5/9/18

* Wrapper classes:

conversion programs

Data Types:

Primitive / Intrinsic :

		Wrapper class
1. short	numeric	Short
2. byte	numeric	Byte
3. int	numeric	Integer
4. long	numeric	Long
5. float		Float
6. double		Double
7. char		character
8. boolean		Boolean

There is a wrapper class corresponding to each primitive data type which can be used to convert primitive data type to object type and vice versa.

wrapper class - java.lang package

eg. `int n = 25;` // n is primitive DT.
`Integer y = new Integer(5);` // y is object type

Object type variable does not store ~~the~~ value but the reference.

```
int z = n+y;  ×
int z = n+y.intValue();  // converts object to primitive
```

parseInt: converts string to decimal

```
int n = Integer.parseInt("1234");
```

toString(): converts to string

```
String s = Integer.toString(1234);
```

Integer Wrapper class:

Constructors : `Integer(i)` int to int object
`Integer(s)` string to string object

Class Methods: parseInt(s)
toString(n)

Instance Methods: byteValue(), doubleValue(), floatValue(), intValue(), longValue(), shortValue(), toString()
object to concerned data type

✱ AWT (Abstract Window Toolkit):

Huge package containing large collection of classes, interface etc.

Elements: Button, Canvas – create a blank window

controls: checkbox, Checkbox Group, Choice, Colour, Component, Container, Dialog

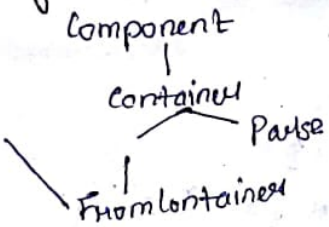
Layout Manager: Position the elements in pre-specified manner.
FlowLayout Manager

Font class, Frame – title bar, resize

~~New~~ NewBar Menu, MenuBar, MenuIndex, ScrollBar

TextArea, TextView

class heirarchy:



void setVisible ( )

6/9/18

Panel: Space in an application

public class Panel extends Container.
public Panel()
public Panel (LayoutManager layout)
FlowLayout

Component: public abstract class ~~Container~~ should component ~~extends~~
extends object.

Container: contains components.
public ~~class~~ abstract class Container extends Components

Window: public class Window extends Container
BorderLayout
public Window (Frame parent)

Frame: public class Frame extends Window implements
MenuContainer.

Canvas: public class Canvas extends Component.

Event Handling:

eg. input.java.awt.*;
input.java.applet.*;
input.java.event.*;

public class myevent extends Applet implements ActionListener.
{
Button b
public void init()
{ b = new Button ("press me");
add(b);
b.addActionListener(this);
public void actionPerformed (ActionEvent ae)
{
if(ae.getSource()==b)
setBackground(Color.yellow);

String / StringBuffer / StringBuilder :

Immutable
Mutable

String str1 = "xyz";
char ch = {'x','y','z'};
String str1 = new String("xyz");

	StringBuffer	StringBuilder
Storage area	const string pool	heap
object mutability		
Thread safe	Safer	
Performance		

chars

classes

StringBuffer sb = new SB();
sb.append("Geeks");
concat3(sb);
Sop("SB:"+sb);

class XYZ
{
public static void concat1(String s1)
{
s2 = s1 + "forgeeks"; // concats to string builder
public static void concat2(String s2)
{ s2.append("forgeeks");
public static void concat3(StringBuffer sb)
{ sb.append("forGeeks");
public void main() { String s1 = "Geeks"; concat1(s1);
Sop("String:"+s1); StringBuilder s2 = new StringBuilder("Geeks");
concat2(s2); Sop("SB:"+s2);

PU

12/9/18

Swing components are platform independent, awt are dependent.
AWT is heavyweight.

* Layout Manager:

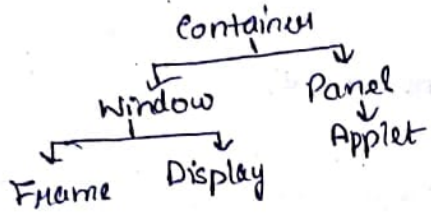
Java uses the following layout managers:

i) Flow layout (default)
ii) Border layout
iii) Grid layout
iv) Card layout

Methods and their prototype:

public void add (Component c)
public void setSize (int width, int length)
public void ~~self~~ setLayout (LayoutManager m)
public void setVisible (boolean status) status is false by default.

Components: button, checkBox, choice, Container, Label



Button:

Constructors: Button()
Button(String str)
void setLabel (String str)
~~void~~ String getLabel()

TextField (single line), TextArea (Multiline)

Flow Layout:

Constructors: FlowLayout(), FlowLayout(int how)
FlowLayout(int how, int hor, int vert)

Space b/w two adjacent components is 5px
FlowLayout.LEFT (constant)

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/* <applet code = FlowLayoutDemo width=250 height=200>
*/ </applet>
```

```
Public class FlowLayoutDemo extends Applet implements ItemListener
{
  String msg = " ";
  Checkbox winXP, winVista, Solaris, mac;
  public void init ()
  {
    setLayout ( new FlowLayout ( FlowLayout.LEFT)
    winXP = new checkbox (" Windows XP");
    winVista = new Checkbox (" Windows Vista");
    Solaris = new checkbox (" Solaris");
    mac = new checkbox (" Mac");
    add (winXP);
    add (winVista);
    add (Solaris);
    add (mac);
    winXP.addItemListener (this');
    winVista.add Item Listener (this);
    Solaris
  }
  public void ItemChanged ( ItemEvent e)
  {
    repeat();
  }
  public void paint (Graphics g)
  {
    msg = " current status";
    g.drawstring (msg, 6, 50);
    msg = "window xP ";
    winXP.getStatus();
    g.drawstring (msg, 6, 100);
  }
```

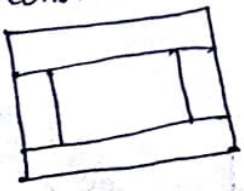
* Border Layout:

Display area is partitioned into 5.

Constructors: BorderLayout() , BorderLayout (int hori, int vert)

BorderLayout. CENTRE / . EAST / . WEST / . NORTH / . SOUTH

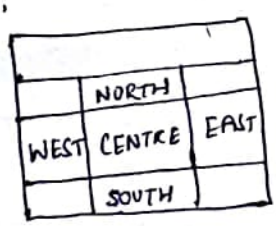
void add (Component compobj , object regron)



13/9/18

```
* import java.awt.*;
  import java.applet.*;
  import java.util.*;
  <applet code = "BorderDemo" width=400 height = 200> </applet>
public class BorderDemo extends Applet
{
public void init() {
 setLayout (new BorderLayout());
 add (new Button ("This is across the top"), BorderLayout.NORTH);
 add (new Label ("~~Border layout~~ The footer message"), BorderLayout
                                                          .SOUTH);
 ~~add (new Button~~
 add (new Button ("Right"), BorderLayout.EAST);
 add (new Button("Left"), BorderLayout.WEST);
 String msg = "
 add (new TextArea (msg), BorderLayout.CENTRE);
 }
}
```

Output:

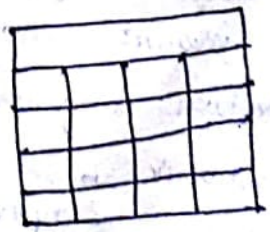


Grid Layout:

Constructors: GridLayout (), GridLayout (int numRows, int numcol)
GridLayout (int numRows, int numCol, int hor, int vert)

```
Public class GridLayoutDemo extends Applet
{
 static final int n=4;
 public void init()
 {
  setLayout (new GridLayout (n,n));
  setFont ("sans-", Font.BOLD, 24);
  for(int i=0; i<n; i++) {
  for(int j=0; j<n; j++) {
int k=i*n+j;
  if(k>0)
  add new Button(" "+k);
  }
 }
}
```

Output:



Card Layout:

There is a deck of cards with

Each card resembles a control. One control is at front and rest behind it. Clicking on any control brings it to the front.

Constructors: CardLayout()
CardLayout(int hor, int vert)

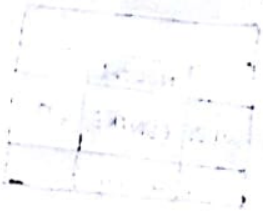
~~Method~~:
void add(Component panelobj, object name)

Methods: void first(Container deck)
void last(Container deck)
void next(Container deck) → Reference to container that holds the card
void previous(Container deck)
void show(Container deck)

Checkbox:

Constructors: Checkbox()
Checkbox(String str)
Checkbox(String str, boolean ob)
Checkbox(String str, boolean on, CheckboxGroup cbGroup)
Checkbox(String str, CheckboxComp cbComp, boolean on)

Methods:
boolean getStatus()
void setStatus(boolean on)
String getLabel()
void setLabel(String str)



*Difference blw awt and Swing:
i) AWT components are heavyweight while swing are light weight.
ii) AWT - platform independent, swing - platform in dependent.
iii) AWT - does not support pluggable look & feel while swing does.
iv) AWT provides lesser no. of controls as compared to swing.
v) AWT does not support MVC architecture, swing does

18/9/18

Networking:

javanet contains two major classes - ServerSocket and Socket.

Network program: A standalone program that runs on different machines.

Network: A set of interconnected collection of autonomous computers. (devices)

The no. of devices connected can range from two to internet. The connection can be wired or wireless.
Wired - UTP, STP, optical fibre etc
Wireless - different frequencies can be used to exchange data, eg. BT

Internet:
TCP/IP is a reference model containing a stack of protocols and four levels.

Network Programs:

Server Program: When it receives a request, it creats a thread.
To exchange data b/w client and server, I/P and o/P streams are created at both ends, and ~~they are conn.~~ these str. the I/P stream of client is connected to the o/P stream of server.

Socket Programming:

Socket is a communication end point. It contains IP address (device) and port. The IP address is of 32 bits and logical (can be changed). The IP address is assigned at the network layer.
Physical (MAC) address - does not change, 48 bits, hard wired.
For creating an object of communication end point at server side, invoke a constructor of ServerSocket class.
Suppose, four processes are active on server side. These addresses are distinguished by their port number. Each process is attached to a particular port.
Telnet uses port no. 21.
The standard applications below 1024 are well-known ports. These are User defined.

ServerSocket Class:

Constructors: (i) ServerSocket (int port) throws IO Exception
(ii) public ServerSocket (int port, int backlog) throws IOException
(iii) public ServerSocket (int port, int backlog, InetAddress address) throws IO Exception
(iv) public ServerSocket () throws IO Exception

(i) creates object and takes port no as argument, If port no. does not match, throws IO exception.
(ii) backlog indicates how many client requests can be put in the waiting queue
(iii) The IP address is taken from InetAddress object.
(iv) An unbound communication end is created.

Methods: (i) public int getLocalPort(): returns the port no. to which server is waiting to recieve the request
(ii) public Socket accept() throw IOException returns a reference to the socket object on which client prgrm can communicate.
Puts the request to wait indefinitely or until next method is called.
↳ setSoTimeout

(iii) public void setSoTimeout(int timeout) : unblocks the client request as soon as it is received

(iv) public void bind(SocketAddress host, int backlog) : If (iv) constructor is used, then bind method is used to get IP address and port no from host object and size of waiting ports from backlog object.

19/9/18

Socket class:

~~Methods:~~ Constructors:

(i) public Socket(String host, int port) throws UnknownHostException, IOException
(host → name of network host to which client is interested to connect)

(ii) public Socket(InetAddress host, int port) throws IOException

(iii) public Socket(String host, int port, InetAddress localAddress, int localport) throws IOException.
(localAddress → IP of client; localport → port no. of client)

(iv) public Socket(InetAddress host, int port, InetAddress localAddress, int localport)

(v) public Socket(): Creates unbound socket

Methods:

(i) public void connect(SocketAddress host, int timeout) throws IOException
(host → IP address & port no.; timeout → waiting time)

(ii) public int getPort(): returns appn attached to server side

(iii) public int getLocalPort(): return appn attached to client side

(iv) public InputStream getInputStream() throws IOException
returns input stream object

(v) public closed() throws IOException : delinks the IP and O/P streams

InetAddress class:

Method:

Static InetAddress getByAddress(byte[] address) : returns Inet address
(address → IP address in sequence of bytes)

* Client Program:

```
import java.net.*;
import java.io.*;
public class GeneratingClient
{
```

```
public static void main (String [] args)
{
  String servername = args[0];
  int port = Integer.parseInt (args[1]);
  try
  {
  Sop ("Connecting to " + servername + " on port " + port);
  Socket client = new Socket (servername, port);
  Sop("Just connecting to " + client.getRemoteSocketAddress());
     OutputStream outToServer = client.getOutputStream(~~outTo~~ server);
  DataOutputStream out = new DataOutputStream (outToServer);
  Out.writeUTF("Hello from " + client.getLocalSocketAddress());
  InputStream inFromServer = client.getInputStream();
  DataInputStream in = new DataInputStream (inFromServer);
  Sop("Server says" + in.readUTF);
  client.close();
  }
  Catch (IOException e) { }
}
}
```

* Server Socket Program

```
import java.net.*;
import java.io.*;
public class GeneratingServer extends Thread
{
Private ServerSocket serverSocket;
public GenerateServer (int port) throws IOException
 {
   serverSocket = new ServerSocket(port);
   serverSocket.setSoTimeout (10000);
 }
 public void run()
 {
   while (true)
   {
   try
   {
   Sop("Call for client o/p " + serverSocket.getLocalPort()+"....");
    Socket server = serverSocket.accept();
   Sop("Just connecting to "+ server.getRemoteSocketAddress();
```

20/9/18

Scanner Class:

Part of util package-
import java.util.*; import whole package
import java.util.Scanner.*; to import Scanner class only

IO system of java is stream based.

Stream → Byte
Stream → Character

Standard stream objects - System.in, System.out, System.err

Whitespace serves as a token separator.

To scan a single token:

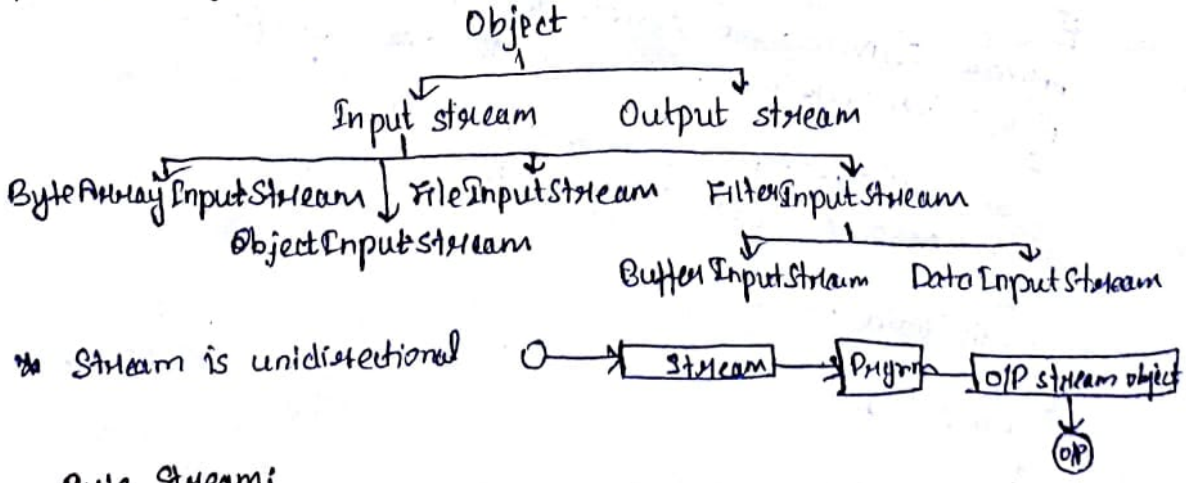
eg
```
Scanner x = new Scanner (System.in);
x.nextInt();
x.nextShort();
x.nextFloat();
x.nextLong();
x.nextDouble();
x.nextBoolean();
x.nextChar();
x.nextByte();
```

~~To scan multiple to~~

rand(): Random number generator

IO:

package: java.io.*;
The IO system of java is hierarchical in nature with object class at top.



* Stream is unidirectional

Byte Stream:

Character stream is unicode (16 bits) - Byte is 1 byte.

Classes: FileInputStream, FileOutputStream

* Program to copy data from I/P file F1 to O/P file F2.

```
import java.io.*;
public class CopyFile
```

```
{
    public static void main(--)
    {
      FileInputStream in = null;
      FileOutputStream out = null;
    try
    {
     in = new FileInputStream ("input.txt");
     out = new FileOutputStream ("output.txt");
     int c;
     while((c = in.read()) != -1)        (-1 => not eof)
      {
       out.write(c);
       }
       }
     finally
    {
     if (in != null)
    {
     in.close();
      }
     if (out != null)
    {
      out.close();
      }
    }
  }
}
```

Methods: close(), finalize(),
public int read(int a) throws IOException
public int read(byte [] a) throws IOException

Character Stream:

Classes: FileReader and FileWriter

5/10/2018

Swing:

Package: javax.swing

Any swing version of applet must extend JApplet.

Content panel Container getContentPane()
void add(comp)

Icon:

ImageIcon(String filename)
ImageIcon(URL url)
} Constructors of ImageIcon class

getIconHeight()
getIconWidth()
paintIcon(Component comp, Graphics g, int x, int y)

Label:

JLabel(Icon i)
JLabel(String s)
JLabel(Icon i, String s)

Methods:

- Icon getIcon()
- String getText()
  getIcon(
  setText(

```
import javax.swing.*;
/*<applet code = "JLabelDemo" width= 250 height =150></applet> */
public class JLabelDemo extends JApplet
{
  public void init()
  {
    Container contentPane = getContentPane();
    ImageIcon il = new ImageIcon("frame.gif");
    JLabel jl = new JLabel("Frame", il, JLabel.CENTER);
    contentPane.add(jl);
  }
}
```

TextFields:

Constructors:

JTextField()
JTextField(int col)
JTextField(String s, int col)
JTextField(String s)

Button:

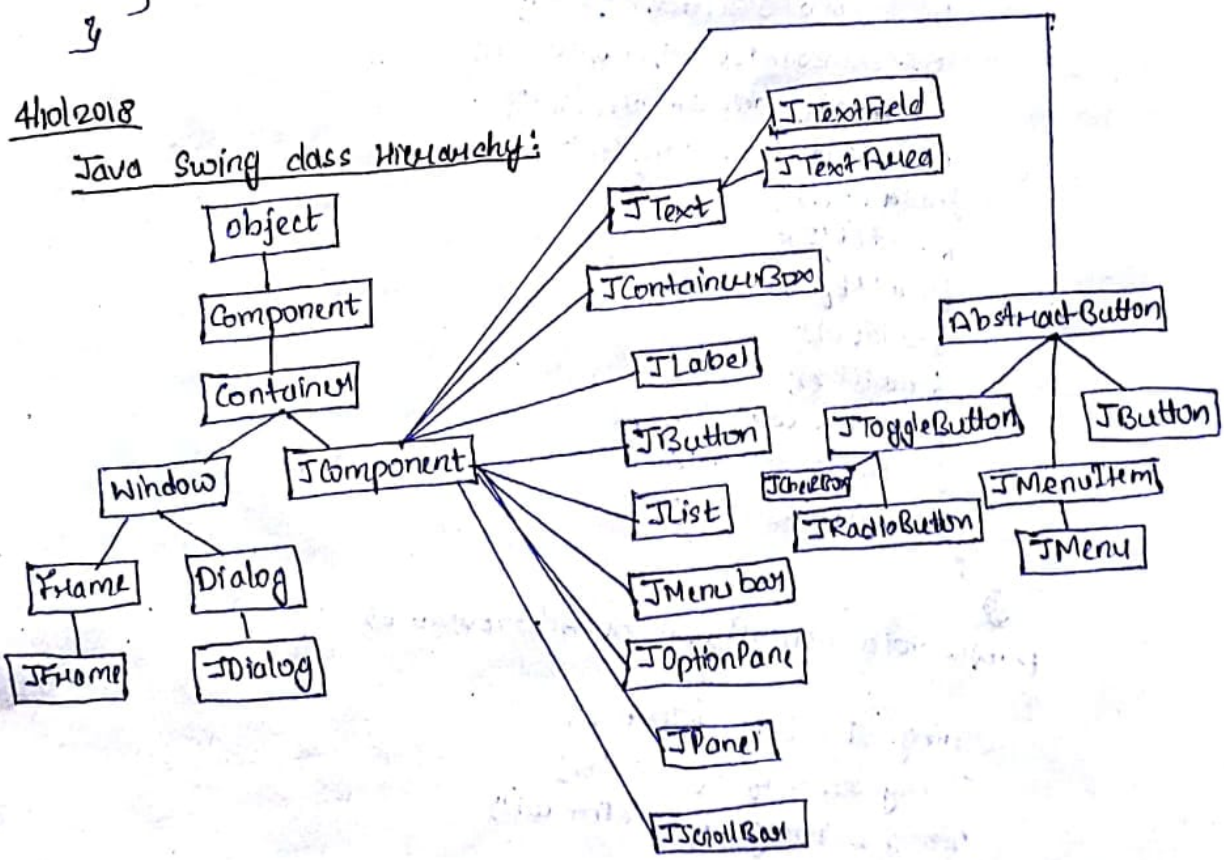
Constructors:

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

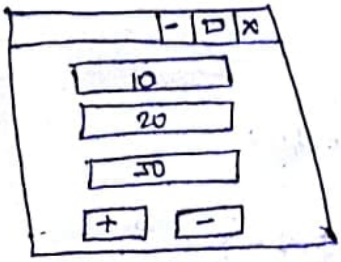
```
import javax.swing.*;
public class FirstSwing
{
public static void main
{
  JFrame f = new JFrame();
  JButton b = new Button ("click");
   b.setBounds (130, 100, 120, 90);
   f.add (b);
   f.setSize (400, 500);
   f.setLayout (null);
   f.setVisible (true);
}
}
```

4/10/2018

Java Swing class Hierarchy:



eg



```
import javax.swing.*;
import java.event.*;
public class TextFieldEx implements ActionListener
{
   JTextField tf1, tf2, tf3;
   JButton b1, b2;
   TextFieldEx()
   {
    JFrame f = new JFrame();
     tf1 = new JTextField();
     tf1 = SetBounds(50, 52, 150, 20);
     tf2 = new JTextField();
     tf2.setBounds(50, 100, 150, 20);
     tf3 = new JTextField();
      tf3.setBounds(50, 150, 150, 20);
       tf3.set  table(false);
        b1 = new JButton("+");
        b1.setBounds(50, 200, 50, 50);
        b2 = new JButton("-");
        b2.setBounds(120, 200, 50, 50);
        b1.Add ActionListener(this);
        b2.Add ActionListener(this);
       f.add(tf1);
       f.add(tf2);
       f.add(tf3);
       f.add(b1);
       f.add(b2);
       f.setSize(300, 300);
       f.setLayout(null);
       f.setVisible(true);
     }
   }
  Public void actionPerformed(ActionEvent e)
  {
     String s1 = tf1.getText();
      String s2 = tf2.getText();
      int a = Integer.parseInt(s1);
      int c = 0;
      if(e.getSource() == b1)
      c = a+b; else c = a-b;
        else
      String result = String.valueOf(c);
      tf3.setText(result); }
      Public static void main(String[] args) {
      new TextFieldEx(); } }
```

* Application to display IP address of a website when its url is taken as input.

Components:

JTextArea:

Constructors: JTextArea(String s)
JTextArea(int row, int col)
JTextArea(String s, int row, int col)

Methods:
void setRows(int n)
void setColumn(int c)
void setFont(Font f)

JPasswordField:
Specialised version of Java TextField, extends JTextField.

Constructors:
JPasswordField()
JPasswordField(String s)
JPasswordField(String s, int c)

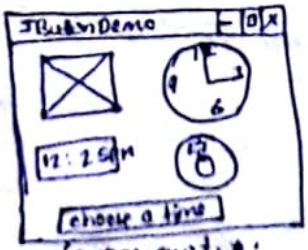
~~JComboBox~~: JCheckBox
It extends JToggleButton and implements Accessible interface

Constructors:
JComboBox()
JComboBox(String s)
JComboBox(String s, Boolean selected)

JCheckBox:
Extends JToggleButton, Implements Accessible interface

Constructors:
JCheckBox()
JCheckBox(String s)
JCheckBox(String s, boolean selected)

9/10/18

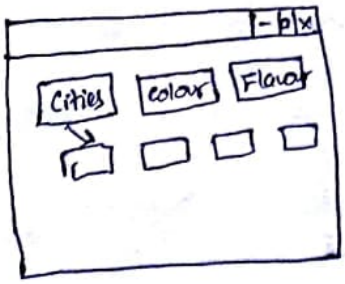



```
import javax.awt.*;
import java.awt.*;
import java.awt.event.*;
import java.swing.*;
public class JButtonDemo implements ActionListener
{
```

```
JLabel jlab;
public JButtonDemo()
{
 JFrame jfrm = new JFrame("JButtonDemo");
jfrm.setLayout(new FlowLayout());
jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JFrame.setSize(500, 450);
ImageIcon hourglass = new ImageIcon("hourglass.png");
JButton jb = new JButton(hourglass);
jb.setActionCommand("Hour Glass");
 jb.addActionListener(this);
 jfrm.add(jb);
 jlab = new JLabel("choose a Time ");
 jfrm.add(jlab);
  jfrm.setVisible(true);
}
public void actionPerformed(ActionEvent ae)
{
 jlab.setText("You selected " + ae.getActionCommand());
 }
public static void main(String[] args)
{
SwingUtilization invokeLater(new Runnable()
 public void run()
 {
```

JTabbedPane:

used to attach components with a tab.



1. Create an instance of JTabbedPane.
2. Add each tab by calling addTab()
3. Add the tabbed pane to the content pane.

```
public void makeGui()
{
 JTabbedPane jtp = new JTabbedPane();
```

```
        jtp.addTab("Cities", new CitiesPanel());
        jtp.addTab("Colour", new ColourPanel());
        jtp.addTab("Flavour", new FlavourPanel());
        add(jtp);
    }
class CitiesPanel extends JPanel
{
    public CitiesPanel()
    {
    JButton b1 = new JButton("New York");
    add(b1);
    JButton b2 = new JButton("London");
    add(b2);
    JButton b3 = new JButton("New Jersey");
    add(b3);
    JButton b4 = new JButton(
    add(b4);
    }
}
```

JScroll Pane:

1. Create the component to be scrolled.
2. Create an instance of JScroll Pane and pass to the object to scroll.
3. Add the scroll pane to the content pane.

```
private void makeGui()
{
    JPanel jp = new JPanel();
    jp.setLayout(new GridLayout(20,20));
    int b=0;
    for(int i=0; i<20; i++) {
    for(int j=0; j<20; j++) {
    jp.add(new JButton("Button "+ b); ++b; } }
    JScrollPane jsp = new JScrollPane(jp);
    add(jsp, BorderLayout.CENTER);
}
```

11/01/2018

~~TREE~~: JTree:

Constructors: JTree(Object obj[]) array
JTree(Vector <>V) Vector
JTree(TreeNode tn)

Display a set of data in a hierarchical order

* Array has homogeneous data type while vector has heterogeneous.

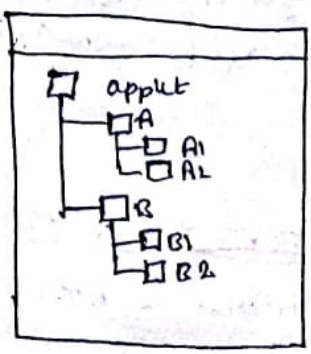
JTable defines two models:
i) Tree model
ii) TreeSelection model

Events:

TreeExpansionEvent : expansion / collapse of tree
TreeSelectionEvent : tree is selected
TreeModelEvent : generated when data or structure is changed.

1. Create an instance of JTree.
2. Create a ScrollPane.
3. Add the tree to the ScrollPane.
4. Add the ScrollPane to the content pane.

```
import java.awt.*;
import java.swing.event.*;
import javax.swing.*;
import java.swing.tree.*;
/* <applet code = "JTreeDemo" width= 400 height = 200> </applet> */
public class JTree Demo extends JApplet
{
  JTree tree;
  JLabel jlab;
  public void init()
  {
  try
  {
   SwingUtilities invokeWait()
   new Runnable ()
   {
    public void run()
    {
      makeGui();
    }
  }};
  catch (Exception exc)
  {
    Sop(—);
  }
}
```



```
private void makeGui()
{
  DefaultMutableTreeNode top = new DefaultMutableTreeNode("option");
  DefaultMutableTreeNode a = new DefaultMutableTreeNode("A");
  top.add(a);
  DefaultMutableTreeNode a1 = new DMTN("A1");
  a.add(a1);
  DMTN a2 = new DMTN("A2");
  a.add(a2);
  // create for B similarly
  tree = new JTree(top);
  JScrollPane jsp = new JScrollPane(tree);    // step 2
  add(jsp);
  jlab = new JLabel();
  add(jlab, BorderLayout.SOUTH);
  tree.addTreeSelectionListener(new TreeSelectionListener() {
  public void valueChanged(TreeSelectionEvent tse)
  {
    jlab.setText("selection is " + tree.getPath());
  }
  }
  }
}
```

JTable:

Constructor: JTable(Object data[][], Object colheading[])

```
private void makeGui()
{
  String[] colHead = {"Name", "Extension", "ID#"};
  object[][] data = {{"Graid", "4562", "555"}, {"ken", "7666", "855"}
  JTable table = new JTable(data, colHead);
  JScrollPane jsp = new JScrollPane(table);
  add(jsp);
  }
}
```

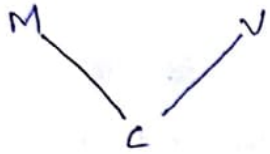
23/10/2018

1. WAP to find the frequency of character in string.
2. WAP to remove all whitespace from string.
3. WAP to lookup enum by string value.
4. WAP to create a list and add itemListener to it.

Servlet:

Server → Web Server (.war file)
→ Application server (.jar file)

MVC architecture:



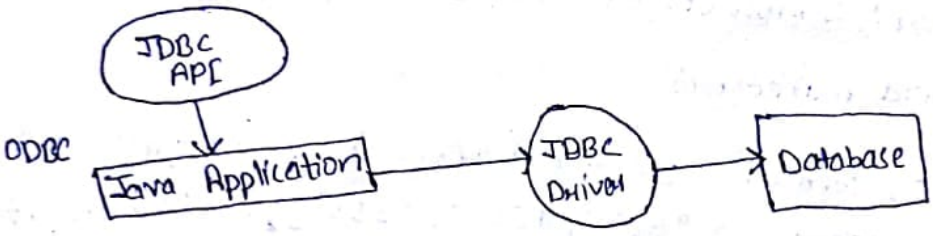
model
control

Servlet follows mvc architecture
It imports following packages:

```
import javax.Servlet
import javax.httpservlet
```

24/10/2018

1. JDBC (Java Database Connectivity):



Disadvantage of ODBC over JDBC:
ODBC is written in C whereas JDBC is written in Java.

JDK 5.0:

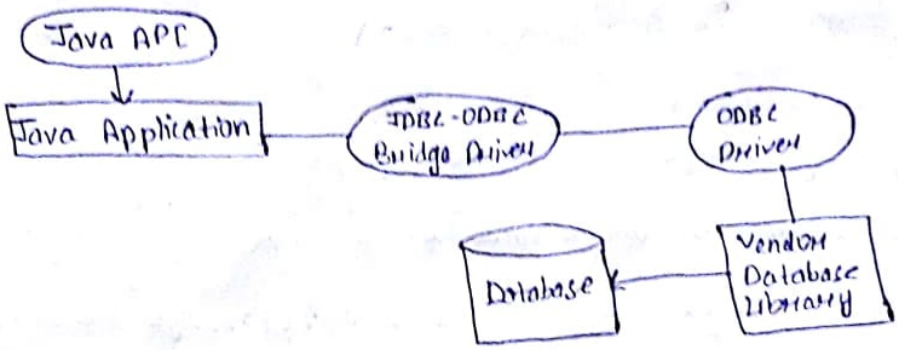
- JAVA_HOME
- CLASSPATH: used by Java compiler to point to compiled code
- PATH: used by OS

→ Environmental variables

Path to set JRE bin file:
PATH = C:\Program Files\Java\jre1.5.0-bin

JDBC Drivers:

JDBC-ODBC Bridge Driver



Advantage: Easy to use

Disadvantages:
1. Performance degradation
2. ODBC need to in client machine

2. Native API Driver (Type-II):

3. Network Protocol Driver
4. Thin Driver

Steps to Connect Java with Database:

i.) Register the driver class, method used → forName()
```
class.forName("Oracle.jdbc.driver.OracleDriver");
```
ii.) Creating connection
```
Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost
:1521:xe","system","password");
```
iii.) Create the statement object
```
Statement stmt = con.CreateStatement(); OracleDriver");
```
iv.) Execute Query
```
Resultset.rs = stmt.execute("select * from emp");
while(rs.next()){ "" + Sop(rs.getInt(1) + rs.getstop(2)); }
```
v.) close the connection object.
```
con.close();
```

25/10/2016

Connecting Java application to Oracle:

i.) Create a database.
java.sql, javax.sql available in j2se 1.5 onwards.

ii) setting the CLASSPATH

JAVA-HOME will point to the location of JRSE

CLASSPATH : c:\Program Files\Java\JDK 1.5.0-2.0\jre\Library

PATH : c:\Program Files\Java\jre 1.5.0-2.0\bin

append these strings

This will be permanent.

For temporary settings use command prompt.

Program to connect to oracle database:

Create table emp(id number(10), name varchar(40), age number(3));

```
import java.sql.*;
class Oracle Con
{
public static void main (String [] args)
{
try
{
class.forName ("Oracle.jdse.driver.OracleDriver");
Connection con = DriverManager.getConnection ("jdse:oracle:thin:@
localhost:1501:Xe", "System", "oracle");
Statement stmt = con.OracleStatement();
ResultSet rs = stmt.executeQuery ("select * from emp");
while (rs.next())
Sop (rs.getInt(1) + " " + rs.getString(2) + " " + rs.getString(3));
con.close();
}
catch (Exception e)
{
Sop(e);
}
}
}
```

ojdbc14.jar file must be loaded.

Two ways to load this file:

1) It is copied in jre\lib\ext

2) setting the CLASSPATH

After finishing this setting start the server.

Assignments:

1. WAP to connect mysql database in Java.
2. WAP to connect DB2 database in Java.

DriverManager:

It maintains a list of drivers.

Methods:

```
public static void registerDriver(Driver driver);
public static void deregisterDriver(Driver driver);
public static connection getConnection(String url);
```

Connection Interface:

Methods:

```
createStatement()
setAutoCommit()
commit()
rollback()
```

Statement Interface:

Methods:

```
executeQuery(String sql)
executeUpdate(String sql)
execute(String sql)
executeBatch(String sql)
```

ResultSet:

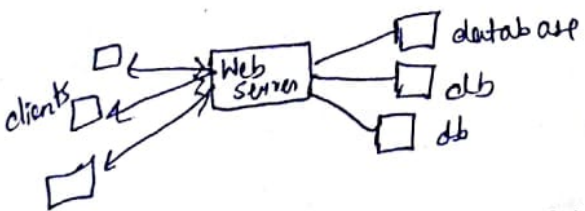
Methods:

```
next()
first()
last()
previous()
```

30/10/2018

Servlet:

Websites can be static or dynamic.
Static webpages show the same content on the browser. eg HTML
Dynamic webpages are created using technologies like servlets, JSP, ASP.NET, CGI, PHP



Packages:
javax.servlet
javax.servlet.http
Not part of traditional JDK. Requires Tomcat

* Creating dynamic web pages using CGI:
client writes url in the browser, the request goes to the server, where it is interpreted as CGI program, run and then sent back to the client after conversion in http form.

CGI causes overloading of memory as each program requires its own memory. ∴ Load on processor and high memory consumption.

Create Servlet development environment:

1. Install Tomcat and save in program directory
   C:\Program Files\Apache\Tomcat-4.0\
2. Set path JAVA_HOME

Start Tomcat:

Command Prompt →

3. Set the classpath C:\Program File\Apache Tomcat 4.0\Library\servlet.jar

How to write a simple servlet:

1. Create and compile the servlet source code.
2. Start Tomcat.
3. Start a web browser and request for the servlet.

```
import java.io.*;
import javax.servlet.*;
public class HelloServlet extends GenericServlet
{
  public void service(ServletRequest request, ServletResponse response)
  throws ServletException, IOException
  {
    response.contentType("Text/html");
    PrintWriter pw = response.getWriter();
    pw.println("<B>Hello");
    pw.close();
  }
}
```

Classes and Interfaces present in the two packages:

1. java.servlet.*;

Servlet API:

Interfaces:
- Servlet
- ServletConfig
- ServletContext
- ServletRequest
- ServletResponse

Classes:
- GenericServlet
- ServletInputStream
- ServletOutputStream
- ServletException
- UnavailableException

Methods:

void destroy()
ServletConfig getServletConfig()
Stop getServletInfo()
void init (ServletConfig sc) throws ServletException
void service (ServletRequest req, ServletResponse resp) throws ServletException, IOException

30/10/2018

Servlet Life Cycle:

1. Load servlet class
2. Create service instance.
3. Call init()
4. Call the service()
5. Destroy

Http is connectionless and stateless → same info cannot be fetched twice.
↓
after sending request, the client disconnects.

1,2,3 are done only once in the entire life-cycle.

public void init (ServletConfig config) throws ServletException
public void service (ServiceRequest rqst, ServiceResponse rsp) throws ServiceException, IOException

Ways to create Servlet:

i) By implementing Servlet interface.
ii) By ~~implementi~~ inheriting Generic Servlet class
iii) By inheriting HttpServlet class

2. javax.servlet.http.*;

Interfaces:

HttpServletRequest : enables servlet program to read data from http
HHpServletResponse
Httpsession

Classes:

Cookie
Httpservlet
Http SessionF

Http Post Request:

Code to select a colour and print the selected colour:

```
<html>
<body>
<center>
<form name = "Form1" method = "post" action = "http://localhost:
8080/servlet examples/servlet/ColourPostServlet" >
<B> Colour : </B>
<select name = "Colour" size = "1" >
```

```
<option value = "Red" > Red </option>
<option value = "Green" > Green </option>
</select>
<br><br>
<input type = submit value = "submit" >
</form>
</body>
</html>
```

```
import java.io.*;
import javax.servlet.*;
import javax.httpservlet.*;
public class ColourPostServlet extends HttpServlet
{
 public void doPost (HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException
 {
  String colour = request.getParameter("colour");
  response.setContentType("text/html");
  PrintWriter pw = response.getWriter();
  pw.println("<B>The selected colour is :");
  pw.println(colour);
  pw.close();
 }
}
```