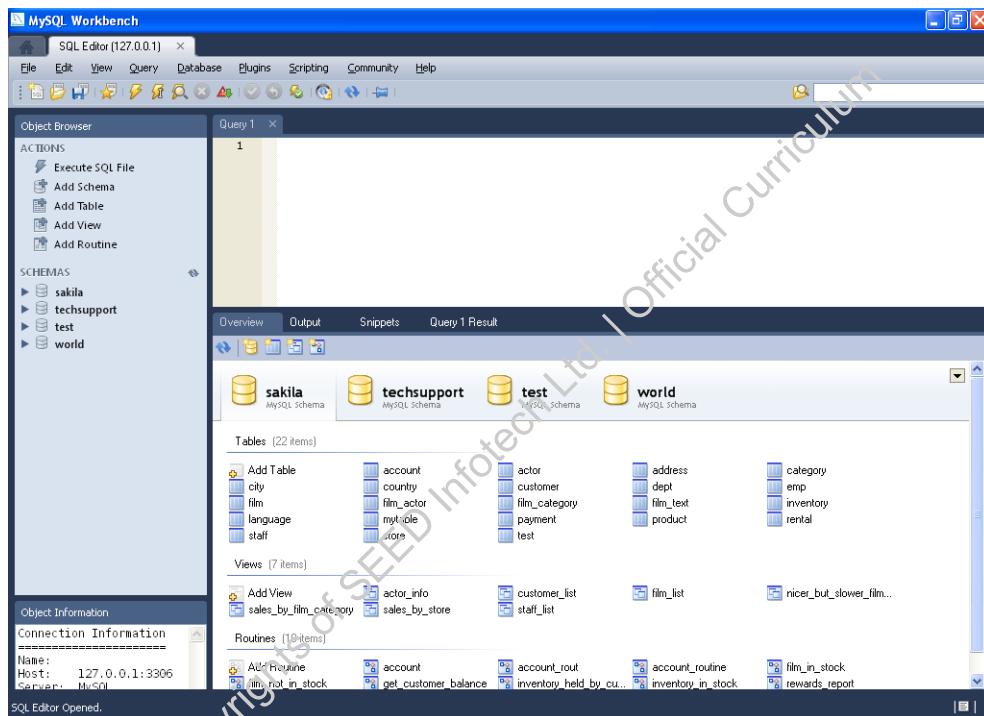
Step 3

Provide the password set during MySQL installation to connect to the MySQL service and click on OK button.

-



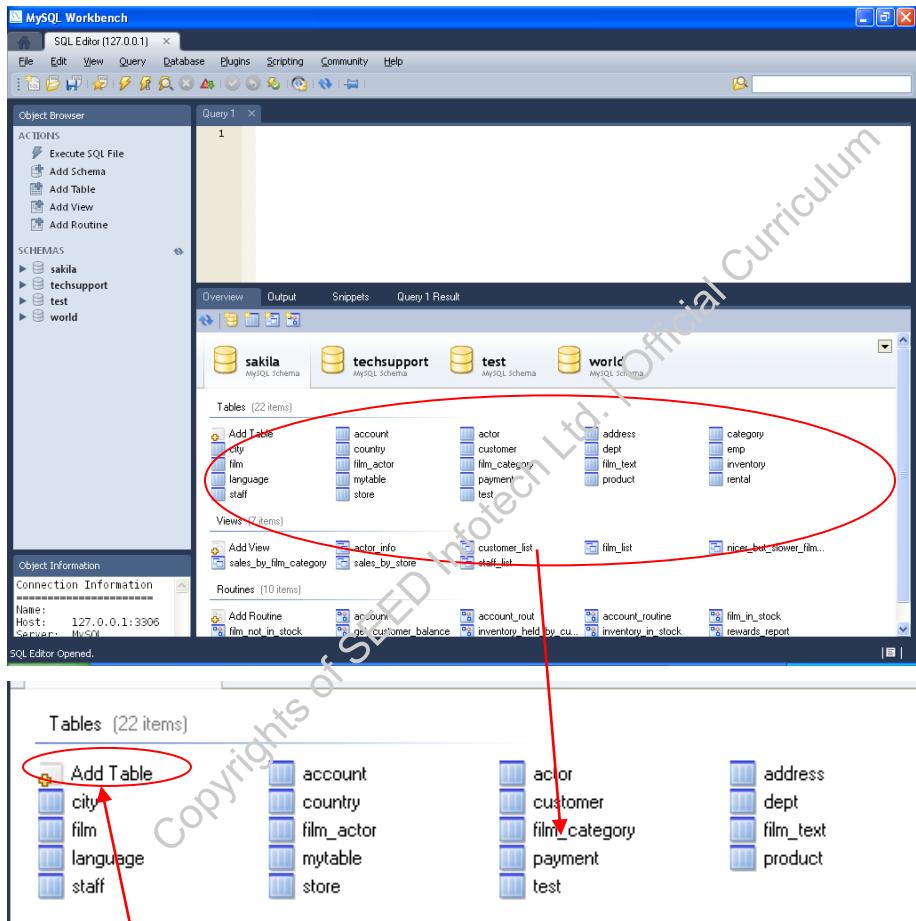
This will open MySQL Workbench window.



MySQL Workbench:

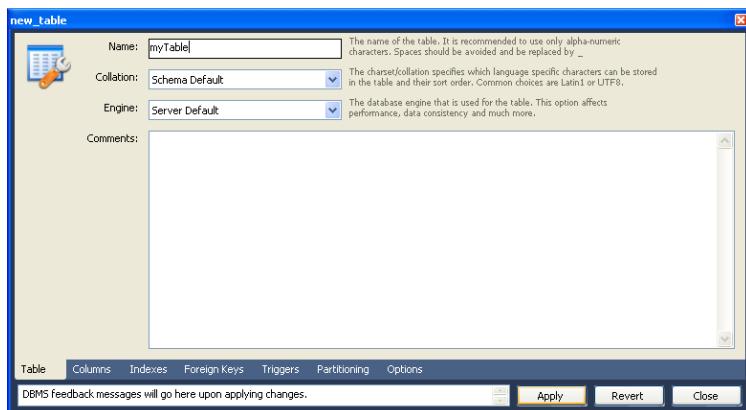
MySQL Workbench is a GUI screen that is used to write SQL statements. It is an SQL editor. Here tables and related operations like select, insert, update and delete can be created using MySQL wizard. Using this wizard, tables, views and SQL stored Procedures can also be created. The Workbench provides some default schemas. Custom schemas can also be created.

2. To create table in MySQL.



Step 1

Click on the 'Add Table' to create a new table. 'new table' window comes into view.



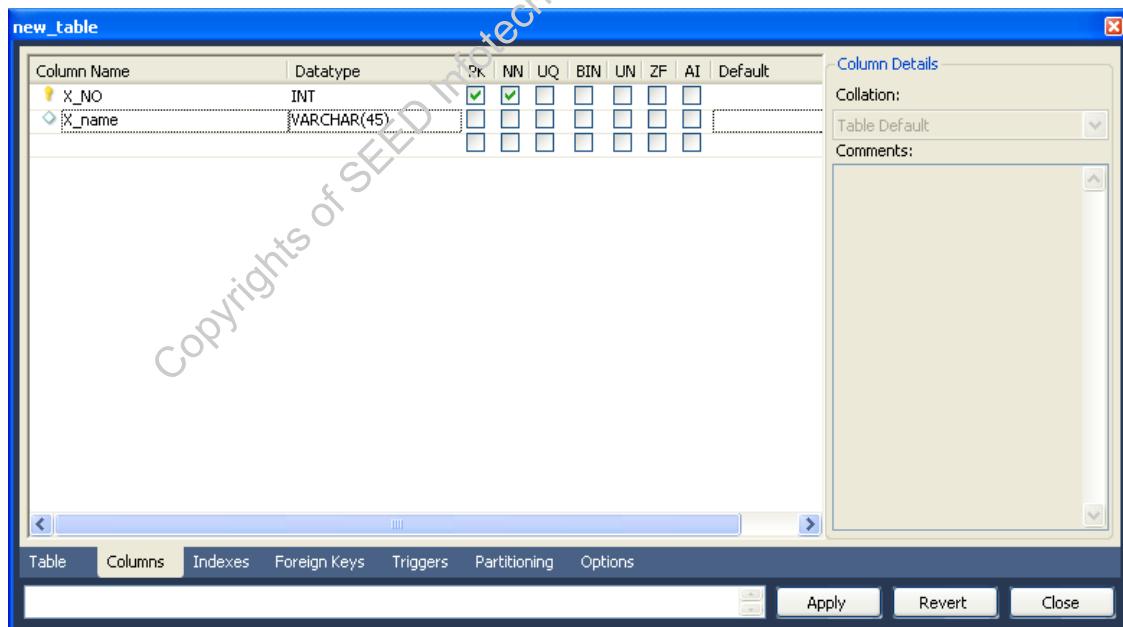
Enter some information like:

- Name: Table name should not contain white spaces. Use underscore (_) to separate words if required. This information is mandatory.

- **Collation:** The charset specifies which language specific characters are valid. Common choices are Latin1 or UTF8. It is recommended to use default value.
- **Engine:** The database engine that is used for table. Recommendation is to use default values.
- **Comments:** optional information.

Step 2

Select Column tab to enter field information.



Provide information related to table columns:

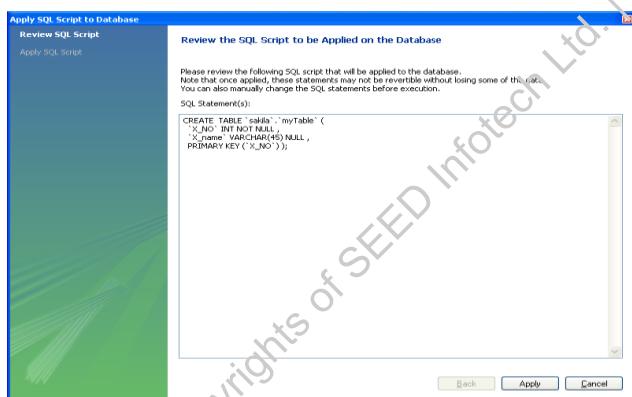
1. Column name
2. Data type
3. Constraints
 - a. PK: Primary Key
 - b. NN: Not Null
 - c. UQ: Unique Index
 - d. BIN: Is Binary Column
 - e. UN: Unsigned data type

- f. ZF: Fill up values to that column with 0's if its numeric
 - g. AI: Auto Incremental
4. Repeat steps 1-3 for all the fields.

Upon completion of column creation process, click on 'Apply' button.

Step 3

Review the SQL script on the database and click on 'Apply' button.



Once you click on 'Apply' button SQL script will be applied to the database. Click on 'Finish' button.

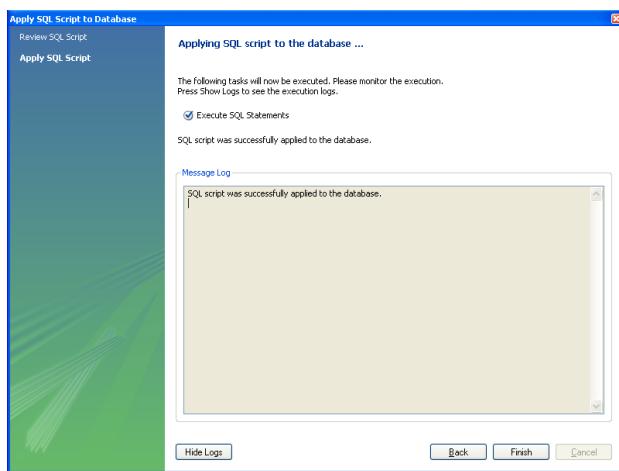


Table will now be created in the database.



The screenshot shows the MySQL Workbench interface. At the top, there is a 'Query 1' window containing the following SQL code:

```

1 delimiter $$ 
2 
3 CREATE TABLE `mytable` (
4     `X_NO` int(11) NOT NULL,
5     `X_name` varchar(45) DEFAULT NULL,
6     PRIMARY KEY (`X_NO`)
7 ) ENGINE=InnoDB DEFAULT CHARSET=utf8$$
8 
9

```

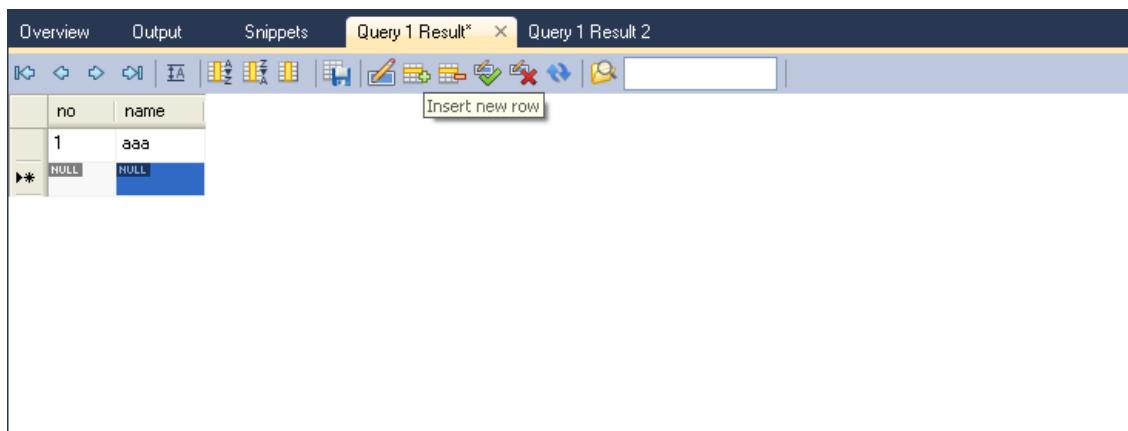
An arrow points from the 'mytable' entry in the 'Tables [23 items]' list below to the 'mytable' entry in the 'Views [7 items]' list below it. The 'mytable' entry in the 'Tables' list is highlighted with a red oval.

3. To insert record in a database table.



The screenshot shows the MySQL Workbench interface with the 'Tables [23 items]' list open. The 'mytable' entry is highlighted with a red oval. An arrow points from the 'mytable' entry in the 'Tables' list to the 'mytable' entry in the 'Views' list below it.

Click on the table name so that Query Result window appears.



The screenshot shows the MySQL Workbench interface with the 'Query 1 Result' tab selected. The window displays a table with two rows:

no	name
1	aaa
NULL	NULL

A button labeled 'Insert new row' is visible at the bottom of the table area.

Enter the values.

OR

Alternative way to insert records in database table is:

- Step 1: Right click on table name
- Step 2: Send to SQL editor
- Step 3: Insert Statement

This is a general syntax of Insert statement where the values can be entered:



```

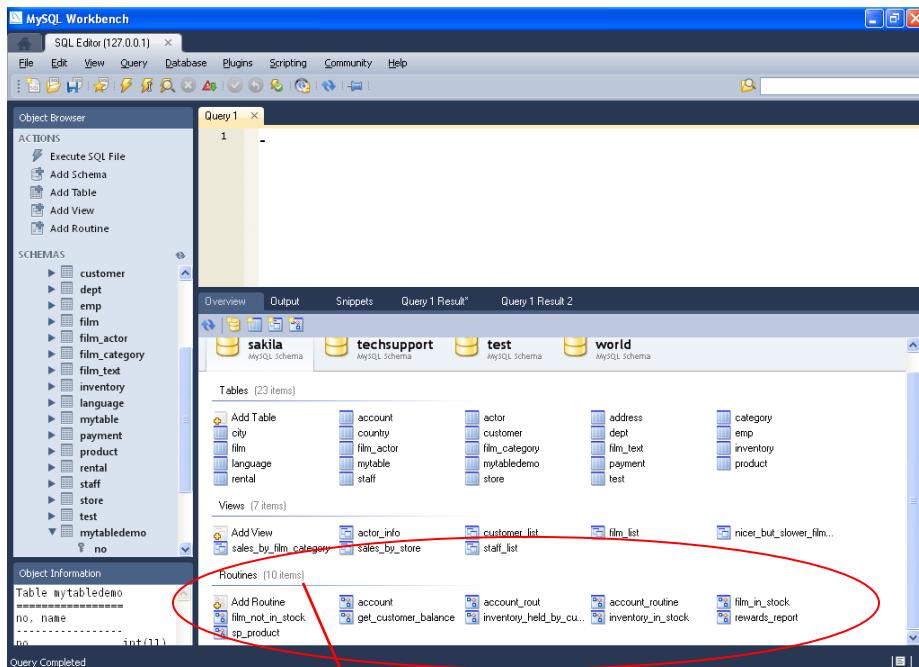
1 • INSERT INTO sakila.myTable
2   ('X_NO',
3    'X_name')
4   VALUES
5    (1, 'AName');
6

```

Step 4: Select all query statement and click on 'Query' menu and then click on 'Execute All'.

Step 5: Record will be created in the database table.

5. To create stored procedure (Routine).

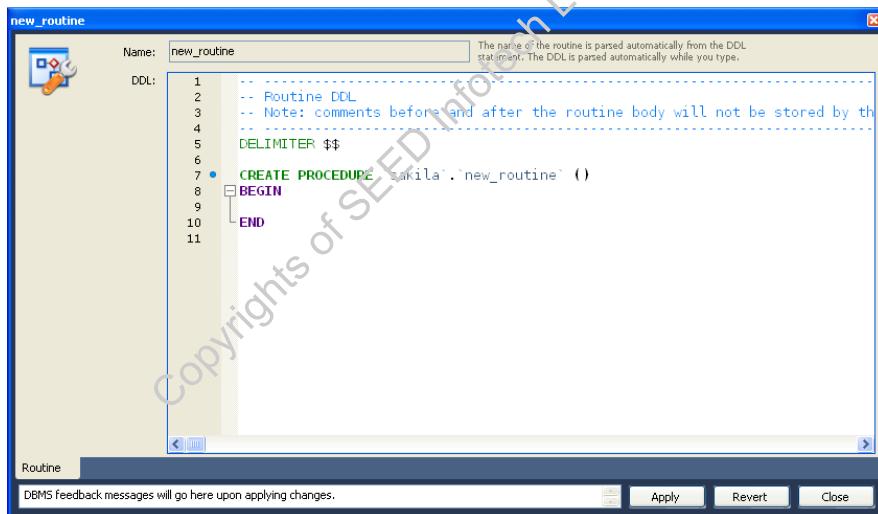




Step 1

Click on the 'Add Routine' to create a new table.

'new routine' window comes into view.



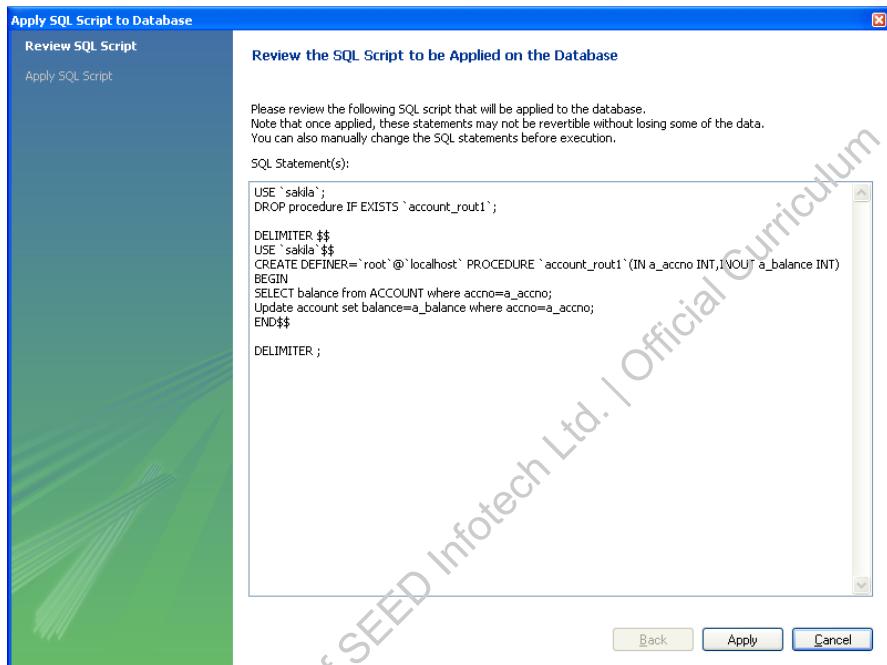
Enter information like:

Name : Not required. The name of routine is passed automatically from the DDL statement. The DDL is passed automatically while writing the routine.

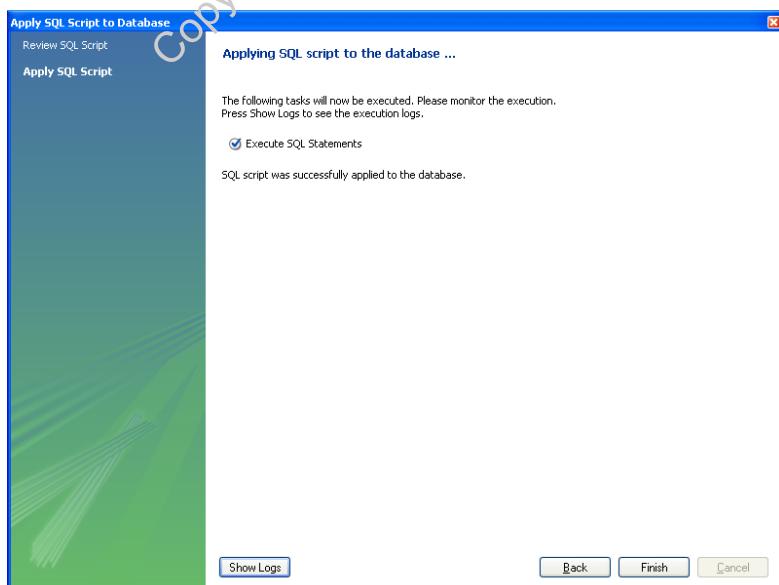
Once the routine is successfully written, click on 'Apply' button.

Step 2

Review the SQL script on the database, and click on 'Apply' button.



SQL script will be applied to the database. Click on ‘Finish’ button to complete the procedure.



Routine will be created in the database. Its snapshot is given below.

Query 1 x

```

1  DELIMITER $$;
2
3  • CREATE DEFINER='root'@'localhost' PROCEDURE `account_rout1` (IN a_acno INT, INOUT a_t
4  BEGIN
5    SELECT balance from ACCOUNT where accno=a_acno;
6    Update account set balance=a_balance where accno=a_acno;
7  END
8

```

Routines [11 items]

Add Routine	account	account_rout	account_rout1	account_routine
film_in_stock	film_not_in_stock	get_customer_balance	inventory_held_by_cu...	inventory_in_stock
rewards_report	sp_product			

6. MySQL Database information.

Step 1

The JDBC users require some information to connect with MySQL database such as:

- Driver name: com.mysql.jdbc.Driver
- URL: jdbc:mysql://localhost:3306: DBName
 - Host: localhost
 - Port: 3306
 - Username: root
 - Password: created during installation

Step 2

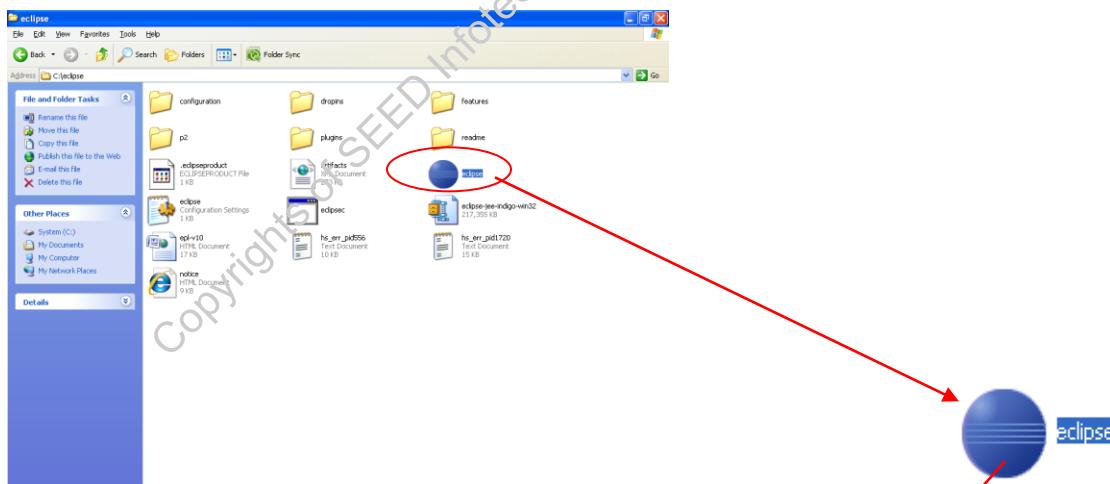
To execute JDBC application classpath needs to point to the mandatory jar file:
Mysql-connector-java-5.1.15-bin.jar

Appendix-D

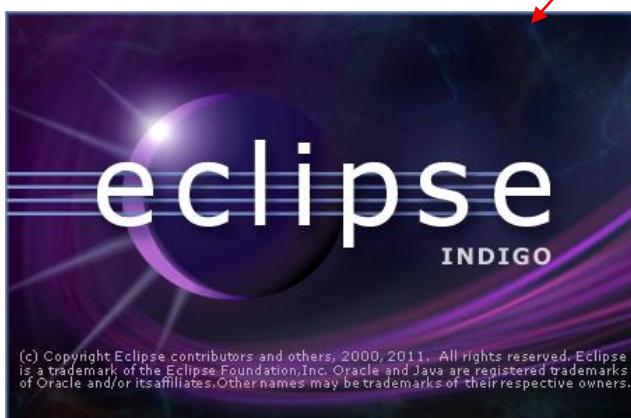
Eclipse is an Integrated Development Environment (IDE) for creating Java applications. It has facilities for creating, executing, debugging and deploying Java applications.

1. To open Eclipse.

Go to the eclipse folder and click on eclipse icon.



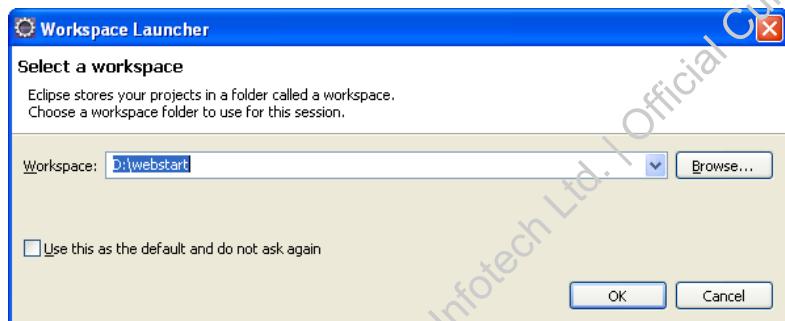
Once we click on eclipse icon, eclipse gets start.



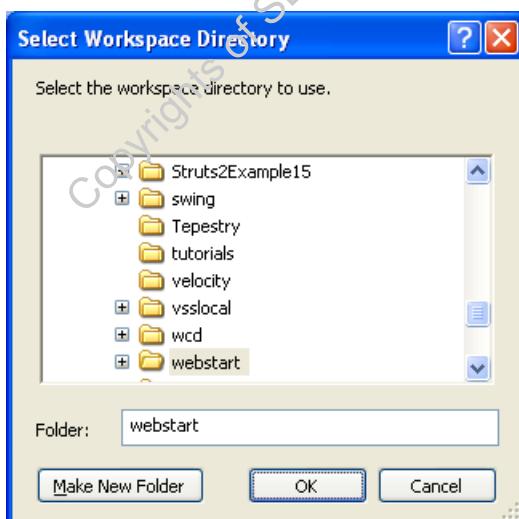
2. Create workspace

Eclipse starts and it asks for workspace. So, create Workspace on the local drive. Choose a workspace folder to use for this session.

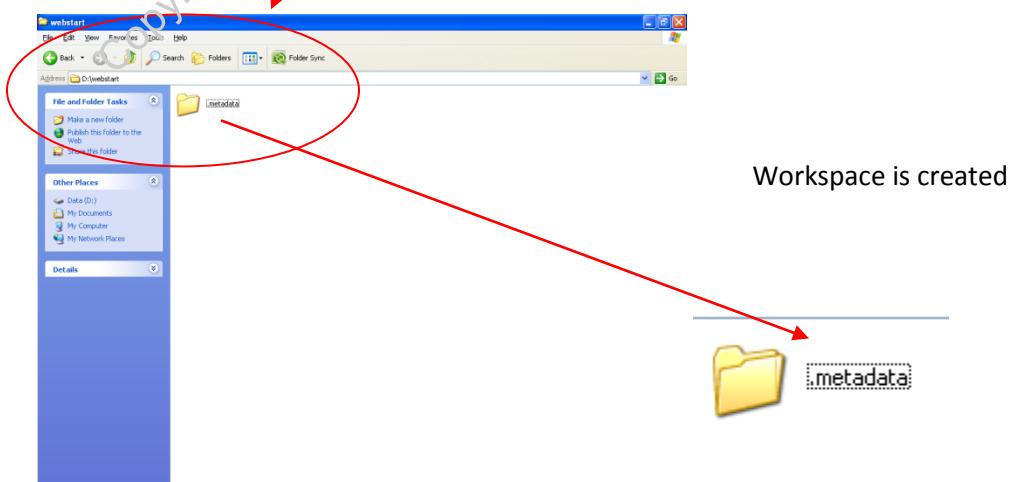
Workspace: Eclipse stores the projects in a folder called a workspace.

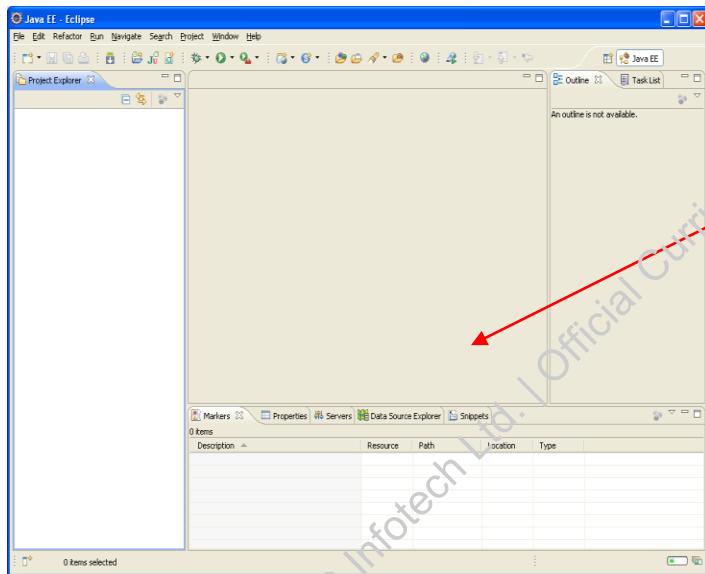


Click on 'Browse' button to select your workspace.



The 'select Workspace Directory' window appears on clicking the browse button. Select the folder if already created or create new folder and then select. To create new folder click on button 'Make New Folder' and finally click on 'OK' button. Workspace will be created in the selected directory with a given name and eclipse splash screen appears.

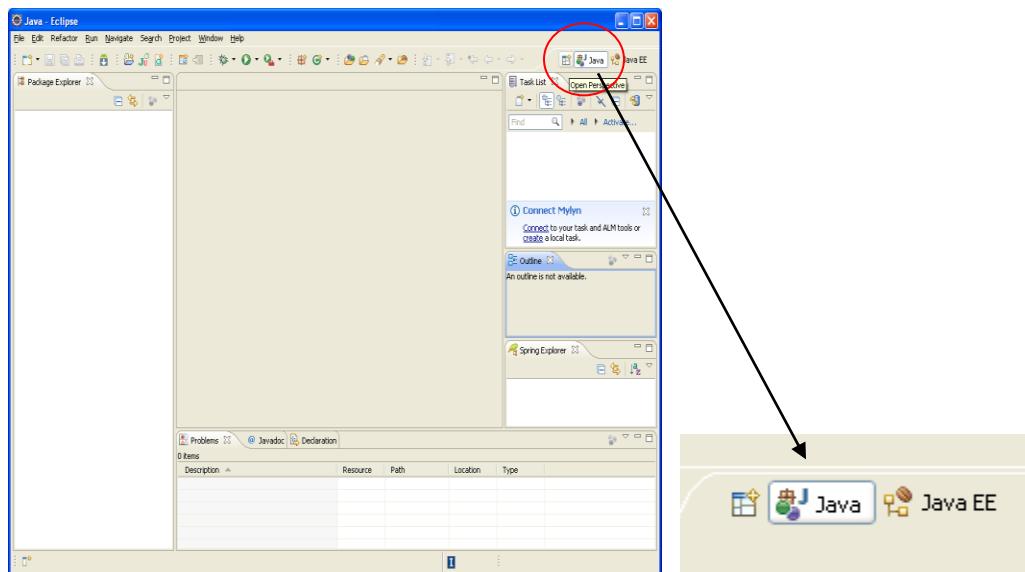




Eclipse Window

3. Select your perspective

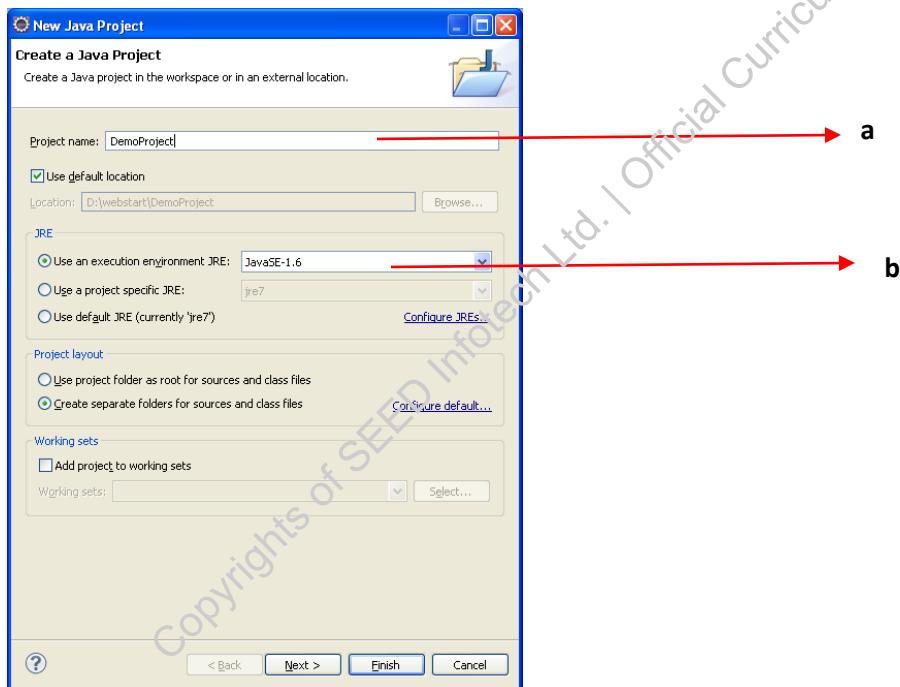
Eclipse is an IDE for the different types of applications. To execute a simple core Java application, tools and properties specific to core Java are required. To execute server-based application, server environment is required. So select the perspective according to the need of the application.



4. Create new Project.

Click on File menu:

File → New → Java Project



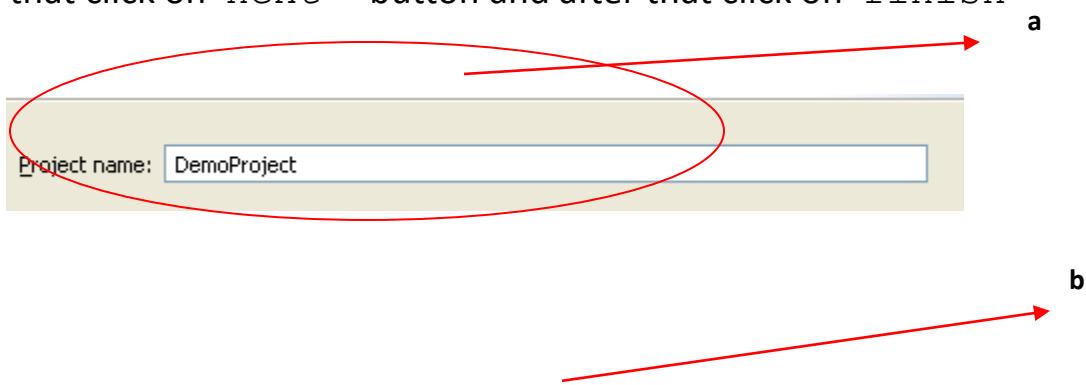
New project window appears, in which certain information has to be entered, such as:

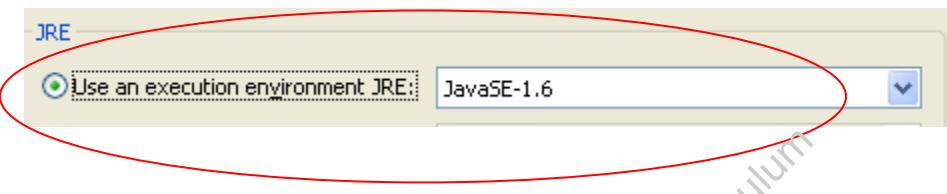
a. Project name: enter the project name e.g. MyProject

b. JRE information

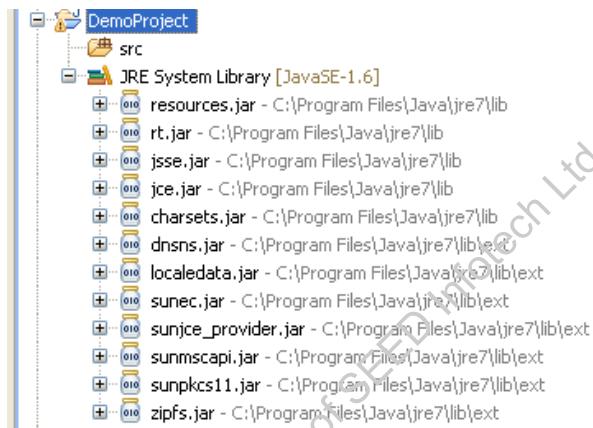
Use an execution environment JRE: select your specific JRE e.g. javaSE-1.6

After that click on 'next' button and after that click on 'finish' button





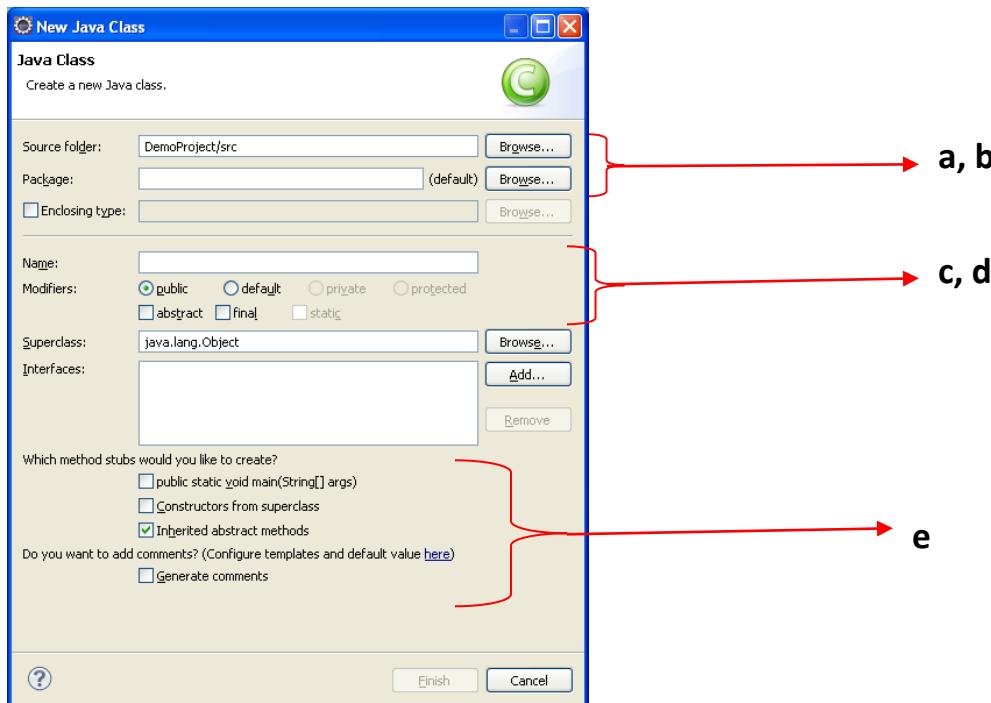
Java project will be created with required library files including the Java runtime.



5. Create Java File.

Select current project and click on File menu.

File → New → Class

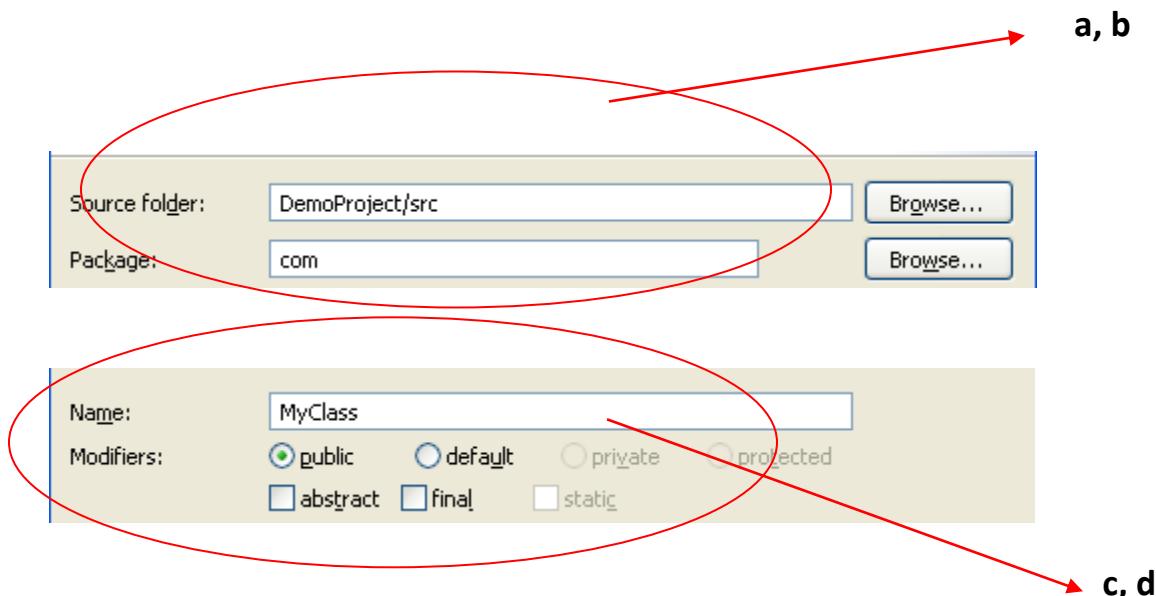


New Java class window appears. Enter the mandatory information, as directed, to create a new Java class.

- a. Source/folder : enter the folder name or source location. Default value is the 'src' folder in the project. e.g. DemoProject/src
- b. Package: Though optional, it is strongly recommended to create a new Java class within a package.
- c. Name : The mandatory name of the class.
- d. Modifiers: Can be public /default
- e. Method specific check boxes:
 - i. Public static void main(String args[])

Default main method will be auto-generated.
 - ii. Constructors from superclass

Recommended to turn on if current class extends from a pre-existing class.
 - iii. Inherited abstract methods

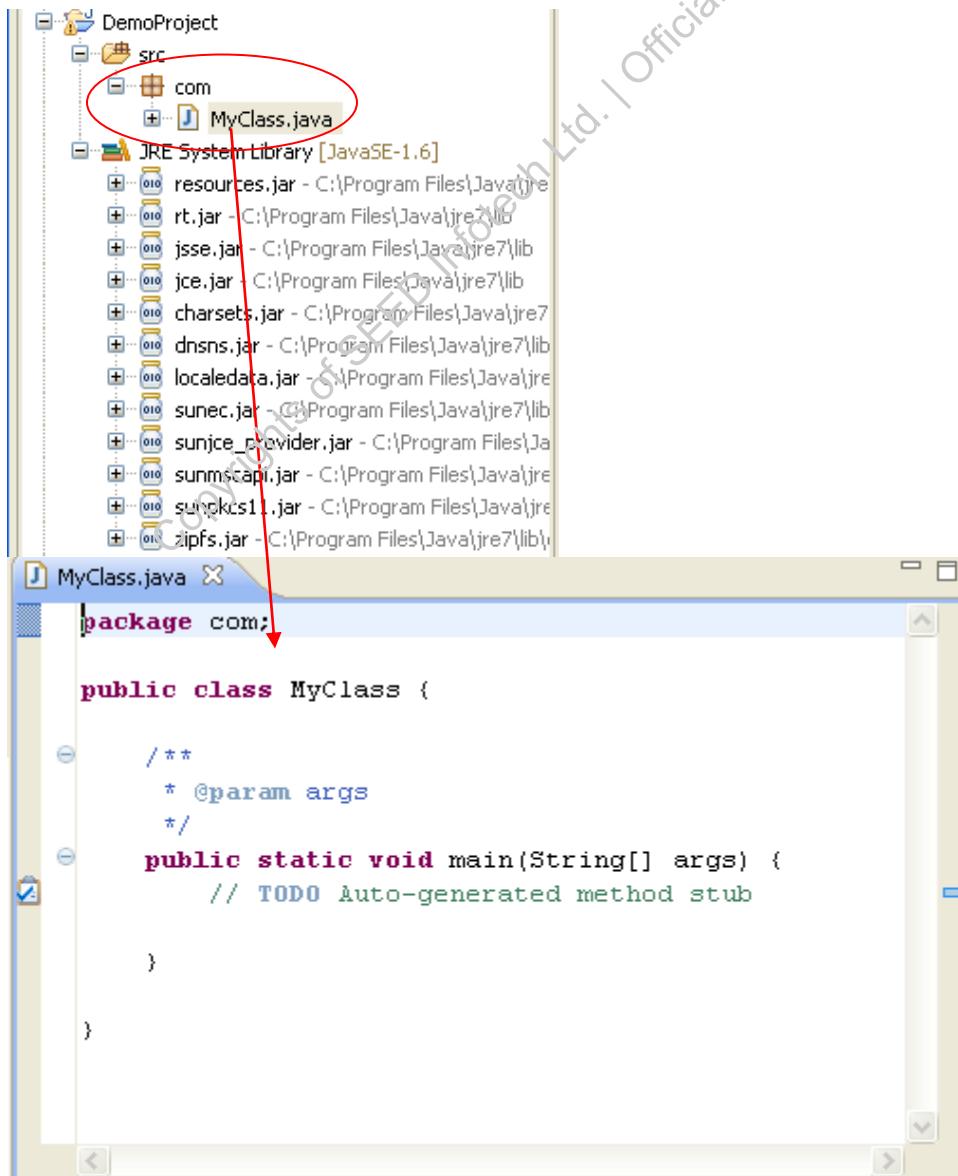


Which method stubs would you like to create?

- public static void main(String[] args)
- Constructors from superclass
- Inherited abstract methods

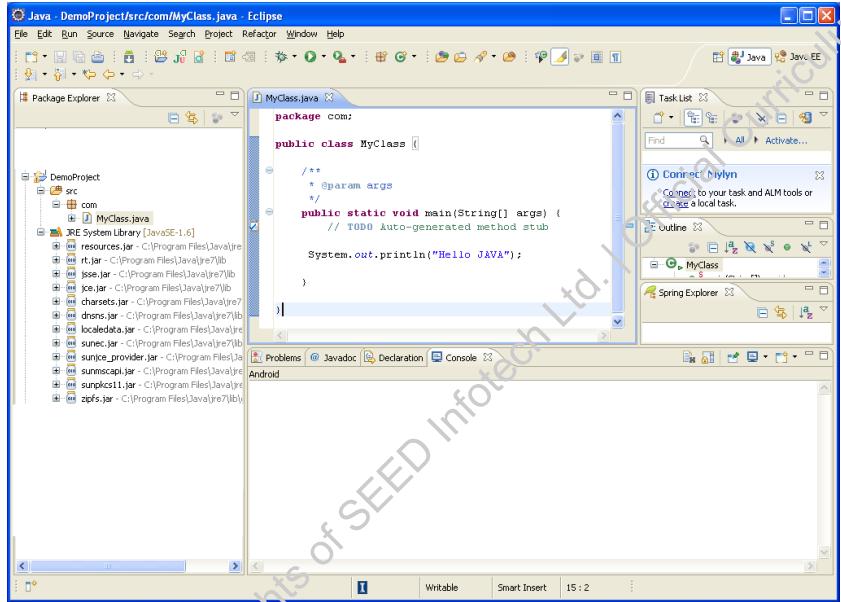
e

Click on the 'finish' button, when all the mandatory is filled in.



6. Execution of JAVA Application

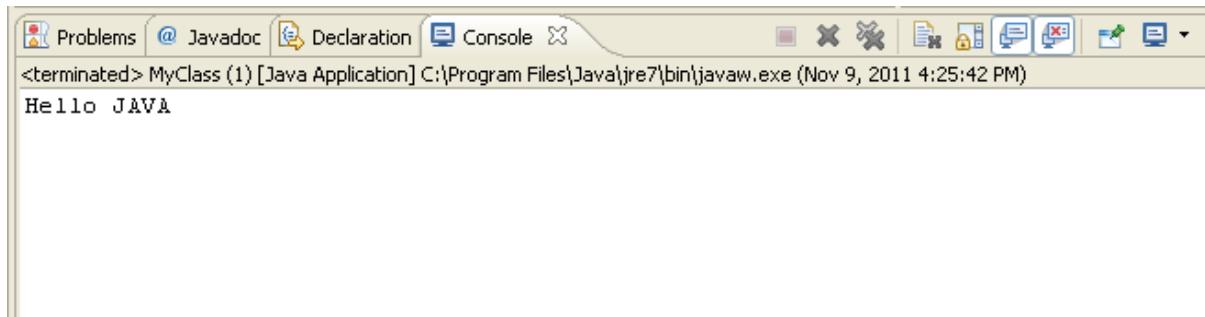
Consider a simple example – a program which will print ‘Hello JAVA’.



To execute or run the java program

Run → Run As → Java Application

Program is executed and will print output in a console window (View).



Copyrights of SEED Infotech Ltd. | Official Curriculum

