Programming in Java Dr. M. T. Wanjari

**Unit – II**

**Vectors:**

J2SE 5.0 version supports the concept of variable arguments to methods. This feature can also be achieved in Java through the use of the vector class contained in the java.util package. This class can be used to create a generic dynamic array known as vector that can hold objects of any type and any number. The objects do not have to be homogeneous. Arrays can be easily implemented as vectors. Vectors are created like arrays as follows:

Vector intVect = new Vector ( ); //declaring without size

Vector list = new Vector (3); //declaring with size

Note that a vector can be declared without specifying any size explicitly. A vector without size can accommodate an unknown number of items. Even, when a size is specified, this can be overlooked and a different number of items may be put into the vector. Remember, in contrast, an array must always have its size specified.

Vectors posses a number of advantages over arrays.

1. It is convenient to use vectors to store objects.

2. A vector can be used to store a list of objects that may vary in size.

3. We can add and delete objects from the list as and when required.

A major constraint in using vectors is that we cannot directly store simple data type in a vector, we can only store objects. Therefore, we need to convert simple types to objects. This can be done using the wrapper classes.

Example: Program to demonstrate working with vectors and arrays.

import java.util.\*;

class LanguageVector

{

public static void main(String args[])

{

Vector list = new Vector ( );

int length = args.length;

for (int i=0;i<length;i++)

{

list.addElement (args[i]);

M.Sc. Semester – I (Computer Science) 1

Programming in Java Dr. M. T. Wanjari

}

list.insertElementAt(“COBOL”, 2);

int size = list.size( );

String listArray[] = new String[size];

list.copyInto(listArray);

System.out.println(“List of Languages”);

for (int i=0;i<size;i++)

{

System.out.println(listArray[i]);

}

}

**Output of the Program**

List of Languages

Ada

BASIC

COBOL

C++

FORTRAN

Java

**Wrapper Classes:**

Vectors cannot handle primitive data types like int, float, long, char and double. Primitive data types may be converted into object types by using the wrapper classes contained in the java.lang package. Table 2.1 shows the simple data types and their corresponding wrapper class types.

**Table: 2.1 Warpper classes for Converting Simple Types**

|  |  |
| --- | --- |
| Simple Type | Wrapper Class |
| boolean | Boolean |
| char | Character |
| double | Double |
| float | Float |
| int | Integer |

M.Sc. Semester – I (Computer Science) 2

Programming in Java Dr. M. T. Wanjari

|  |  |
| --- | --- |
| long | Long |

The wrapper classes have a number of unique methods for handling primitive data types and objects.

**Example: A program to demonstrate Use of wrapper class methods**

import java.io.\*;

class Invest

{

public static void main(String args[])

{

Float principalAmount = new Float ( 0); //converting number to object Float interestRate = new Float (0);

int numYears = 0;

try

{

DataInputStream in = new DataInputStream(System.in);

System.out.println(“Enter Principal Amount : ”);

System.out.flush();

String principalString = in.readLine();

principalAmount = Float.valueOf(principalString);

//String object to number object

System.out.print(“Enter Interest Rate : ”);

System.out.flush();

String interestString = in.readLine();

interestRate = Float.valueOf(interestString);

System.out.print(“Enter Number of Years : ”);

Syste.out.flush();

String yearsString = in.readLine();

numYears = Interger.parseInt(yearsString); //Numeric strings to numbers }

catch(IOException e)

M.Sc. Semester – I (Computer Science) 3

Programming in Java Dr. M. T. Wanjari

{

Sysetm.out.println(“I/O Error”);

Syste.exit(1);

}

float value = loan(principalAmount.floatValue(), interestRate.floatValue().numYears); printline();

System.out.println(“Final Value = ” + value);

Printline();

}

//method to compute Final value

static float loan (float p, float r, int n)

{

int year = 1;

float sum = p;

While (year <= n)

{

sum = sum \* (l+r);

year = year +1;

}

return sum;

}

//Method to draw a line

static void printline()

{

for(int i=1; i<=30; i++)

{

System.out.print(“=”);

}

System.out.println(“ “);

}

M.Sc. Semester – I (Computer Science) 4

Programming in Java Dr. M. T. Wanjari

}

**The output of the program is**

Enter principal Amount : 5000

Enter Interest Rate : 0.15

Enter Number of Years : 4

Final Value : 8745.03

**Autoboxing and Unboxing:**

The autoboxing and unboxing feature, introduced in J2SE 5.0, facilitates the process of handling primitive data types in collections. We can use this feature to convert primitive data types to wrapper class types automatically. The compiler generates a code implicitly to convert primitive type to the corresponding wrapper class type and vice-versa. For example, consider the following statements:

Double d\_object = 98.42;

double d\_primitive = d\_object.doubleValue( );

Using the autoboxing and unboxing feature, we can rewrite the above code as: Double d\_object = 98.42;

double d\_primitive = d\_object;

How, the Java compiler provides restrictions to perform the following conversions: • Convert from null type to any primitive type.

• Convert to the null type other than the identify conversion.

• Convert from any class type C to any type if C is not object.

**Applet:**

An applet is a small program typically embedded within the web page which is used to create a dynamic and interactive application. They provide interactive features to a web page which cannot be provided by HTML only. For example, applet enable capturing user inputs in the form of mouse clicks, text entry, checkbox selection, etc. and generating response to the user’s actions.

Each applet that is created must be a subclass of the Applet class, contained within the java.applet package. This class contains methods which govern the life and behavior of the applets. In addition, applets use various methods of the Graphics class contained inside the java.awt package. The Graphics class is responsible for all the operations related to display

M.Sc. Semester – I (Computer Science) 5

Programming in Java Dr. M. T. Wanjari

(output) of an applet. The Java applets can be executed either through an appletviewer (a tool of Java Development Kit) or any Java compatible web browser.

**Life Cycle of an Applet:**

A Java applet enters into various states during its entire life cycle which include born state, running state, idle state and dead state. These states occur when different methods of the Applet class are invoked by the Java runtime system. The invocation of these methods makes an applet undergo a series of state change right from the time it is loaded till it destroyed and frees all the resources held by it. The life cycle of an applet can be depicted as shown in Fig. 2.1.

**Applet Begins**

**init ()**

**Born**

**state ()**

**paint ()**

**Running**

**stop () start ()**

**Idle**

**destroy ()**

**Dead**

**Applet Ends**

**Fig. 2.1 Applet Life Cycle**

The order of invocation when an applet is loaded is:

M.Sc. Semester – I (Computer Science) 6

Programming in Java Dr. M. T. Wanjari

**1. init ():** The life cycle of an applet begins when it is first loaded and the init () method is invoked. This method is invoked only once during the entire lifetime of an applet. The body of the method includes statements related to variable initialization, object creation, adding components like buttons, textboxes etc., setting colors of the applet, loading of images or fonts, etc. After the invocation of the init () method, the applet enters the born state.

**2. state ():** This method is automatically invoked after the init () method. Unlike init () method, start () method may be invoked more than once. This method is called every time the web page containing the applet is executed and displayed on the screen. With the invocation of this method, the applet enters the running state.

**3. panit ():** This method is invoked to display the output on the screen in the form of text, graphics, etc. Sometimes it may happen that the window in which applet is running gets covered by another window, or gets minimized or resized. In all these cases, paint () method is re invoked for the output to be redrawn on the screen. As depicted in the Fig. 2.1 the applet remains in the running state while paint ( ) method in invoked.

The order of method invocation at the time of termination of an applet is: **1. stop ():** This method is invoked automatically when the web page containing the running applet is closed or left temporarily and applet enters the idle state. The user can also stop the running applet by invoking the stop () method explicitly.

**2. destroy ():** This method is invoked to remove the applet permanently from the memory. It releases all the resources held by the applet. Like init () method, this method in invoked only once during the entire life cycle of an applet. The applet becomes dead when this method is invoked.

**Note:** The init () and destroy () method are called only once whereas start (), paint () and stop () methods can be called multiple times in an applet.

**Steps to Build an Applet:**

To built an applet, first of all we need to create an applet code (.java file). After successful compilation, its corresponding .class file is created. Then a web page also known as HTML document is to be developed. Once the HTML document is created, the applet is to be inserted in it. Finally, the resultant HTML document will be executed to produce the desired output. The detailed description of these steps is given in this section.

**Creating an Applet Code**

M.Sc. Semester – I (Computer Science) 7

Programming in Java Dr. M. T. Wanjari

The first step in building an applet is to write an applet code. The creation of an applet code is same as creation of a Java source file.

**The general form of an applet code is**

import java.applet.\*;

import java.awt.\*;

public class AppletCode extends Applet

{

public void init ()

{

:

}

public void paint (Graphics gra)

{

:

}

}

Here, the program begins with the two import statements. The first statement imports java.applet package which contains the Applet class and the second statement imports java.awt package. The next statement declares the class AppletCode. Note that it is declared public as it will be accessed by the code outside the program. The AppletCode class contains the paint () method which takes the object of Graphics class as a parameter. This object describes the graphical environment in which the applet will be running.

To show the complete procedure of the creation of an applet application let us start with writing an applet code.

**Example: An example of an applet code**

import java.applet.\*;

import java.awt.\*;

public class AppletCode extends Applet

{

String fs, ss;

public void init ()

M.Sc. Semester – I (Computer Science) 8

Programming in Java Dr. M. T. Wanjari

{

fs=”Hello!!! ”;

ss=”This is my first applet”;

}

public void paint (Graphics gra)

{

Gra.drwString (fs+ss, 20, 70);

}

}

In this example, the drawstring () method invoked inside the paint () method will display the string “Hello!!! This is my first applet” at the location x=20, y=70 relative to the screen. It should be noted that since the paint () method has no definition of its own it must be overridden. **Creating an Executable Applet**

To create an executable applet, compile the applet code. An applet code is compiled using the command javac AppletCode.java on the command prompt. This command will generate the .class file of the applet code known as executable applet.

**Designing an HTML Document**

As stated earlier, an applet is not an an independent entity as it cannot run on its own and it needs to be inserted into the HTML document (web page). Therefore, the next step is to design an HTML document which will contain the applet code. An HTML document is a collection of text and HTML tags interpreted and executed by the web browser or an appletviewer. Every HTML document can be broadly divided into four major elements, they are HTML, HEAD, TITLE and BODY.

• **HTML element** marks the beginning and ending of the HTML document. This element indicates to the web browser that the page loaded is an HTML document. The syntax of HTML element is

<HTML> … </HTML>

• **HEAD element** specifies the header information about the HTML document such as title, keywords, etc. The content inside the HEAD tag is not displayed in the web page. The syntax of HEAD element is

<HEAD> … </HEAD>

M.Sc. Semester – I (Computer Science) 9

Programming in Java Dr. M. T. Wanjari

• **TITLE element** is defined within the HEAD tag of the HTML document. The title of the web page is displayed in the title bar of the web browser.

The syntax of TITLE element is

< TITLE> … </ TITLE >

• **BODY element** contains the content of the web page such as text, images, etc. It determines the appearance of the web page on the screen. The BODY tag is placed exactly after the HEAD tag.

The syntax of BODY element is

< BODY> … </BODY >

**Example: An example of an HTML document**

<HTML>

<HEAD>

<TITLE>

Welcome to the Java Applet </TITLE>

</HEAD>

<BODY>

<CENTER>

<H1> Welcome </H1>

<H4> This is a structure of an HTML document. </H4>

</CENTER>

</BODY>

</HTML>

**The output of the program is**

Welcome

This is a structure of an HTML document

**Note:** An HTML document is prepared using notepad or any text editor and saved with .html extension.

**An Applet Tag:**

The <APPLET> tag specifies the applet code that is to be loaded and at the same time tells the browser about its space requirement on the web page.

This simplest form of the <APPLET> tag is

M.Sc. Semester – I (Computer Science) 10

Programming in Java Dr. M. T. Wanjari

<APPLET

CODE=AppletCode.class

WIDTH=600

HEIGHT=250>

</APPLET>

This code tells the browser or the appletviewer to reserve the display area having width of 600 pixels and height of 250 pixels for the applet and load the compiled Java applet named AppletCode.class. The three attributes, namely, CODE, WIDTH and HEIGHT must be specified while defining an applet tag. Some other attributes of the <APPLET> tag are listed in Table 2.2.

**Table: 2.2 Attributes of an Applet Tag**

|  |  |
| --- | --- |
| **Attribute (optional)** | **Description** |
| NAME=applet\_name | It specifies the name of the applet. In inter-applet communication, applets refer to the other applets using their names. |
| CODEBASE=codebase\_URL | It indicates the URL of the directory where the applet resides. This can be omitted in case the applet and HTML document are in the same directory. |
| ALT=alternate\_text | The text written in this attribute will be displayed if the browser is unable to run the applet. |
| ALIGN=alignment | It determines the alignment of the applet on the web page. The alignment can be LEFT, RIGHT, TOP, MIDDLE, BOTTOM, BASELINE, ABSMIDDLE, TEXTTOP, ABSBOTTOM. |
| VSPACE=pixels | It specifies the amount of vertical blank space that should be left around the applet by the browser. |
| HSPACE=pixels | It specifies the amount of horizontal blank space that should be left around the applet by the browser. |

**Adding an Applet to HTML**

An applet is added to the HTML document by placing <APPLET> tag inside the <BODY> tag of the HTML document. Here, we have shown the entire HTML document with an embedded <APPLET> tag of AppletCode.java applet.

M.Sc. Semester – I (Computer Science) 11

Programming in Java Dr. M. T. Wanjari

<HTML>

<HEAD>

</HEAD>

<BODY>

<CENTER>

<APPLET

CODE=AppletCode.class

WIDTH=600

HEIGHT=250>

</APPET>

</CENTER>

</BODY>

</HTML>

**Note:** The .class file and its corresponding .html file must be present in the same directory. **Running the Applet**

As stated earlier, an applet can be executed either through any Java compatible web browser or an appletviewer. An appletviewer is a tool of Java Development Kit which is specifically designed for viewing applets. Using the web browser enables the user to see the entire HTML document containing the applet. Conversely, if we run the applet using the appletviewer we see the output of an applet only as it ignors the HTML tags.

The command to run an applet using appletviewer is

Appletviewer AppletCode.html

Output:

Hello!!!This is my first applet

**Passing Parameters to Applets**

Java allows users to pass user-defined parameter to an applet with the help of <PARAM> tags. The <PARAM> tag has a NAME attribute which defines the name of the parameter and a VALUE attribute which specifies the value of the parameter. In this applet source code, the applet can refer to the parameter by its NAME to find its value.

The syntax of the <PARAM> tag is

<APPLET>

M.Sc. Semester – I (Computer Science) 12

Programming in Java Dr. M. T. Wanjari

<PARAM NAME=parameter1\_name VALUE=parameter1\_value>

<PARAM NAME=parameter2\_name VALUE=parameter2\_value>

.

.

.

<PARAM NAME=parametern\_name VALUE=parametern\_value>

</APPLET>

For example, consider the following statements to set the text attribute of applet to This is an example of Parameter ! ! !

<APPLET>

<PARAM NAME=text VALUE= This is an example of Parameter ! ! !>

</APPLET>

Note that the <PARAM> tags must be included between the <APPLET> and </APPLET> tags. The init () method in the applet retrieves user-defined values of the parameters defines in the <PARAM> tags by using the getParameter () method. This method accepts one string argument that holds the name of the parameter and returns a string which contains the value of that parameter. Since it returns String object, any data type other than String must be converted into its corresponding data type before it can be used.

**Example: An applet code to demonstrate passing of parameters to an applet.** import java.awt.\*;

import java.applet.\*;

public class ParameterExample extends Applet

{

String pstr;

int i, j, sum;

public void init()

{

pstr=getParameter(“msg”);

i=Integer.parseInt(getParameter(“firstnumber”));

j=Integer.parseInt(getParameter(“secondnumber”));

sum=i+j;

M.Sc. Semester – I (Computer Science) 13

Programming in Java Dr. M. T. Wanjari

}

public void paint(Graphics gra)

{

gra.drawString(“Operation is: ”+pstr. 20, 20);

gra.drawString(“First Number is: ”+i. 20, 40);

gra.drawString(“Second Number is: ”+j. 20, 60); gra.drawString(“Total: ”+sum. 20, 80);

}

}

**The HTML code for ParameterExample is**

<HTML>

<HEAD>

</HEAD>

<BODY>

<CENTER>

<APPLET

CODE=”ParameterExample.class”

WIDTH=600

HEIGHT=250>

<PARAM NAME=msg VALUE=Summation> <PARAM NAME=firstnumber VALUE=14>

<PARAM NAME=secondnumber VALUE=2> </APPLET>

</CENTER>

</BODY>

</HTML>

**The output of the HTML code is:**

Operation is: Summation

firstnumber is: 14

secondnumber is: 2

Total: 16

M.Sc. Semester – I (Computer Science) 14

Programming in Java Dr. M. T. Wanjari

**Drawing Images on the Applet**

The Image class is used to load and display images. To load an image the getImage() method of the Image class is used and to display the image the drawImage () method of the Graphics class is used.

The general form of the getImage () method is

Image getImage (URL pathname, String filename)

Image getImage (URL pathname)

where,

pathname is the address of the image file on web. When the image file and the source file are in the same directory, getCodeBase () method is used as first parameter to the method. filename is the name of the image file

The general form of the drawImage () method is

boolen drawImage (Image image, int startx, int starty, int width, int height, ImageObserver img\_obj)

where,

image is the image to be loaded in the applet

startx is the pixels space from the left corner of the screen

starty is the pixels space from upper corner of the screen

width is the width of the image

height is the height of the image

img\_obj is object of the class that implements ImageObserver interface

**Example: An applet code to demonstrate the use of Image class.**

import java.applet.\*;

import java.awt.\*;

public class ImageExample extends Applet

{

private Image pic;

public void init ()

{

pic =getImage (getCodeBase (), “java\_pic.gif”);

}

M.Sc. Semester – I (Computer Science) 15

Programming in Java Dr. M. T. Wanjari

public void paint (Graphics gra)

{

gra.drawImage (pic, 150, 50, 120, 200, this);

}

}

**The HTML code for ImageExample is**

<HTML>

<HEAD>

</HEAD>

<BODY>

<CENTER>

<APPLET

CODE=”ImageExample.class”

WIDTH=”600”

HEIGHT=”250”>

</APPLET>

</CENTER>

</BODY>

</HTML>

**AWT (Abstract Window Toolkit)**

➢ The AWT (Abstract Window Toolkit) is java’s original platform independent windowing graphics and user interface toolkit.

➢ AWT is now part of the Java Foundation Class (JFC) the standard API for providing a graphical user interface (GUI) for a java program.

➢ JFC is short for Java foundation classes, which encompass a group of features for building graphical user interfaces (GUI) and adding rich graphics functionality and interactivity to java applications.

➢ Features: i) swing GUI Components

ii) Pluggable look and feel support etc

➢ GUI is the interaction between user and machine

M.Sc. Semester – I (Computer Science) 16

Programming in Java Dr. M. T. Wanjari

➢ The AWT contains numerous classes and methods that allow to create and manage windows, manage fonts, output text and utilize graphics.

➢ The main purpose of the AWT is to support applet windows, it can also be used to create standalone windows that run in a GUI environment such as windows.

➢ The AWT classes are contained in java.awt package. And it is one of java’s largest package. This package is logically organized in a top-down, hierarchical fashion & so that it is easy to understand.

➢ This package gives various tools to design GUI.

GUI

AWT SWING

(java.awt.\*) (java.swing.\*)

heavy weight component light weight component

depends on OS not depends on OS efficient than awt

gives tree structure

➢ AWT has various uses

i) It provides GUI components

ii) Allows to extend feature in user application

iii) Ensures that every components displayed on the screen is a subclass of the abstract class component

iv) AWT has container which is an abstract subclass of component and includes various subclasses like Window, Panel

M.Sc. Semester – I (Computer Science) 17

Programming in Java Dr. M. T. Wanjari

awt package (List of classes)

java.awt.\*;

(Layouts)

- BorderLayout

- CardLayout Layout managers to arrange controls in window

- FlowLayout

- GridBagLayout

- GridLayout

- Button

- checkboxGroup // Radiobuttons List - Color Scrollbar - Dimension Text Component - Dialog

- Component

- Container TextArea TextField - FontMatrices

- Image

- Insets

- Point Container

- Polygon

- Rectangle

- Frame Panel Window ScrollPane - Font

- Event // Notification of selecting something Applet Frame

- Graphics Dialog

- Menu

filedialog

Menu Component

Menu bar MenuItem

Menu Checkbox

(pop-up menu) MenuItem

➢ AWT defines windows according to a class hierarchy that adds functionality and specificity with each level.

M.Sc. Semester – I (Computer Science) 18

Programming in Java Dr. M. T. Wanjari

**Component (Controls)**

➢ Component class is at the top of the AWT hierarchy. It is the abstract class and that encapsulates all of the attributes of a visual component. All user interface elements that are displayed on the screen and that interact with the user are subclasses of Component.

➢ There are hundred public methods that are responsible for managing events, such as mouse and keyboard input, positioning and sizing the window, and repainting. ➢ A Component object is responsible for remembering the current foreground and background colors and the currently selected text font.

**Container (having layouts)**

➢ The Container class is a subclass of Component class.

➢ It has additional methods that allow other Component objects to be nested within it. & other Container objects can be stored inside of a Container. This makes for a multileveled containment system.

AWT

Component

Container

Window Panel

Frame Applet

Dialog

Filedialog

**Fig: 2.2 class hierarchy of AWT**

**Panel**

➢ The Panel class is a concrete subclass of Container. It doesn’t add any new methods, it simply implements Container.

➢ Panel is the superclass for Applet. When screen output is directed to an applet, it is drawn on the surface of a Panel object. A Panel is a window that does not contain a title bar, menu bar, or border. (That’s why you don’t see these items when an applet is run inside a browser) When run an applet using an applet viewer, the applet viewer provides the title and border.

M.Sc. Semester – I (Computer Science) 19

Programming in Java Dr. M. T. Wanjari

➢ Other components can be added to a Panel by its add( ) method inherited from Container. Once these components have been added, position and resize them manually using the setLocation( ), setSize( ), or setBounds( ) methods defined by Component. **Window**

➢ The Window class creates a top-level window. A *top-level window* is not contained within any other object.

➢ We cannot create Window objects directly, to create window by using a subclass of window called Frame.

**Frame**

➢ It is a subclass of Window and has a title bar, menu bar, borders, and resizing corners. ➢ If you create a Frame object from within an applet, it will contain a warning message, such as “Java Applet Window” to the user that an applet window has been created. This message warns users that the window they see was started by an applet and not by software running on their computer.

➢ When a Frame window is created by a program rather than an applet, a normal window is created.

**Canvas**

➢ It is not part of the hierarchy for applet or frame windows; there is one other type of window which is valuable: Canvas.

➢ This Canvas encapsulates a blank window upon which we can draw.

**Working with Frame Windows**

➢ After the applet, the type of window create is derived from Frame.

➢ Use to create child windows within applets, and top-level or child windows for applications.

➢ It creates a standard-style window.

Here are two of Frame’s constructors:

Frame( )

Frame(String *title*)

➢ The first form creates a standard window that does not contain a title.

➢ The second form creates a window with the title specified by *title.*

M.Sc. Semester – I (Computer Science) 20

Programming in Java Dr. M. T. Wanjari

➢ We cannot specify the dimensions of the window.

➢ We must set the size of the window after it has been created.

➢ There are several methods will use when working with Frame windows. **Setting the Window’s Dimensions**

➢ The **setSize( )** method is used to set the dimensions of the window.

void setSize(int *newWidth*, int *newHeight*)

void setSize(Dimension *newSize*)

➢ The new size of the window is specified by *newWidth* and *newHeight*, or by the width and height fields of the Dimension object passed in *newSize*.

➢ The dimensions are specified in terms of pixels.

➢ The **getSize( )** method is used to obtain the current size of a window.

Dimension getSize( )

➢ This method returns the current size of the window contained within the width and height fields of a Dimension object.

**Hiding and Showing Window**

➢ After a frame window has been created, it will not be visible until we call setVisible( ). void setVisible(boolean *visibleFlag*)

➢ The component is visible if the argument to this method is **true**. Otherwise, it is hidden. **Setting a Windows Title**

➢ We can change the title in a frame window using setTitle( ).

void setTitle(String *newTitle*)

➢ Here, *newTitle* is the new title for the window.

**Closing a Frame Window**

➢ When using a frame window, program must remove that window from the screen when it is closed, by calling setVisible(false).

➢ To intercept a window-close event, must implement the windowClosing( ) method of the WindowListener interface.

➢ Inside windowClosing( ), must remove the window from the screen.

M.Sc. Semester – I (Computer Science) 21

Programming in Java Dr. M. T. Wanjari

**Layout Managers (Components)**

➢ A *layout manager* automatically positions components within a container. Thus, the appearance of a window is determined by a combination of the controls that it contains and the layout manager used to position them.

➢ Each Container object has a layout manager associated with it.

➢ A layout manager is an instance of any class that implements the LayoutManager interface.

➢ The layout manager is set by the setLayout( ) method. And if no call to setLayout( ) then the default layoutis assigned.

➢ General form

void setLayout(LayoutManager *layoutObj*)

Where, *layoutObj* is a reference to the desired layout manager.

➢ If the programmer wish to disable layout manager and position components manually, pass null for *layoutObj.* And After that setting the size of window manually by the method seBounds() defined by Component.

➢ Each layout manager keeps track of a list of components that are stored by their names. The layout manager is notified each time when we add a component to a container. ➢ Whenever the container needs to be resized, the layout manager is managed via minimumLayoutSize( ) and preferredLayoutSize( ) methods.

➢ Each component that is being managed by a layout manager contains the getPreferredSize( ) and getMinimumSize( ) methods.

➢ These return the preferred and minimum size required to display each component. ➢ Java has several predefined LayoutManager classes.

**FlowLayout**

➢ FlowLayout is the default layout manager for Panel and Applet.

➢ FlowLayout implements a simple layout style, which is similar to how words flow in a text editor.

➢ Components are laid out from the upper-left corner, left to right and top to bottom. ➢ When no more components fit on a line, the next one appears on the next line. ➢ There are three constructors for FlowLayout:

i) FlowLayout( ) // default constructor

M.Sc. Semester – I (Computer Science) 22

Programming in Java Dr. M. T. Wanjari

This creates the default layout, which placed components in the center and five pixel of space between each component.

ii) FlowLayout(int *how*)

This constructor forms and specify how each line is aligned. Valid values of *how* are as follows:

FlowLayout.LEFT

FlowLayout.CENTER

FlowLayout.RIGHT

These values specify left, center, and right alignment.

iii) FlowLayout(int *how*, int *horz\_gap*, int *vert\_gap*)

The constructor allows to specify the horizontal and vertical space provide between components in horz\_gap & ver\_gap respectively.

**Example: A program to demonstrate Flowlayout Demo.**

import java.awt.\*;

import java.awt.event.\*;

import java.applet.\*;

public class FlowLayoutDemo extends Applet implements ItemListener

{

String msg = "";

Checkbox Mouse, keyb, joystc, mon;

public void init()

{

// set left-aligned flow layout

setLayout(new FlowLayout(FlowLayout.LEFT));

mouse = new Checkbox("Mouse", null, true);

keyb = new Checkbox("Keyboard");

joystc = new Checkbox("Joystick");

mon = new Checkbox("Monitor");

add(mouse);

add(keyb);

add(joystc);

M.Sc. Semester – I (Computer Science) 23

Programming in Java Dr. M. T. Wanjari

add(mon);

// register to receive item events

mouse.addItemListener(this);

keyb.addItemListener(this);

joystc.addItemListener(this);

mon.addItemListener(this);

}

// Repaint when status of a check box changes.

public void itemStateChanged(ItemEvent ie)

{

repaint();

}

// Display current state of the check boxes.

public void paint(Graphics g)

{

msg = "Current state: ";

g.drawString(msg, 6, 80);

msg = " Mouse: " + Mouse.getState();

g.drawString(msg, 6, 100);

msg = " Keyboard: " + keyb.getState();

g.drawString(msg, 6, 120);

msg = " Joystick: " + joystc.getState();

g.drawString(msg, 6, 140);

msg = " Monitor: " + mon.getState();

g.drawString(msg, 6, 160);

}

}

**BorderLayout**

➢ The BorderLayout class implements a common layout style for top-level windows. ➢ This is default layout for Frame and Dialog.

➢ It has four narrow, fixed-width components at the edges and one large area in the center.

M.Sc. Semester – I (Computer Science) 24

Programming in Java Dr. M. T. Wanjari

➢ The four sides are referred to as north, south, east, and west and the middle area is called the center.

➢ There are two constructors defined by **BorderLayout**:

BorderLayout( )

This constructor creates a default border layout.

i) BorderLayout(int *horz*, int *vert*)

This allows to specify the horizontal and vertical space left between components. ➢ BorderLayout defines the following constants that specify the regions: BorderLayout.CENTER, BorderLayout.SOUTH

BorderLayout.EAST, BorderLayout.WEST & BorderLayout.NORTH

➢ When adding components, these constants with the following form of add( ), which is defined by Container:

void add(Component *compObj,* Object *region*);

eg. f1.add(b1. BorderLayout.NORTH);

Here, *compObj* is the component to be added, and *region* specifies where the component will be added.

Here is an example of a **BorderLayout** with a component in each layout area: // Demonstrate BorderLayout.

import java.awt.\*;

import java.applet.\*;

import java.util.\*;

public class BorderLayoutDemo extends Applet

{

public void init()

{

setLayout(new BorderLayout());

add(new Button("This is across the top."),

BorderLayout.NORTH);

add(new Label("The footer message might go here."),

BorderLayout.SOUTH);

add(new Button("Right"), BorderLayout.EAST);

M.Sc. Semester – I (Computer Science) 25

Programming in Java Dr. M. T. Wanjari

add(new Button("Left"), BorderLayout.WEST);

String msg = "The reasonable man adapts " + "himself to the world;\n" + "the unreasonable one persists in " + "trying to adapt the world to himself.\n" + "Therefore all progress depends " + "on the unreasonable man.\n\n" +

" - George Bernard Shaw\n\n";

add(new TextArea(msg), BorderLayout.CENTER);

}

}

**Using Insets**

➢ To leave a small amount of space between the container that holds and the window that contains it.

➢ To do this, override the getInsets( ) method that is defined by Container. This function returns an Insets object that contains the top, bottom, left, and right inset to be used when the container is displayed.

➢ These values are used by the layout manager to inset the components when it lays out the window.

➢ The constructor for Insets is shown here:

Insets(int *top*, int *left*, int *bottom*, int *right*)

➢ The values passed in *top*, *left*, *bottom*, and *right* specify the amount of space between the container and its enclosing window.

The **getInsets( )** method has this general form:

Insets getInsets( )

➢ When overriding one of these methods, must return a new **Insets** object that contains the inset spacing desire.

➢ **BorderLayout** example modified that it insets components ten pixels from each border. ➢ The background color has been set to cyan to make the insets more visible. **GridLayout**

GridLayout lays out components in a two-dimensional grid. To instantiate a GridLayout, define number of rows and columns.

➢ The constructors supported by GridLayout are:

i) GridLayout( )

M.Sc. Semester – I (Computer Science) 26

Programming in Java Dr. M. T. Wanjari

ii) GridLayout(int *numRows*, int *numColumns* )

iii) GridLayout(int *numRows*, int *numColumns*, int *horz*, int *vert*)

➢ The first constructor creates a single-column grid layout.

➢ The second constructor creates a grid layout with the specified number of rows and columns.

➢ The third constructor allows to specify the horizontal and vertical space left between components in *horz* and *vert*, variables respectively. Either *numRows* or *numColumns* can be zero. Specifying *numRows* as zero allows for unlimited-length columns. Specifying *numColumns* as zero allows for unlimited-length rows.

// Demonstrate GridLayout

import java.awt.\*;

import java.applet.\*;

public class gm\_Applet extends Applet

{

public void init()

{

Label fnLabel = new Label("First Name");

Label lnLabel = new Label("Last Name");

TextField fnText = new TextField(10);

TextField lnText = new TextField(10);

setLayout(new GridLayout(2,2));

add(fnLabel);

add(fnText);

add(lnLabel);

add(lnText);

}

}

**CardLayout**

➢ The CardLayout class is unique among the other layout managers in that it stores several different layouts.

M.Sc. Semester – I (Computer Science) 27

Programming in Java Dr. M. T. Wanjari

➢ Each layout can be thought of as being on a separate index card in a deck that can be shuffled so that any card is on top at a given time. This can be useful for user interfaces with optional components that can be dynamically enabled and disabled upon user input. ➢ There are two constructors of CardLayout class:

CardLayout( )

CardLayout(int *horz*, int *vert*)

The first form creates a default card layout.

The second form allows to specify the horizontal and vertical space left between components in *horz* and *vert,* respectively.

➢ The cards are typically held in an object of type Panel. And the card form the deck are also typically objects of type Panel.

➢ Thus, programmer creates a panel that contains the deck and a panel for each card in a deck. And assign cardlayout to each panel.

➢ When card panels are added to a panel, they are usually given a name. and to add these cards there is a add( ) method.

void add(Component *panelObj,* Object *name*);

where, *name* is a string that specifies the name of the card whose panel is specified by *panelObj*.

➢ After creating a deck, to calling specific card there are methods it defined by CardLayout void first(Container *deck*)

void last(Container *deck*)

void next(Container *deck*)

void previous(Container *deck*)

void show(Container *deck*, String *cardName*)

where, *deck* is a reference to the container i.e. panel that holds the cards and *cardName* is the name of a card.

➢ Calling first( ) causes the first card in the deck to be shown. To show the last card, call last( ). To show the next card, call next( ). To show the previous card, call previous( ). ➢ To show( ) method displays the card whose name is passed in *cardName.*

M.Sc. Semester – I (Computer Science) 28

Programming in Java Dr. M. T. Wanjari

**GridBagLayout**

➢ GridBagLayout is specified by GridBagLayout class.

➢ The grid bag useful is that it can specify the relative placement of components by specifying their positions within cells inside a grid.

➢ The key to the grid bag is that each component can be a different size, and each row in the grid can have a different no. of columns. This is why Layout is gridbag. ➢ It is a collection of small grid joined together. The location and size of each component in grid bag are determined by a set of constraints link to it.

➢ The constraints are contained in an object of type GridBagConstraints. **GridBagConstraints**

➢ Constraints include the height and width of a cell and the placement of component, its alignment and its anchor point within the cell.

➢ The general procedure for using a gridbag is to first create a new GridBagLayout object and to make it the current Layout manager. Then, set the constraints that apply to each component that will be added to the gridbag. Finally, add the components to the layout managers.

➢ It is quit complicated than other Layout managers but it support the exact placement of component in a window.

➢ It supports only one constructor.

i) GridBagLayout()

➢ There are many methods

ii) setConstraints()

void setConstraints(Component comp, GridConstraints gbc)

where comp is a component for which the constraints specified by gbc. This method sets the constraints that can apply to each component in the gridbag.

➢ The key to successfully using GridBagLayout is the proper setting of the constraints, which are stored in a GridBagConstraints object.

➢ GridBagConstraints defines several static fields that contain standard constraint values such as GridBagConstraints.CENTER & GridBagConstraints.VERTICAL. ➢ When component is smaller than its cell, programmer can use the anchor field to specify where within the cell the components top left corner will be located.

M.Sc. Semester – I (Computer Science) 29

Programming in Java Dr. M. T. Wanjari

➢ There are three types of values give to another. Listed as follows

First: GridBagConstraints.CENTER

GridBagConstraints.EAST

GridBagConstraints.NORTH

GridBagConstraints.NORTHEAST & so on

Second: second value is relative

GridBagConstraints.FIRST\_LINE\_END

GridBagConstraints.FIRST\_LINE\_START & so on

Third: third type of value given to anchor were added by Java SE6. They are GridBagConstraints.BASELINE

GridBagConstraints.ABOVE\_BASELINE

➢ The weightx & weighty fields are also important & their values determine how much of the extra space within a container is allocated to each row and column. By default both of these values are zero.

➢ That’s all of these information about GridBagLayout makes it powerful than other Layout managers.

**AWT Components**

➢ AWT supports various types of components Buttons, Lists, Labels, checkboxes, Scrollbars, Textediting(TextArea, TextField)

➢ All these components are subclass of class component.

➢ To add a component in a window, firstly create an object of the component and all it to window by calling add(), which is defined by class container.

➢ General form of add()

Component add(Component obj)

Where obj is an object of a component which we want to add.

➢ Once component is added, it will automatically be visible whenever its parent window is displayed.

➢ If you want to remove the component from window, call remove() method. This is also defined by container class.

void remove (Component obj)

➢ For remove all Components from window, calling removeAll()

M.Sc. Semester – I (Computer Science) 30

Programming in Java Dr. M. T. Wanjari

**Labels**

It is easiest control.

➢ A *label* is an object of type Label, and it contains a string. Label l1 = new Label(); which it displays.

➢ Labels are passive controls that do not support any interaction with the user. ➢ **Label** defines the following constructors:

Label( )

it creates a blank label

Label(String *str*)

it creates a label that contains string specified by str.

Label(String *str*, int *how*)

this creates a label with the string specified by str & how given by following constants. Label.LEFT, Label.RIGHT, or Label.CENTER.

➢ To set or change the text in a label use the method

void setText(String *str*) &

obtain the current Label by calling getText( )

String getText()

➢ To set an alignment of the string within the label by calling setAlignment( ). & obtain the current alignment, call getAlignment( ).

void setAlignment(int *how*)

int getAlignment( )

where, *how* specified by constants specified earlier.

**Buttons**

➢ This component is mostly used and is also called as push button.

➢ A button is a component that contains a label and that generates an event when it is pressed.

Button b1 = new Button();

b1 is an object of class Button

➢ There are two constructors

i) Button( )

This creates a button without label

M.Sc. Semester – I (Computer Science) 31

Programming in Java Dr. M. T. Wanjari

ii) Button(String *str*)

This creates a button with Label specified by str.

➢ If creating a button without label or to change the Labels by calling setLabel() & get Label of component by calling getLabel( ).

void setLabel(String *str*)

String getLabel( )

where, *str is a* new label for the button.

**Handling Buttons**

➢ Each time a button is pressed, an action event is generated.

➢ This sent to an ActionListener interface. Which implements and defines the actionPerformed( ) method, which is called when an event occurs. An ActionEvent object is supplied as the argument to this method. It contains both a reference to the button that generated the event and a string that is the label of the button.

➢ For eg. if Frame contains three buttons & any one of them is pressed a message is displayed in the frame.

**Checkboxes**

➢ A *check box* is a control which is used to turn an option on or off.

➢ It consists of a small box that can either contain a check mark or not.

➢ There is a label associated with each check box that describes which option the box represents.

➢ Checkbox supports following constructors:

i) Checkbox( )

This creates a check box whose label is initially blank.

ii) Checkbox(String *str*)

This creates a check box whose label is specified by *str.*

iii) Checkbox(String *str*, boolean *on*)

This form allows to set the initial state of the check box. if true, then it is initially checked; otherwise, it is cleared.

iv) Checkbox(String *str*, boolean *on*, CheckboxGroup *cbGroup*)

v) Checkbox(String *str*, CheckboxGroup *cbGroup*, boolean *on*)

M.Sc. Semester – I (Computer Science) 32

Programming in Java Dr. M. T. Wanjari

These fourth and fifth creates a check boxes with labels is specified by *str* and whose group is specified by *cbg.* If check box is not part of a group, then set it to null. ➢ To retrieve the current state of a check box, call getState( ).

➢ To set its state, call setState( ).

➢ To set the label, call setLabel( ).

➢ To set Label call getLabel().

boolean getState( )

void setState(boolean *on*)

String getLabel( )

void setLabel(String *str*)

where, if *on* is **true**, the box is checked. If it is **false**, the box is cleared. The string passed in *str* becomes the new label associated with the invoking check box. **Handling Checkboxes**

➢ Each time a check box is selected or deselected, an item event is generated. ➢ This sent to any listeners that previously registered an interest in receiving item event notifications from that component.

➢ Each listener implements the ItemListener interface.

➢ That interface defines the itemStateChanged( ) method.

➢ An ItemEvent object is supplied as the argument to this method. It contains information about the event i.e. selection or deselection.

**Checkbox Group**

➢ It is possible to create a set of mutually exclusive check boxes in which one and only one check box in the group can be checked at any one time. These check boxes are called radio buttons.

➢ To create set of mutually exclusive check boxes, first define the group to which they will belong and then specify that group when you construct the check boxes.

➢ Only the default constructor is defined, which creates an empty group. ➢ To determine which check box in a group is currently selected by calling getSelectedCheckbox( ).

➢ To set a check box by calling setSelectedCheckbox( ).

➢ Checkbox getSelectedCheckbox( )

M.Sc. Semester – I (Computer Science) 33

Programming in Java Dr. M. T. Wanjari

void setSelectedCheckbox(Checkbox *which*)

where, *which* is the check box selected. The previously selected check box will be turned off. Here is a program that uses check boxes that are part of a group:

**Choice Controls (Combo box)**

➢ The Choice class is used to create a pop-up list of items from which the user may choose. Thus, a Choice control is a form of menu.

➢ When inactive, it shows only currently selected item & when the user clicks on it, the whole list of choices pops up, and a new selection possible.

➢ Each item in the list is a string that appears as a left-justified label in the order it is added to the Choice object.

➢ It have only default constructor, which creates an empty list.

➢ To add a selection to the list, call add( ).

void add(String str)

➢ To determine which item is currently selected, call either getSelectedItem( ) or getSelectedIndex( ).

String getSelectedItem( ) //return selected string

int getSelectedIndex( ) //returns selected index

➢ To obtain the number of items in the list, call **getItemCount( )**.

➢ To set the current selected item which either index or a string by the select( ) int getItemCount( )

void select(int *index*)

void select(String *name*)

**Handling Choice Lists**

➢ Each time a choice Lists is selected, an item event is generated.

➢ This implements the **ItemListener** interface.

➢ That interface defines **itemStateChanged( )** method. An **ItemEvent** class object is supplied as the argument to this method.

**Lists**

➢ The **List** class provides a compact, multiple-choice, scrolling selection list. ➢ It is not like choicelist because it shows, multiple items in single visible window. ➢ It can also be created to allow multiple selections.

M.Sc. Semester – I (Computer Science) 34

Programming in Java Dr. M. T. Wanjari

➢ Constructors are

i) List( )

This creates a **List** control that allows only one item to be selected at any one time. ii) List(int nRows)

This creates the list where nRows specifies the number of entries in the list. iii) List(int nRows, boolean multipleSelect)

This sets multiselection & if its value is true, then the user may select two or more items at a time. If it is false, only one item may be selected.

➢ To add items in the list, call **add( )**.

void add(String *name*)

void add(String *name*, int *index*)

➢ Indexing begins at zero. You can specify –1 to add the item to the end of the list. ➢ Only single selection, by calling either **getSelectedItem( )** or **getSelectedIndex( )**. String getSelectedItem( )

int getSelectedIndex( )

➢ For multiple selection by calling, **getSelectedItems( )** or **getSelectedIndexes( )**. String[ ] getSelectedItems( ) //returns an array

int[ ] getSelectedIndexes( ) //returns an array

➢ To obtain the total no. of items, calling getItemCount( ) and set currently selected item by using select()

int getItemCount()

void Select (int index)

➢ To obtain the name of the item at specified index by calling getItem (int index) method. String getItem(int index);

**Handling Lists**

➢ To process list events, implement the **ActionListener** interface.

➢ Each time a **List** item is double-clicked, an **ActionEvent** object is generated. Its **getActionCommand( )** method can be used to retrieve the name of the newly selected item.

➢ Also, each time an item is selected or deselected with a single click, an **ItemEvent** object is generated.

M.Sc. Semester – I (Computer Science) 35

Programming in Java Dr. M. T. Wanjari

➢ Its **getStateChange( )** method can be used to determine whether a selection or deselection triggered this event. **getItemSelectable( )** returns a reference to the object that triggered this event.

**ScrollBars**

➢ *Scroll bars* are used to select continuous values between a specified minimum and maximum.

➢ Scroll bars may be oriented horizontally or vertically.

➢ A scroll bar is actually a composite of several individual parts.

➢ Each end has an arrow that click to move the current value of the scroll bar one unit in the direction of the arrow.

➢ It defines the following constructors:

i) Scrollbar( )

This creates a vertical scroll bar.

ii) Scrollbar(int *style*)

This creates a scroll bar by using a *style & it* is specified by values Scrollbar.VERTICAL and Scrollbar.HORIZONTAL,

iii) Scrollbar(int *style,* int *initialValue*, int *thumbSize*, int *min*, int *max*) This creates a scrollbar with style values specified as earlier and initial value is specified, the number of units represented by the height of the thumb is passed in *thumbSize.* ➢ When programmer can use first constructor & after that want to set initial values. This can be to do so with the help of **setValue( )** method & to get values by getValue(). **Handling Scrollbars**

➢ To process scroll bar events, there is need to implement the **AdjustmentListener** interface.

➢ Each time a user interacts with a scroll bar, an **AdjustmentEvent** object is generated. Its **getAdjustmentType( )** method can be used to determine the type of the adjustment. ➢ The types of adjustment events are as follows:

BLOCK\_DECREMENT A page-down event has been generated.

BLOCK\_INCREMENT A page-up event has been generated.

TRACK An absolute tracking event has been generated.

UNIT\_DECREMENT The line-down button in a scroll bar has been pressed.

M.Sc. Semester – I (Computer Science) 36

Programming in Java Dr. M. T. Wanjari

UNIT\_INCREMENT The line-up button in a scroll bar has been pressed.

**TextField**

➢ This **TextField** class implements a single-line text-entry area, usually called an *edit control.*

➢ Text fields allow the user to enter strings and to edit the text using the arrow keys, cut and paste keys.

➢ **TextField** is a subclass of **TextComponent**.

➢ **TextField** defines the following constructors:

i) TextField( )

This creates a default text field.

ii) TextField(int *numChars*)

Number of characters are specified with nochars.

iii) TextField(String *str*)

initialized a textfield with the string *str.*

iv) TextField(String *str*, int *numChars*)

initialize a text field and sets its width.

➢ To obtain the string currently contained in the text field, call **getText( )**. To set the text, call **setText( )**.

➢ These methods are as follows:

String getText( )

setEditable(true/false)

isEditable()

➢ To enter text but that is not displayed such as password for to do so, call setEchoChar() method.

**Handling TextField**

➢ After entering a text in a textbox & user press the ENTER. Then ActionEvent is generated & it calls actionPerformed() after the ActionListener interface.

**TextArea**

➢ When multiple lines are entered to textfield then it is not support for such purpose & to do that operation. TextArea class is used.

➢ Following constructors are

M.Sc. Semester – I (Computer Science) 37

Programming in Java Dr. M. T. Wanjari

TextArea( )

TextArea(int *numLines,* int *numChars*)

TextArea(String *str*)

TextArea(String *str*, int *numLines*, int *numChars*)

TextArea(String *str*, int *numLines*, int *numChars*, int *sBars*)

where, *numLines -* specifies the height, in lines, of the text area,

*numChars -* specifies its width, in characters.

str - Initial text

*sBars -* SCROLLBARS\_BOTH

SCROLLBARS\_NONE

SCROLLBARS\_HORIZONTAL\_ONLY

SCROLLBARS\_VERTICAL\_ONLY

➢ It is a subclass of **TextComponent**. So that it supports the **getText( )**, **setText( )**, **getSelectedText( )**, **select( )**, **isEditable( )**, and **setEditable( )** methods.

It adds mor methods:

void append(String *str*)

The **append( )** method appends the string specified by *str* to the end of the current text. void insert(String *str*, int *index*)

**insert( )** inserts the string passed in *str* at the specified index.

void replaceRange(String *str*, int *startIndex*, int *endIndex*)

To replace text,

**MenuBars and Menus**

➢ A menu bar displays a list of top-level menu choices.

➢ Each choice is associated with a drop-down menu.

➢ This concept is implemented in Java by the following classes:

**MenuBar**, **Menu**, and **MenuItem**.

➢ In general, a menu bar contains one or more **Menu** objects.

➢ Each **Menu** object contains a list of **MenuItem** objects.

➢ Each **MenuItem** object represents something that can be selected by the user. Since **Menu** is a subclass of **MenuItem**, a hierarchy of nested submenus can be created.

M.Sc. Semester – I (Computer Science) 38

Programming in Java Dr. M. T. Wanjari

➢ These are menu options of type **CheckboxMenuItem** and will have a check mark next to them when they are selected.

➢ To create a menu bar, first create an instance of **MenuBar**.

➢ This class only defines the default constructor, create instances of **Menu** that will define the selections displayed on the bar.

➢ The constructors for **Menu**:

Menu( )

Menu(String *optionName*)

Menu(String *optionName*, boolean *removable*)

where, *optionName* specifies the name of the menu selection. If *removable* is **true**, the pop-up menu can be removed and allowed to float free. Otherwise, it will remain attached to the menu bar.

➢ To creates an empty menu, individual menu items are of type **MenuItem**. It defines these constructors:

MenuItem( )

MenuItem(String *itemName*)

MenuItem(String *itemName*, MenuShortcut *keyAccel*)

where, *itemName* is the name shown in the menu, and *keyAccel* is the menu shortcut for this item. It can disable or enable a menu item by using the **setEnabled( )** method. Its form is shown here:

void setEnabled(boolean *enabledFlag*)

If the argument *enabledFlag* is **true**, the menu item is enabled. If **false**, the menu item is disabled. it can determine an item’s status by calling **isEnabled( )**. This method is shown here: boolean isEnabled( )

**isEnabled( )** returns **true** if the menu item on which it is called is enabled. Otherwise, it returns **false**. You can change the name of a menu item by calling **setLabel( )**, retrieve the current name by using **getLabel( )**. These methods are as follows:

void setLabel(String *newName*)

String getLabel( )

M.Sc. Semester – I (Computer Science) 39

Programming in Java Dr. M. T. Wanjari

where, *newName* becomes the new name of the invoking menu item. **getLabel( )** returns the current name. You can create a checkable menu item by using a subclass of **MenuItem** called **CheckboxMenuItem**. It has these constructors:

CheckboxMenuItem( )

CheckboxMenuItem(String *itemName*)

CheckboxMenuItem(String *itemName*, boolean *on*)

where, *itemName* is the name shown in the menu. Checkable items operate as toggles. Each time one is selected, its state changes. In the first two forms, the checkable entry is unchecked. In the third form, if *on* is **true**, the checkable entry is initially checked. Otherwise, it is cleared. You can obtain the status of a checkable item by calling **getState( )**. You can set it to a known state by using **setState( )**. These methods are shown here:

boolean getState( )

void setState(boolean *checked*)

➢ If the item is checked, **getState( )** returns **true**. Otherwise, it returns **false**. ➢ To check an item, pass **true** to **setState( )**.

➢ To clear an item, pass **false**, created a menu item, must add the item to a **Menu** object by using **add( )**, which has the following general form:

MenuItem add(MenuItem *item*)

where, *item* is the item being added. Items are added to a menu in the order in which the calls to **add( )** take place. The *item* is returned. Once you have added all items to a **Menu** object, you can add that object to the menu bar by using this version of **add( )** defined by **MenuBar**: Menu add(Menu *menu*)

Here, *menu* is the menu being added. The *menu* is returned.

➢ Menus only generate events when an item of type **MenuItem** or **CheckboxMenuItem** is selected.

➢ Do not generate events when a menu bar is accessed to display a drop-down menu, for example. Each time a menu item is selected, an **ActionEvent** object is generated. ➢ Each time a check box menu item is checked or unchecked, an **ItemEvent** object is generated.

M.Sc. Semester – I (Computer Science) 40

Programming in Java Dr. M. T. Wanjari

➢ Thus, must implement the **ActionListener** and **ItemListener** interfaces in order to handle these menu events. The **getItem( )** method of **ItemEvent** returns a reference to the item that generated this event. The general form of this method is shown here: Object getItem( )

**Swing**

➢ Swing is a set of classes that provides more powerful and flexible components than are possible with the AWT.

➢ The familiar components, such as buttons, check boxes, and labels, Swing supplies several exciting additions, including tabbed panes, scroll panes, trees, and tables. ➢ Even familiar components such as buttons have more capabilities in Swing. ➢ For example, a button may have both an image and a text string associated with it. Also, the image can be changed as the state of the button changes.

➢ Unlike AWT components, Swing components are not implemented by platform-specific code.

➢ Instead, they are written entirely in Java and, therefore, are platform-independent. ➢ The term *lightweight* is used to describe such elements.

➢ The number of classes and interfaces in the Swing packages is substantial. ➢ Swing is an area want to explore further on own.

➢ The Swing-related classes are contained in **javax.swing** and its subpackages, such as **javax.swing.tree**.

**JApplet**

➢ Fundamental to Swing is the **JApplet** class, which extends **Applet**.

➢ Applets that use Swing must be subclasses of **JApplet**.

➢ **JApplet** is rich with functionality that is not found in **Applet**.

➢ For example, **JApplet** supports various “panes,” such as the content pane, the glass pane, and the root pane.

➢ However, one difference between **Applet** and **JApplet** is important, because it is used by the sample applets. When adding a component to an instance of **JApplet**, do not invoke the **add( )** method of the applet.

➢ Instead, call **add( )** for the *content pane* of the **JApplet** object.

➢ The content pane can be obtained via the method shown here:

M.Sc. Semester – I (Computer Science) 41

Programming in Java Dr. M. T. Wanjari

Container getContentPane( )

➢ The **add( )** method of **Container** can be used to add a component to a content pane. Its form is shown here:

void add(*comp*)

where, *comp* is the component to be added to the content pane.

**Icons and Labels**

➢ In Swing, icons are encapsulated by the **ImageIcon** class, which paints an icon from an image. Two of its constructors are shown here:

ImageIcon(String *filename*)

It uses the image in the file named *filename*.

ImageIcon(URL *url*)

It uses the image in the resource identified by *url*.

➢ The **ImageIcon** class implements the **Icon** interface that declares the methods shown here:

Method Description

int getIconHeight( ) Returns the height of the icon in pixels.

int getIconWidth( ) Returns the width of the icon in pixels.

void paintIcon(Component *comp*, Graphics *g*, int *x*, int *y*)

➢ Paints the icon at position *x*, *y* on the graphics context *g*.

➢ Additional information about the paint operation can be provided in *comp*. ➢ Swing labels are instances of the **JLabel** class, which extends **JComponent**. ➢ It can display text and/or an icon. Some of its constructors are shown here: JLabel(Icon *i*)

Label(String *s*)

JLabel(String *s*, Icon *i*, int *align*)

where, *s* and *i* are the text and icon used for the label. The *align* argument is either **LEFT**, **RIGHT**, **CENTER**, **LEADING**, or **TRAILING**.

➢ These constants are defined in the **SwingConstants** interface, along with several others used by the Swing classes.

➢ The icon and text associated with the label can be read and written by the following methods:

M.Sc. Semester – I (Computer Science) 42

Programming in Java Dr. M. T. Wanjari

Icon getIcon( )

String getText( )

void setIcon(Icon *i*)

void setText(String *s*)

where, *i* and *s* are the icon and text, respectively.

**Text Fields**

➢ The Swing text field is encapsulated by the **JTextComponent** class, which extends **JComponent**.

➢ It provides functionality that is common to Swing text components. One of its subclasses is **JTextField**, which allows you to edit one line of text. Some of its constructors are shown here:

JTextField( )

JTextField(int *cols*)

JTextField(String *s*, int *cols*)

JTextField(String *s*)

Here, *s* is the string to be presented, and *cols* is the number of columns in the text field. **Buttons**

➢ Swing buttons provide features that are not found in the **Button** class defined by the AWT.

➢ For example, Swing buttons are subclasses of the **AbstractButton** class, which extends **JComponent**.

➢ **AbstractButton** contains many methods that allow you to control the behavior of buttons, check boxes, and radio buttons. The following are the methods that control this behavior:

void setDisabledIcon(Icon *di*)

void setPressedIcon(Icon *pi*)

void setSelectedIcon(Icon *si*)

void setRolloverIcon(Icon *ri*)

where, *di*, *pi*, *si*, and *ri* are the icons to be used for these different conditions. ➢ The text associated with a button can be read and written via the following methods: String getText( )

M.Sc. Semester – I (Computer Science) 43

Programming in Java Dr. M. T. Wanjari

void setText(String *s*)

where, *s* is the text to be associated with the button.

➢ Concrete subclasses of **AbstractButton** generate action events when they are pressed. Listeners register and unregister for these events via the methods shown here: void addActionListener(ActionListener *al*)

void removeActionListener(ActionListener *al*)

where, *al* is the action listener. **AbstractButton** is a superclass for push buttons, check boxes, and radio buttons. Each is examined next.

**The JButton Class**

➢ The **JButton** class provides the functionality of a push button.

➢ **JButton** allows an icon, a string, or both to be associated with the push button. Some of its constructors are shown here:

JButton(Icon *i*)

JButton(String *s*)

JButton(String *s*, Icon *i*)

Here, *s* and *i* are the string and icon used for the button.

**Check Boxes**

➢ The **JCheckBox** class, which provides the functionality of a check box, is a concrete implementation of **AbstractButton**.

➢ Its immediate superclass is **JToggleButton**, which provides support for two-state buttons. Some of its constructors are shown here:

JCheckBox(Icon *i*)

JCheckBox(Icon *i*, boolean *state*)

JCheckBox(String *s*)

JCheckBox(String *s*, boolean *state*)

JCheckBox(String *s*, Icon *i*)

JCheckBox(String *s*, Icon *i*, boolean *state*)

where, *i* is the icon for the button. The text is specified by *s*. If *state* is **true**, the check box is initially selected. Otherwise, it is not. The state of the check box can be changed via the following method:

void setSelected(boolean *state*)

M.Sc. Semester – I (Computer Science) 44

Programming in Java Dr. M. T. Wanjari

where, *state* is **true** if the check box should be checked.

**Radio Buttons**

➢ Radio buttons are supported by the **JRadioButton** class, which is a concrete implementation of **AbstractButton**.

➢ Its immediate superclass is **JToggleButton**, which provides support for two-state buttons. Some of its constructors are shown here:

JRadioButton(Icon *i*)

JRadioButton(Icon *i*, boolean *state*)

JRadioButton(String *s*)

JRadioButton(String *s*, boolean *state*)

JRadioButton(String *s*, Icon *i*)

JRadioButton(String *s*, Icon *i*, boolean *state*)

where, *i* is the icon for the button. The text is specified by *s*. If *state* is **true**, the button is initially selected. Otherwise, it is not. Radio buttons must be configured into a group. Only one of the buttons in that group can be selected at any time.

**Combo Boxes**

➢ Swing provides a *combo box* through the **JComboBox** class, which extends **JComponent**.

➢ A combo box normally displays one entry.

➢ However, it can also display a drop-down list that allows a user to select a different entry. ➢ It can also type your selection into the text field. Two of **JComboBox**’s constructors are shown here:

JComboBox( )

JComboBox(Vector *v*)

where, *v* is a vector that initializes the combo box.

➢ Items are added to the list of choices via the **addItem( )** method, whose signature is shown here:

void addItem(Object *obj*)

where, *obj* is the object to be added to the combo box.

**Tabbed Panes**

➢ A *tabbed pane* is a component that appears as a group of folders in a file cabinet.

M.Sc. Semester – I (Computer Science) 45

Programming in Java Dr. M. T. Wanjari

➢ Each folder has a title. When a user selects a folder, its contents become visible. Only one of the folders may be selected at a time.

➢ Tabbed panes are commonly used for setting configuration options.

➢ Tabbed panes are encapsulated by the **JTabbedPane** class, which extends **JComponent**. We use its default constructor. Tabs are defined via the following method: void addTab(String *str*, Component *comp*)

where, *str* is the title for the tab, and *comp* is the component that should be added to the tab. ➢ Typically, a **JPanel** or a subclass of it is added. The general procedure to use a tabbed pane in an applet is outlined here:

1. Create a **JTabbedPane** object.

2. Call **addTab( )** to add a tab to the pane. (The arguments to this method define the title of the tab and the component it contains.)

3. Repeat step 2 for each tab.

4. Add the tabbed pane to the content pane of the applet.

**Scroll Panes**

➢ A *scroll pane* is a component that presents a rectangular area in which a component may be viewed.

➢ Horizontal and/or vertical scroll bars may be provided if necessary.

➢ Scroll panes are implemented in Swing by the **JScrollPane** class, which extends **JComponent**. Some of its constructors are shown here:

JScrollPane(Component *comp*)

JScrollPane(int *vsb*, int *hsb*)

JScrollPane(Component *comp*, int *vsb*, int *hsb*)

where, *comp* is the component to be added to the scroll pane. *vsb* and *hsb* are **int** constants that define when vertical and horizontal scroll bars for this scroll pane are shown. These constants are defined by the **ScrollPaneConstants** interface. Some examples of these constants are described as follows:

Constant Description

HORIZONTAL\_SCROLLBAR\_ALWAYS Always provide horizontal

scroll bar

HORIZONTAL\_SCROLLBAR\_AS\_NEEDED Provide horizontal scroll bar,

M.Sc. Semester – I (Computer Science) 46

Programming in Java Dr. M. T. Wanjari

if needed

VERTICAL\_SCROLLBAR\_ALWAYS Always provide vertical

scroll bar

VERTICAL\_SCROLLBAR\_AS\_NEEDED Provide vertical scroll bar, if needed Here are the steps that you should follow to use a scroll pane in an applet: 1. Create a **JComponent** object.

2. Create a **JScrollPane** object. (The arguments to the constructor specify the component and the policies for vertical and horizontal scroll bars.)

3. Add the scroll pane to the content pane of the applet.

**Trees**

➢ A *tree* is a component that presents a hierarchical view of data.

➢ A user has the ability to expand or collapse individual subtrees in this display. ➢ Trees are implemented in Swing by the **JTree** class, which extends **JComponent**. Some of its constructors are shown here:

JTree(Hashtable *ht*)

JTree(Object *obj*[ ])

JTree(TreeNode *tn*)

JTree(Vector *v*)

The first form creates a tree in which each element of the hash table *ht* is a child node. Each element of the array *obj* is a child node in the second form. The tree node *tn* is the root of the tree in the third form. Finally, the last form uses the elements of vector *v* as child nodes.

➢ A **JTree** object generates events when a node is expanded or collapsed. ➢ The **addTreeExpansionListener( )** and **removeTreeExpansionListener( )** methods allow listeners to register and unregister for these notifications. The signatures of these methods are shown here:

void addTreeExpansionListener(TreeExpansionListener *tel*)

void removeTreeExpansionListener(TreeExpansionListener *tel)*

where, *tel* is the listener object.

➢ The **getPathForLocation( )** method is used to translate a mouse click on a specific point of the tree to a tree path. Its signature is shown here:

TreePath getPathForLocation(int *x*, int *y*)

M.Sc. Semester – I (Computer Science) 47

Programming in Java Dr. M. T. Wanjari

where, *x* and *y* are the coordinates at which the mouse is clicked.

➢ The return value is a **TreePath** object that encapsulates information about the tree node that was selected by the user.

➢ The **TreePath** class encapsulates information about a path to a particular node in a tree. It provides several constructors and methods. In this book, only the **toString( )** method is used.

➢ It returns a string equivalent of the tree path. The **TreeNode** interface declares methods that obtain information about a tree node.

➢ For example, it is possible to obtain a reference to the parent node or an enumeration of the child nodes. The **MutableTreeNode** interface extends **TreeNode**. It declares methods that can insert and remove child nodes or change the parent node.

➢ The **DefaultMutableTreeNode** class implements the **MutableTreeNode** interface. ➢ It represents a node in a tree. One of its constructors is shown here:

DefaultMutableTreeNode(Object *obj*)

where, *obj* is the object to be enclosed in this tree node. The new tree node doesn’t have a parent or children. To create a hierarchy of tree nodes, the **add( )** method of **DefaultMutableTreeNode**

can be used. Its signature is shown here:

void add(MutableTreeNode *child*)

where, *child* is a mutable tree node that is to be added as a child to the current node. ➢ Tree expansion events are described by the class **TreeExpansionEvent** in the **javax.swing.event** package.

➢ The **getPath( )** method of this class returns a **TreePath** object that describes the path to the changed node. Its signature is shown here:

TreePath getPath( )

➢ The **TreeExpansionListener** interface provides the following two methods: void treeCollapsed(TreeExpansionEvent *tee*)

void treeExpanded(TreeExpansionEvent *tee*)

where, *tee* is the tree expansion event. The first method is called when a subtree is hidden, and the second method is called when a subtree becomes visible. Here are the steps that you should follow to use a tree in an applet:

M.Sc. Semester – I (Computer Science) 48

Programming in Java Dr. M. T. Wanjari

1. Create a **JTree** object.

2. Create a **JScrollPane** object. (The arguments to the constructor specify the tree and the policies for vertical and horizontal scroll bars.)

3. Add the tree to the scroll pane.

4. Add the scroll pane to the content pane of the applet.

**Tables**

➢ A *table* is a component that displays rows and columns of data.

➢ We can drag the cursor on column boundaries to resize columns.

➢ We can also drag a column to a new position.

➢ Tables are implemented by the **JTable** class, which extends **JComponent**. One of its constructors is shown here:

JTable(Object *data*[ ][ ], Object *colHeads*[ ])

where, *data* is a two-dimensional array of the information to be presented, and *colHeads* is a one-dimensional array with the column headings. Here are the steps for using a table in an applet:

1. Create a **JTable** object.

2. Create a **JScrollPane** object. (The arguments to the constructor specify the table and the policies for vertical and horizontal scroll bars.)

3. Add the table to the scroll pane.

4. Add the scroll pane to the content pane of the applet.

**---------------------------------------------------------------------------------------------------------------------**

M.Sc. Semester – I (Computer Science) 49