Module 3

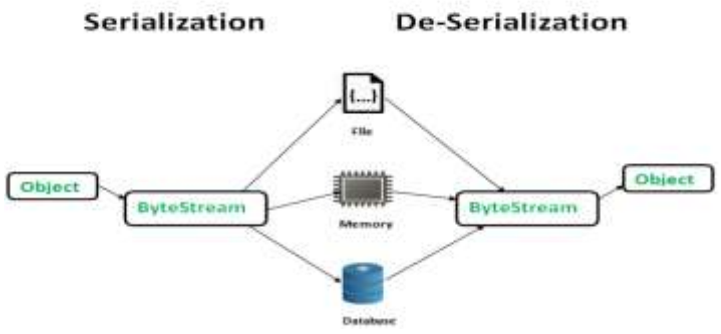
Serialization and Deserialization in Java with Example

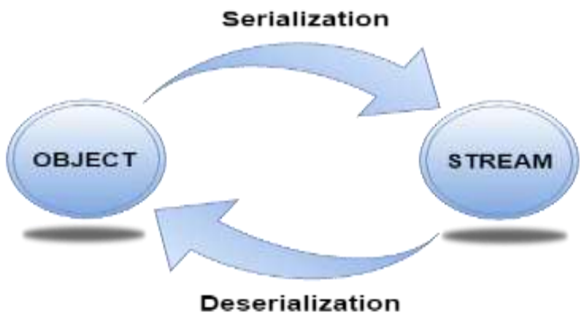
Serialization is a mechanism of converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.

The byte stream created is platform independent. So, the object serialized on one platform can be deserialized on a different platform.

To make a Java object serializable we implement the **java.io.Serializable** interface.

For serializing the object, we call the **writeObject()** method of *ObjectOutputStream* and for deserialization we call the **readObject()** method of *ObjectInputStream* class





Serializable is a marker interface (has no data member and method). It is used to "mark" Java classes so that the objects of these classes may get a certain capability. The Cloneable and Remote are also marker interfaces. It must be implemented by the class whose object you want to persist.
The String class and all the wrapper classes implement the *java.io.Serializable* interface by default.

ObjectOutputStream class

The **ObjectOutputStream** class is used to write primitive data types, and Java objects to an OutputStream. Only objects that support the java.io.Serializable interface can be written to streams.

ObjectInputStream class

An ObjectInputStream deserializes objects and primitive data written using an ObjectOutputStream.

For a class to be serialized successfully, two conditions must be met −

- The class must implement the java.io.Serializable interface.
- All of the fields in the class must be serializable. If a field is not serializable, it must be marked **transient**.

Advantages of Serialization in Java

1. This is a built-in feature of Java. Hence you need not use third-party services to implement Serialization.
2. This concept is easy to understand.
3. It is customizable as per the programmer’s needs.
4. This process is universal and all kinds of developers are familiar with the concept.
5. This allows Java to perform Encryption, Authentication, and Compression and secure Java computing.
6. Most of the technologies we use daily, rely on serialization.

Disadvantages of Serialization in Java

1. Sometimes the byte streams do not convert into objects completely which leads to errors.
2. Whenever you declare a variable as transient, the compiler assigns a memory space to it. But the constructor of the class remains uncalled. This results in a variation of **Java Standard Flow**.
3. This process is inefficient when it comes to memory utilization.
4. Serialization is not useful in applications that need concurrent access without using third party APIs. This is because serialization does not offer any transition control mechanism.
5. It does not allow fine control when accessing objects.

Example:

```
import java.io.*;

public class Bike implements Serializable
{
public String name;
public int model;
public int cost;

public Bike(String name, int model, int cost)
{
this.name=name;
this.model=model;
this.cost=cost;
}
}



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

//create object
Bike b = new Bike("Pulsar",1982,234);
//create a txt file
FileOutputStream fout = new FileOutputStream("E:\\bike.txt");

//create an object stream
ObjectOutputStream out = new ObjectOutputStream(fout);

//write the object stream onto the txt file
out.writeObject(b);

System.out.println(" Serialization happened ");
fout.close();
out.close();
 }
}

public class DSerialDemo
{
public static void main(String args[])
{
//create a txt file
FileInputStream fin = new FileInputStream("E:\\bike.txt");

//create an object stream
ObjectInputStream in = new ObjectInputStream(fin);

//read the txt file (Byte stream)
Bike b = (Bike)in.readObject();
System.out.println("Name: "+b.name);
System.out.println("Model: "+b.model);
System.out.println("Cost: "+b.cost);

System.out.println(" DeSerialization happened ");
fin.close();
in.close();

 }
}
```

In case of **transient variables:-** A variable defined with transient keyword is not serialized during serialization process. This variable will be initialized with default value during deserialization. (e.g: for objects it is null, for int it is 0).

In case of **static Variables:-** A variable defined with static keyword is not serialized during serialization process. This variable will be loaded with current value defined in the class during deserialization.

Externalization in Java

Externalization in Java is used to customize the serialization mechanism. Java serialization is not much efficient. When we have bloated objects that hold several attributes and properties, it is not good to serialize them. Here, the externalization will be more efficient.

Java Serialization with Inheritance (IS-A Relationship)

If a class implements serializable then all its sub classes will also be serializable.

Example:

```
import java.io.*;

public class Bike implements Serializable
{
public String name;
public int model;
public int cost;

public Bike(String name, int model, int cost)
{
this.name=name;
this.model=model;
this.cost=cost;
}
}

public class OffRoad extends Bike
{
```

```
public int capacity;
public OffRoad(String name, int model, int cost, int capacity)
{
super(name,model,cost);
this.capacity = capacity;
}

public void disp()
{
super.disp();
System.out.println(" capacity : " +capacity);
}
}

public class IsASerial
{
public static void main(String args[]) throws Exception
{

//create object
OffRoad or = new OffRoad("Pulsar",1982,234,456);

//create a txt file
FileOutputStream fout = new FileOutputStream("E:\\orbike.txt");

//create an object stream
ObjectOutputStream out = new ObjectOutputStream(fout);

//write the object stream onto the txt file
out.writeObject(or);

System.out.println(" IS_A Serialization happened ");
fout.close();
out.close();
}
}

public class IsADSerial
{
public static void main(String args[]) throws Exception
```

```
{
//create a txt file
FileInputStream fin = new FileInputStream("E:\\orbike.txt");

//create an object stream
ObjectInputStream in = new ObjectInputStream(fin);

//read the txt file (Byte stream)
OffRoad or = (OffRoad)in.readObject();

System.out.println(" IS_A DeSerialization happened ");
or.disp();
fin.close();
in.close();
}
}
```

Java Serialization with Aggregation (HAS-A Relationship)

If a class has a reference to another class, all the references must be Serializable otherwise serialization process will not be performed.

Example:

```
import java.io.*;

public class HasABike implements Serializable
{
public String category;
Bike b = new Bike("Java",2012,123);     //"has a" relationship

public HasABike(String category)
{
this.category = category;
}

public void disp()
{
b.disp();
```

```
System.out.println(" category : " +category);
}
}

public class HasASerial
{
public static void main(String args[])throws Exception
{

//create object
HasABike hb = new HasABike("no - pillion ");

//create a txt file
FileOutputStream fout = new FileOutputStream("E:\\hbbike.txt");

//create an object stream
ObjectOutputStream out = new ObjectOutputStream(fout);

//write the object stream onto the txt file
out.writeObject(hb);

System.out.println(" Has a Serialization happened ");
fout.close();
out.close();

}
}

public class HasADSerial
{
public static void main(String args[])throws Exception
{

//create a txt file
FileInputStream fin = new FileInputStream("E:\\hbbike.txt");

//create an object stream
ObjectInputStream in = new ObjectInputStream(fin);

//read the txt file (Byte stream)
```

```
HasABike hb = (HasABike)in.readObject();

System.out.println(" Has-A DeSerialization happened ");
hb.disp();
fin.close();
in.close();
}
}
```

IS-A Re1lationship:

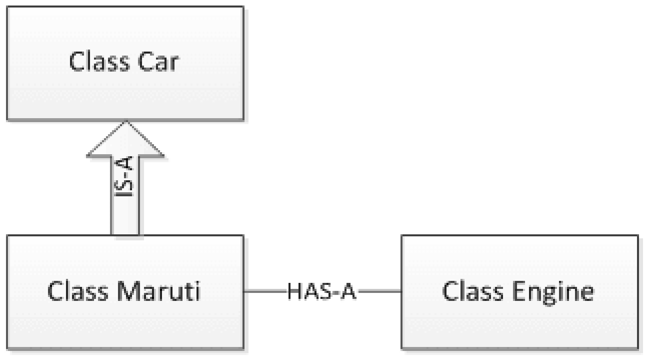
In object-oriented programming, the concept of IS-A is a totally based on Inheritance, which can be of two types Class Inheritance or Interface Inheritance. It is just like saying "A is a B type of thing". For example, Apple is a Fruit, Car is a Vehicle etc. Inheritance is uni-directional. For example, House is a Building. But Building is not a House.

It is a key point to note that you can easily identify the IS-A relationship. Wherever you see an extends keyword or implements keyword in a class declaration, then this class is said to have IS-A relationship.

HAS-A Relationship:

Composition(HAS-A) simply mean the use of instance variables that are references to other objects. For example Maruti has Engine, or House has Bathroom.

Let’s understand these concepts with an example of Car class.

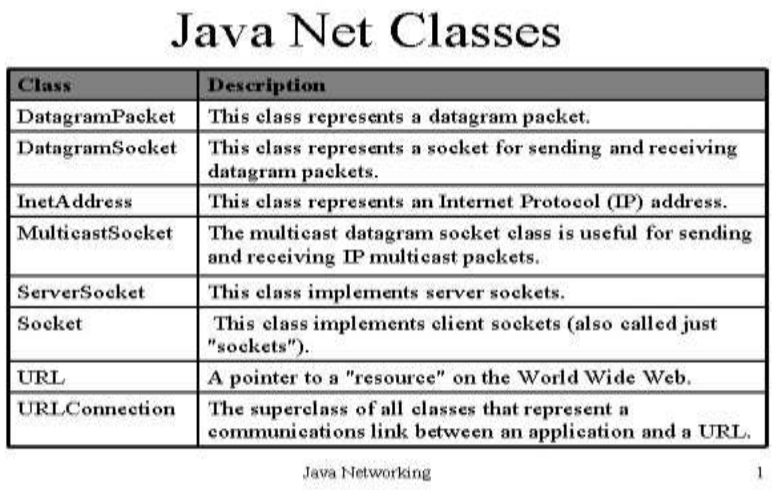


Transient

transient is a variables modifier used in serialization. At the time of serialization, if we don’t want to save value of a particular variable in a file, then we use **transient** keyword. When JVM comes across **transient** keyword, it ignores original value of the variable and save default value of that variable data type.

```
import java.io.*;
class Student implements Serializable{
int id;
String name;
transient int age;//Now it will not be serialized
public Student(int id, String name,int age) {
this.id = id;
this.name = name;
this.age=age;
 }
}
class MSerT{
public static void main(String args[])throws Exception{
Student s1 =new Student (211,"ravi",22);
FileOutputStream f=new FileOutputStream("f.txt");
ObjectOutputStream out=new ObjectOutputStream(f);
out.writeObject(s1);
out.flush();
out.close();
f.close();
System.out.println("success");
 }
}

class MDSerT{
public static void main(String args[])throws Exception{
ObjectInputStream in=new ObjectInputStream(new FileInputStream("f.txt"));
Student s=(Student)in.readObject();
System.out.println(s.id+" "+s.name+" "+s.age);
in.close();
 }
}
```

Networking



Java Net Classes

Class	Description
DatagramPacket	This class represents a datagram packet.
DatagramSocket	This class represents a socket for sending and receiving datagram packets.
InetAddress	This class represents an Internet Protocol (IP) address.
MulticastSocket	The multicast datagram socket class is useful for sending and receiving IP multicast packets.
ServerSocket	This class implements server sockets.
Socket	This class implements client sockets (also called just "sockets").
URL	A pointer to a "resource" on the World Wide Web.
URLConnection	The superclass of all classes that represent a communications link between an application and a URL.

Java Networking 1

The term ***network programming*** refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network.

The java.net package of the J2SE APIs contains a collection of classes and interfaces that provide the low-level communication details, allowing you to write programs that focus on solving the problem at hand.

The java.net package provides support for the two common network protocols −

- **TCP** − TCP stands for *Transmission Control Protocol*, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP. It ensures that the connection is established and maintained until the data packet is transferring between the sender and receiver is complete.
- **UDP** − UDP stands for *User Datagram Protocol*, a connection-less protocol that allows for packets of data to be transmitted between applications. A UDP does not ensure to deliver the data packets to the correct destination, and it does not generate any acknowledgment about the sender's data. Similarly, it does not acknowledge the receiver about the data. Hence, it is an unreliable protocol.

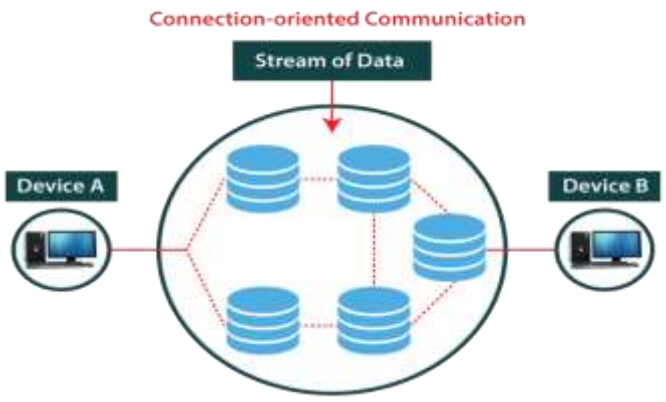
Connection(Alice and Bob)

- An endpoint(telephone) for communication is created on both ends
- An address(phone number) is assigned to both ends to distinguish them from the rest of the network
- One of the end points(caller) initiates a connection to the other
- The other end(receiver) point waits for the communication to start
- Once a connection has been made, data is exchanged
- Once data has been exchanged, the endpoints are closed.

There are two ways to establish a connection before sending data from one device to another, that are **Connection-Oriented** and **Connectionless Service**. Connection-oriented service involves the creation and termination of the connection for sending the data between two or more devices. In contrast, connectionless service does not require establishing any connection and termination process for transferring the data over a network.

Connection-Oriented Service

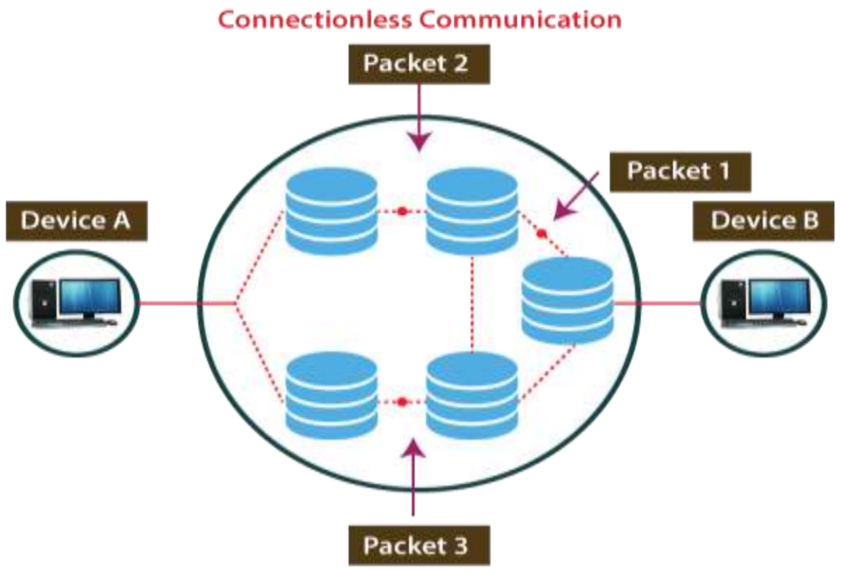
A connection-oriented service is a network service that was designed and developed after the telephone system. A connection-oriented service is used to create an end to end connection between the sender and the receiver before transmitting the data over the same or different networks. In connection-oriented service, packets are transmitted to the receiver in the same order the sender has sent them. It uses a handshake method that creates a connection between the user and sender for transmitting the data over the network. Hence it is also known as a reliable network service.



Suppose, a sender wants to send data to the receiver. Then, first, the sender sends a request packet to a receiver in the form of an **SYN** packet. After that, the receiver responds to the sender's request with an (SYN-ACK) signal/packets. That represents the confirmation is received by the receiver to start the communication between the sender and the receiver. Now a sender can send the message or data to the receiver.

Connectionless Service

A connection is similar to a **postal system**, in which each letter takes along different route paths from the source to the destination address. Connectionless service is used in the network system to transfer data from one end to another end without creating any connection. So it does not require establishing a connection before sending the data from the sender to the receiver. It is not a reliable network service because it does not guarantee the transfer of data packets to the receiver, and data packets can be received in any order to the receiver. Therefore we can say that the data packet does not follow a **defined** path. In connectionless service, the transmitted data packet is not received by the receiver due to network congestion, and the data may be lost.



For example, a sender can directly send any data to the receiver without establishing any connection because it is a connectionless service. Data sent by the sender will be in the packet or data streams containing the receiver's address. In connectionless service, the data can be travelled and received in any order. However, it does not guarantee to transfer of the packets to the right destination.

Difference between Connection-oriented and Connection-less Services:

S.NO	Connection-oriented Service	Connection-less Service
1.	Connection-oriented service is related to the telephone system.	Connection-less service is related to the postal system.
2.	Connection-oriented service is preferred by long and steady communication.	Connection-less Service is preferred by bursty communication.
3.	Connection-oriented Service is necessary.	Connection-less Service is not compulsory.
4.	Connection-oriented Service is feasible.	Connection-less Service is not feasible.
5.	In connection-oriented Service, Congestion is not possible.	In connection-less Service, Congestion is possible.
6.	Connection-oriented Service gives the guarantee of reliability.	Connection-less Service does not give the guarantee of reliability.
7.	In connection-oriented Service, Packets follow the same route.	In connection-less Service, Packets do not follow the same route.
8.	Connection-oriented Services requires a bandwidth of high range.	Connection-less Service requires a bandwidth of low range.

Socket Programming

Sockets is an endpoint of a two-way communication link between programs running on the network using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server.

When the connection is made, the server creates a socket object on its end of the communication. The client and the server can now communicate by writing to and reading from the socket.

The java.net.Socket class represents a socket, and the java.net.ServerSocket class provides a mechanism for the server proto sergram to listen for clients and establish connections with them.

Sockets allow communication between two different processes on the same or different machines.

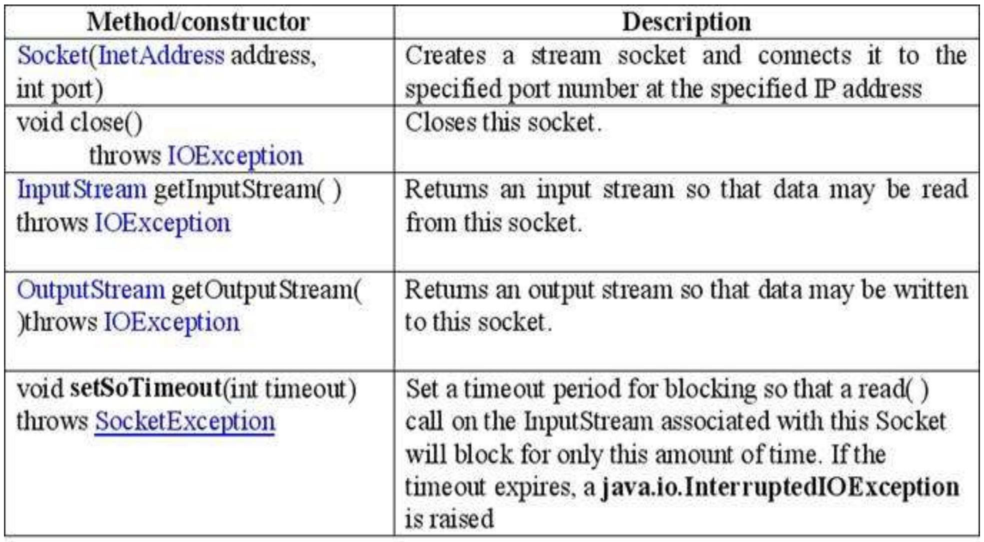
To create Connection

- Client socket sends a request to server socket.
- If server socket accepts it and catches the incoming socket the connection is established.
- Data can be transferred via the input and output streams of sockets.

The following steps occur when establishing a TCP connection between two computers using sockets −

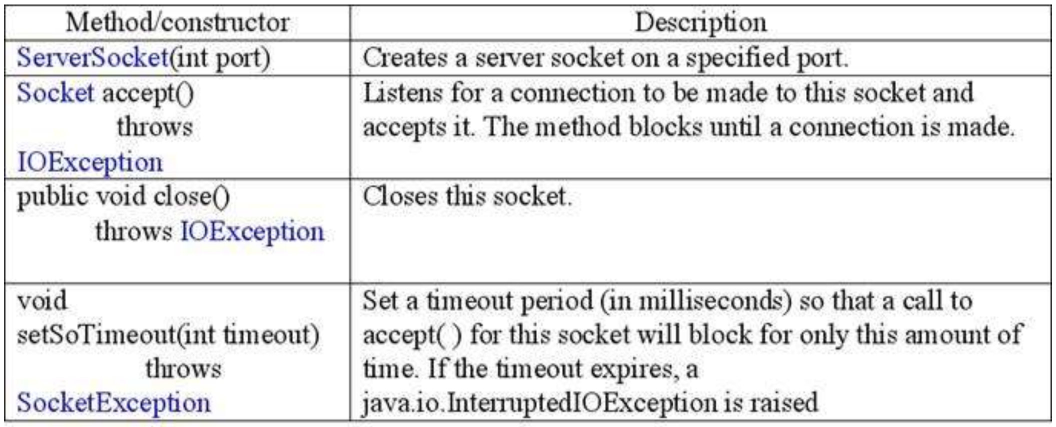
- The server instantiates a ServerSocket object, denoting which port number communication is to occur on.
- The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port.
- After the server is waiting, a client instantiates a Socket object, specifying the server name and the port number to connect to.
- The constructor of the Socket class attempts to connect the client to the specified server and the port number. If communication is established, the client now has a Socket object capable of communicating with the server.
- On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket.

Server Methods



Method/constructor	Description
Socket(InetAddress address, int port)	Creates a stream socket and connects it to the specified port number at the specified IP address
void close() throws IOException	Closes this socket.
InputStream getInputStream( ) throws IOException	Returns an input stream so that data may be read from this socket.
OutputStream getOutputStream( )throws IOException	Returns an output stream so that data may be written to this socket.
void **setSoTimeout**(int timeout) throws SocketException	Set a timeout period for blocking so that a read( ) call on the InputStream associated with this Socket will block for only this amount of time. If the timeout expires, a **java.io.InterruptedIOException** is raised

Server Socket methods



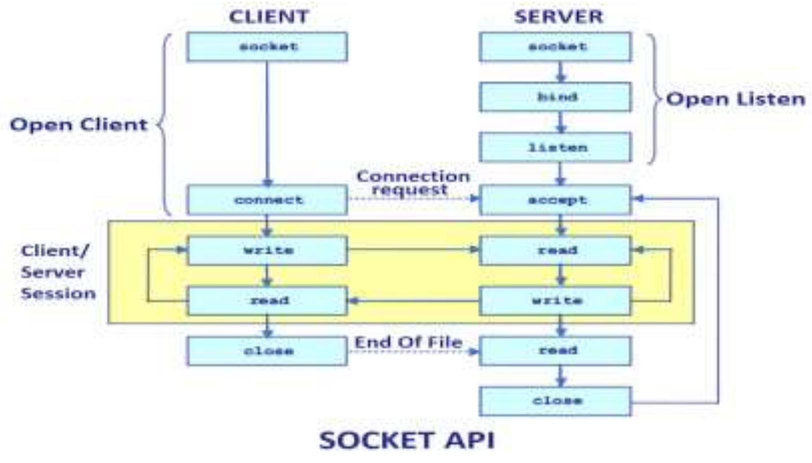
Method/constructor	Description
ServerSocket(int port)	Creates a server socket on a specified port.
Socket accept() throws IOException	Listens for a connection to be made to this socket and accepts it. The method blocks until a connection is made.
public void close() throws IOException	Closes this socket.
void setSoTimeout(int timeout) throws SocketException	Set a timeout period (in milliseconds) so that a call to accept( ) for this socket will block for only this amount of time. If the timeout expires, a java.io.InterruptedIOException is raised

Port

A port number is a unique identifier used with an IP address. A port is a 16-bit unsigned integer, and the total number of ports available in the TCP/IP model is 65,535 ports. Therefore, the range of port numbers is 0 to 65535.

Why do we require port numbers?

A single client can have multiple connections with the same server or multiple servers. The client may be running multiple applications at the same time. When the client tries to access some service, then the IP address is not sufficient to access the service. To access the service from a server, the port number is required. So, the transport layer plays a major role in providing multiple communication between these applications by assigning a port number to the applications.



Client server implementation

Creating Server:

To create the server application, we need to create the instance of ServerSocket class. Here, we are using 6666 port number for the communication between the client and server. You may also choose any other port number. The accept() method waits for the client. If clients connects with the given port number, it returns an instance of Socket.

Creating Client:

To create the client application, we need to create the instance of Socket class. Here, we need to pass the IP address or hostname of the Server and a port number. Here, we are using "localhost" because our server is running on same system.

Program 1:

Client.java

```
import java.io.*;
import java.net.*;
public class MyClient {
public static void main(String[] args) throws Exception{
Socket s=new Socket("localhost",6666);
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
DataInputStream din=new DataInputStream(System.in);
System.out.println("Enter the message to the server ");
String str = din.readLine();
dout.writeUTF(str);
dout.close();
s.close();
}
}
```

Server.java

```
import java.io.*;
import java.net.*;
public class MyServer {
public static void main(String[] args) throws Exception{
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();//establishes connection
DataInputStream dis=new DataInputStream(s.getInputStream());
String  str=(String)dis.readUTF();
System.out.println("message= "+str);
}
}
```

Program 2:

Client.java

```
import java.io.*;
import java.net.*;
public class SocketClient {
public static void main(String[] args) throws Exception{
Socket s=new Socket("localhost",6666);
DataInputStream kin=new DataInputStream(System.in);
DataInputStream sin=new DataInputStream(s.getInputStream());
DataOutputStream sout=new DataOutputStream(s.getOutputStream());
String str;
while(true){
str = sin.readUTF();
If(str.equals(“bye”)) break;
System.out.println("Server says  " +str);
System.out.println(“Enter data/bye for the server”);
str = kin.readLine();
sout.writeUTF(str);
if(str.equals(“bye”)) break;
}
}
}
```

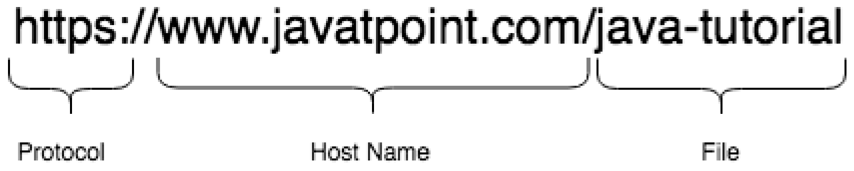
Server.java

```
import java.io.*;
import java.net.*;
public class SocketServer {
public static void main(String[] args) throws Exception{
ServerSocket ss=new ServerSocket(456);
Socket s=ss.accept();//establishes connection
DataInputStream din=new DataInputStream(System.in);
DataInputStream sin=new DataInputStream(s.getInputStream());
DataOutputStream sout=new DataOutputStream(s.getOutputStream());
string  str;
sout.writeUTF("Welcome");
while(true){
str = sin.readUTF();
system.out.println("Client says "+str);
if(str.equals("bye"))  break;
system.out.println("Enter data/bye for the client");
str = din.readLine();
sout.writeUTF(str);
if(str.equals("bye")) break;
ss.close();
}
}
}
```

Java URL

The **Java URL** class represents an URL. URL is an acronym for ***Uniform Resource Locator***. It points to a resource on the World Wide Web. For example:

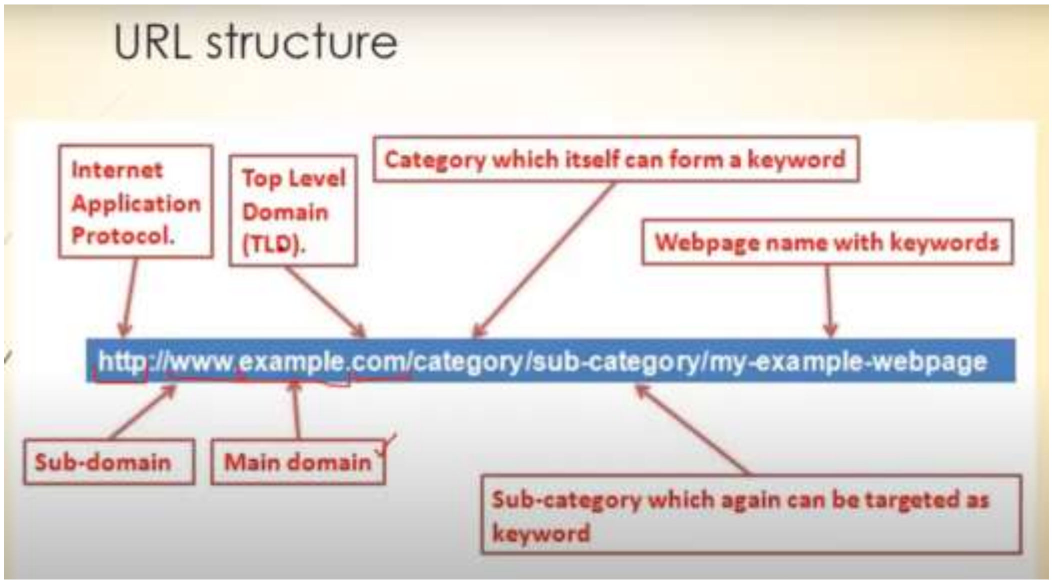
1. https://www.javatpoint.com/java-tutorial



A URL contains many information:

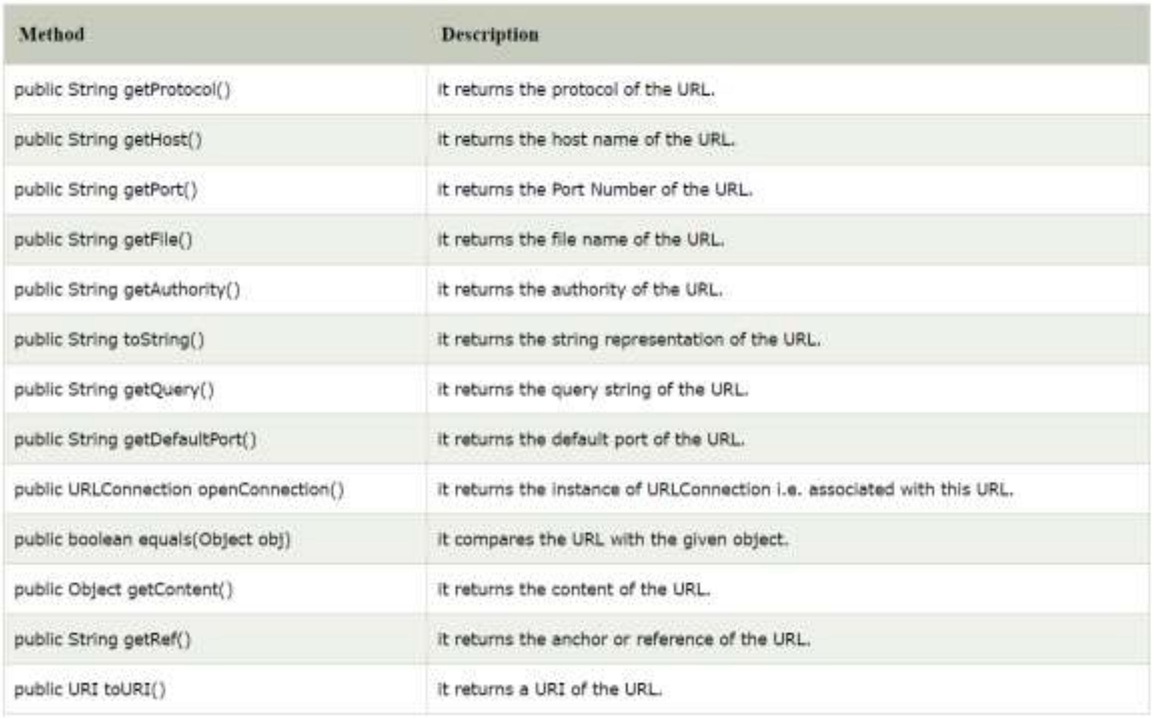
1. **Protocol:** In this case, http is the protocol.
2. **Server name or IP Address:** In this case, www.javatpoint.com is the server name.
3. **Port Number:** It is an optional attribute. If we write http//ww.javatpoint.com:80/sonoojaiswal/ , 80 is the port number. If port number is not mentioned in the URL, it returns -1.

4. **File Name or directory name:** In this case, index.jsp is the file name.


URL structure
Internet Application Protocol.
Top Level Domain (TLD).
Category which itself can form a keyword
Webpage name with keywords
http://www.example.com/category/sub-category/my-example-webpage
Sub-domain
Main domain
Sub-category which again can be targeted as keyword

URL methods

The java.net.URL class provides many methods.



Method	Description
public String getProtocol()	it returns the protocol of the URL.
public String getHost()	it returns the host name of the URL.
public String getPort()	it returns the Port Number of the URL.
public String getFile()	it returns the file name of the URL.
public String getAuthority()	it returns the authority of the URL.
public String toString()	it returns the string representation of the URL.
public String getQuery()	it returns the query string of the URL.
public String getDefaultPort()	it returns the default port of the URL.
public URLConnection openConnection()	it returns the instance of URLConnection i.e. associated with this URL.
public boolean equals(Object obj)	it compares the URL with the given object.
public Object getContent()	it returns the content of the URL.
public String getRef()	it returns the anchor or reference of the URL.
public URI toURI()	it returns a URI of the URL.

Program:

```
import java.net.*;
public class URLDemo{
public static void main(String[] args){
try{
URL url=new URL("https://www.google.com/search?q=javatpoint&oq=javatp
oint&sourceid=chrome&ie=UTF-8");

System.out.println("Protocol: "+url.getProtocol());
System.out.println("Host Name: "+url.getHost());
System.out.println("Port Number: "+url.getPort());
System.out.println("Default Port Number: "+url.getDefaultPort());
System.out.println("Query String: "+url.getQuery());
System.out.println("Path: "+url.getPath());
System.out.println("File: "+url.getFile());

}catch(Exception e){System.out.println(e);}
}
}
```

Output:

```
Protocol: https
Host Name: www.google.com
Port Number: -1
Default Port Number: 443
Query String: q=javatpoint&oq=javatpoint&sourceid=chrome&ie=UTF-8
Path: /search
File: /search?q=javatpoint&oq=javatpoint&sourceid=chrome&ie=UTF-8
```

Java URLConnection class

The **Java URLConnection** class represents a communication link between the URL and the application. This class can be used to read and write data to the specified resource referred by the URL.

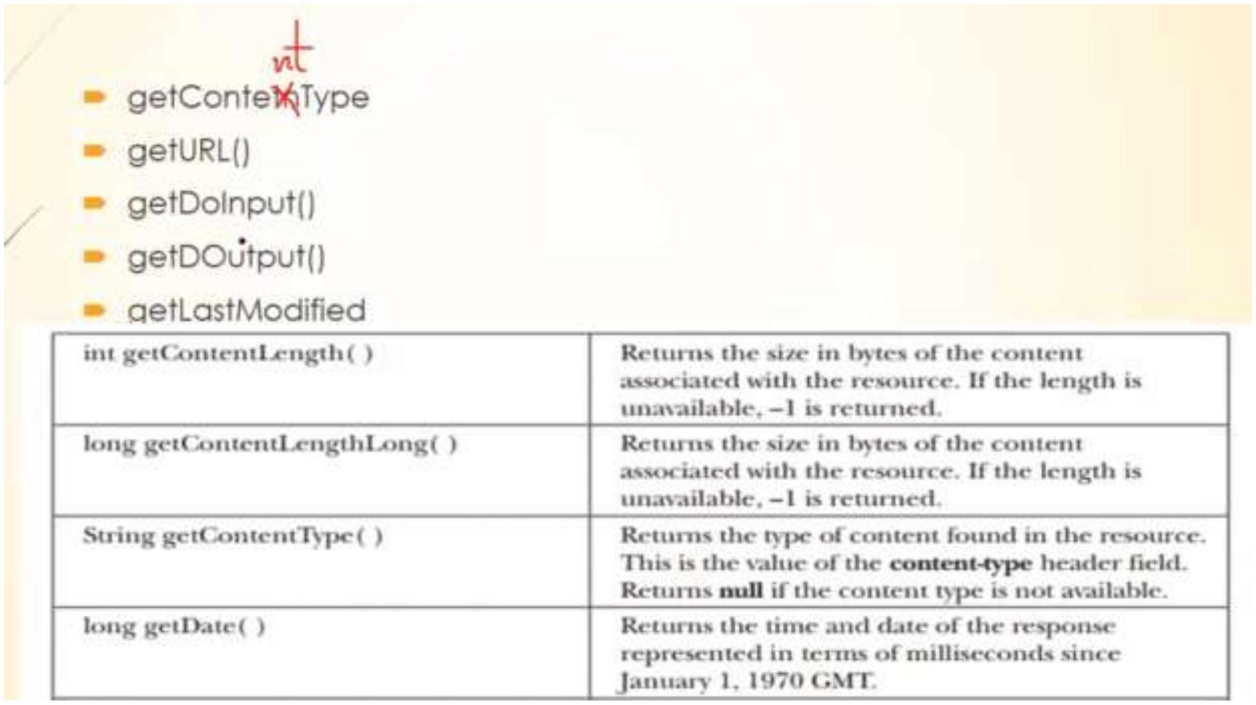
How to get the object of URLConnection class

The openConnection() method of URL class returns the object of URLConnection class. Syntax:

1. **public** URLConnection openConnection()**throws** IOException{}

The URLConnection class provides many methods, we can display all the data of a webpage by using the getInputStream() method. The getInputStream() method returns all the data of the specified URL in the stream that can be read and displayed.

URL Connection class methods



- getConte~~x~~ntType
- getURL()
- getDoInput()
- getDOutput()
- getLastModified

int getContentLength( )	Returns the size in bytes of the content associated with the resource. If the length is unavailable, −1 is returned.
long getContentLengthLong( )	Returns the size in bytes of the content associated with the resource. If the length is unavailable, −1 is returned.
String getContentType( )	Returns the type of content found in the resource. This is the value of the **content-type** header field. Returns **null** if the content type is not available.
long getDate( )	Returns the time and date of the response represented in terms of milliseconds since January 1, 1970 GMT.

Program 1:

```
import java.io.*;
import java.net.*;
public class URLConDemo {
public static void main(String[] args) throws Exception{
URL url=new URL("http://www.javatpoint.com/java-tutorial");
URLConnection urlcon=url.openConnection();
System.out.println(“ The URL is:” +urlcon.getURL());
System.out.println(“ The User Interaction is:”
+urlcon.getAllowUserInteraction());
System.out.println(“ The Content Type is:” +urlcon.getContentType());
System.out.println(“ The URLDo Input is:” +urlcon.getDoInput());
System.out.println(“ The URLDo Output is:” +urlcon.getDoOutput());
System.out.println(“Last modified Date is:” +new
Date(urlcon.getLastModified());
System.out.println(“ The content length is:” +urlcon.getContentLength());
System.out.println(“ The content encoding is:” +urlcon.getContentEncoding());
}
}
}
```

Program 2 (download):

```
import java.io.*;
import java.net.*;
public class URLConnection {
public static void main(String[] args){
URL url=new URL("http://www.javatpoint.com/java-tutorial");
URLConnection urlcon=url.openConnection();
InputStream is=urlcon.getInputStream();
FileOutputStream os = new FileOutputStream(“C: //abcd.pdf”);
Byte[] b = new byte[1024];
int i;
while((i=is.read())!=-1){
os.write(b,0,length);
}
is.close();
os.close();
}
}
```

Java InetAddress class

Java InetAddress class represents an IP address. The java.net.InetAddress class provides methods to get the IP of any host name *for example* www.javatpoint.com, www.google.com, www.facebook.com, etc.

An IP address is represented by 32-bit or 128-bit unsigned number. An instance of InetAddress represents the IP address with its corresponding host name. There are two types of address types: Unicast and Multicast. The Unicast is an identifier for a single interface whereas Multicast is an identifier for a set of interfaces.

Moreover, InetAddress has a cache mechanism to store successful and unsuccessful host name resolutions.

Program:

```
package net;
import java.net.*;
public class InetDemo{
public static void main(String args[]) throws Exception {

InetAddress in = InetAddress.getByName(www.rajagiri.edu);
System.out.println("The InetAddress is " +in);
System.out.println("The Hostname is " +in.getHostName());
System.out.println("The IP is " +in.getHostAddress());

InetAddress im = InetAddress.getLocalHost();
System.out.println("The InetAddress is " +im);
System.out.println("The Hostname is " +im.getHostName());
System.out.println("The IP is " +im.getHostAddress());
}
}
}
```

Datagram

A datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed.

- Datagrams plays a vital role as an alternative.
- Datagrams are bundles of information passed between machines. Once the datagram has been released to its intended target, there is no assurance that it will arrive or even that someone will be there to catch it.
- Likewise, when the datagram is received, there is no assurance that it hasn't been damaged in transit or that whoever sent it is still there to receive a response and it is crucial point to note.

Java implements datagrams on top of the **UDP (User Datagram Protocol)** protocol by using two classes:

1. **DatagramPacket** object is the data container.
2. **DatagramSocket** is the mechanism used to send or receive the DatagramPackets.

Java DatagramSocket and DatagramPacket

Java DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

Java DatagramSocket class

Java DatagramSocket class represents a connection-less socket for sending and receiving datagram packets.

A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.

Commonly used Constructors of DatagramSocket class:

- **DatagramSocket() throws SocketException:** it creates a datagram socket and binds it with the available Port Number on the localhost machine.
- **DatagramSocket(int port) throws SocketException:** it creates a datagram socket and binds it with the given Port Number.
- **DatagramSocket (intport, InetAddress address) throws SocketException:** it creates a datagram socket and binds it with the specified port number and host address.

Syntax:

public DatagramPacket(byte[] buf, int offset, int length,

InetAddress address, int port)

Parameters:

Buf	: byte array
Offset	: offset into the array
Length	: length of message to deliver
Address	: address of destination
Port	: port number of destination

Java DatagramPacket class

Java DatagramPacket is a message that can be sent or received. If you send multiple packet, it may arrive in any order. Additionally, packet delivery is not guaranteed.

Commonly used Constructors of DatagramPacket class

- **DatagramPacket(byte[] barr, int length):** it creates a datagram packet. This constructor is used to receive the packets.
- **DatagramPacket(byte[] barr, int length, InetAddress address, int port):** it creates a datagram packet. This constructor is used to send the packets.

Syntax:

DiagramPacket(byte[] barr, int length, InetAddress address, int port)

Methods of Datagram Sockets

- Bind(): binds this socket to specified address and port number.
- Connect(): connects to the specified address and port.
- Disconnect(): disconnects the socket.
- isBound(): returns a Boolean value indicating whether this socket is bound or not
- isConnected(): returns Boolean value representing connection state of the socket.
- getInetAddress(): returns the address to which this socket is connected.
- getPort(): returns the port on the machine to which this socket is connected.
- getRemoteSocketAddress(): returns the socket address(IP address+port number)
- getLocalSocketAddress(): returns the address of the machine this socket is bound to
- send()
- receive()

Program:

UDPClient.java

```
package net;
import java.net.*;
public class UDPClient{
public static void main(String args[]) throws Exception{
DatagramSocket ds = new DatagramSocket();
int i=12;
byte[] b = String.valueof(i).getBytes();
InetAddress ia = InetAddress.getLocalHost();
DatagramPacket dp = new DatagramPacket(b,b.length,ia,236);
ds.send(dp);
}
}
```

UDPServer.java

```
package net;
import java.net.*;
public class UDPServer{
public static void main(String args[]) throws Exception{
DatagramSocket ds = new DatagramSocket(236);
byte[] b1 = new byte[1024];
DatagramPacket dp = new DatagramPacket(b1,b1.length);
ds.receive(dp);
String str = new String(dp.getData());
int num = Integer.parseInt(str.trim());
System.out.println(“The number(server) i is: ” +num);
int result = num*num;
System.out.println(“The result(server) i is: ” +result);
}
}
```

Output:

```
The number(server) i is: 12
The result(server) i is:144
```

Program 2: (Two way communication-UDP)

UDPClient.java

```
package net;
import java.net.*;
public class UDPClient{
public static void main(String args[]) throws Exception{
DatagramSocket ds = new DatagramSocket();

//sending
int i=12;
byte[] b = String.valueof(i).getBytes();
InetAddress ia = InetAddress.getLocalHost();
DatagramPacket dp = new DatagramPacket(b,b.length,ia,236);
ds.send(dp);

//receiving
byte[] b1 = new byte[1024];
DatagramPacket dp1 = new DatagramPacket(b1,b1.length);
ds.receive(dp1);

String str = new String(dp1.getData());
int r = Integer.parseInt(str.trim());

System.out.println(“The result(client)  is: ” +r);
int rnew = r+10;
System.out.println(“The new result(client) is: “ +rnew);
}
}
```

Output(client):

```
The result(client) is: 144
The new result(client) is: 154
```

UDPServer.java

```
package net;
import java.net.*;
public class UDPServer{
public static void main(String args[]) throws Exception{
DatagramSocket ds = new DatagramSocket(236);

// receiving
byte[] b1 = new byte[1024];
DatagramPacket dp = new DatagramPacket(b1,b1.length);
ds.receive(dp);

String str = new String(dp.getData());
int num = Integer.parseInt(str.trim());

System.out.println("The number(server) i is: " +num);
int result = num*num;
System.out.println("The result(server) i is: " +result);

//sending
byte[] b2 = String.valueof(result).getBytes();
InetAddress ia = InetAddress.getLocalHost();
DatagramPacket dp1 = new DatagramPacket(b2,b2.length,ia,dp.getPort());
ds.send(dp1);
}
}
```

Output(Server):

```
The number(server) i is: 12
The result(server) i is:144
```

Module 4 -JDBC

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

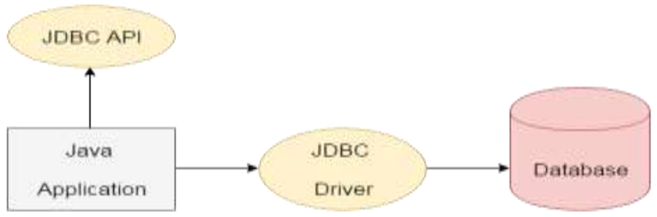
- JDBC-ODBC Bridge Driver,
- Native Driver,
- Network Protocol Driver, and
- Thin Driver

The **java.sql** package contains classes and interfaces for JDBC API. A list of popular *interfaces* of JDBC API are given below:

- Driver interface
- Connection interface
- Statement interface
- PreparedStatement interface
- CallableStatement interface
- ResultSet interface
- ResultSetMetaData interface
- DatabaseMetaData interface
- RowSet interface

A list of popular *classes* of JDBC API are given below:

- DriverManager class
- Blob class
- Clob class
- Types class



Why Should We Use JDBC

Before JDBC, ODBC API was the database API to connect and execute the query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

JDBC API

JDBC stands for **J**ava **D**ata**b**ase **C**onnectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage. We can use JDBC API to handle database using Java program and can perform the following activities:

1. Connect to the database
2. Execute queries and update statements to the database
3. Retrieve the result received from the database.

Applications of JDBC

Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database. Java can be used to write different types of executables, such as −

- Java Applications
- Java Applets
- Java Servlets
- Java ServerPages (JSPs)

- Enterprise JavaBeans (EJBs).

All of these different executables are able to use a JDBC driver to access a database, and take advantage of the stored data.

JDBC provides the same capabilities as ODBC, allowing Java programs to contain database-independent code.

What is API

API (Application programming interface) is a document that contains a description of all the features of a product or software. It represents classes and interfaces that software programs can follow to communicate with each other. An API can be created for applications, libraries, operating systems, etc.

JDBC Driver

JDBC Driver is a software component that enables java application to interact with the database.
There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

1) JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database.

The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

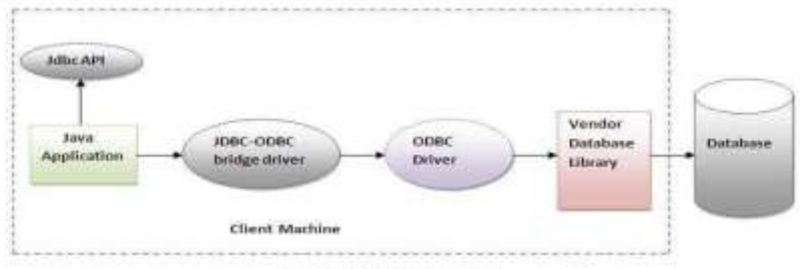


Figure- JDBC-ODBC Bridge Driver

In Java 8, the JDBC-ODBC Bridge has been removed.

Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

Advantages:

- easy to use.
- can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

2) Native-API driver

The Native API driver uses the client-side libraries of the database.

The driver converts JDBC method calls into native calls of the database API.

It is not written entirely in java.

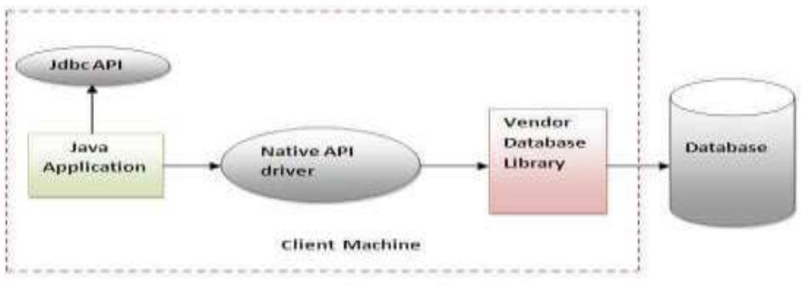


Figure- Native API Driver

Advantage:

- performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.

3) Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

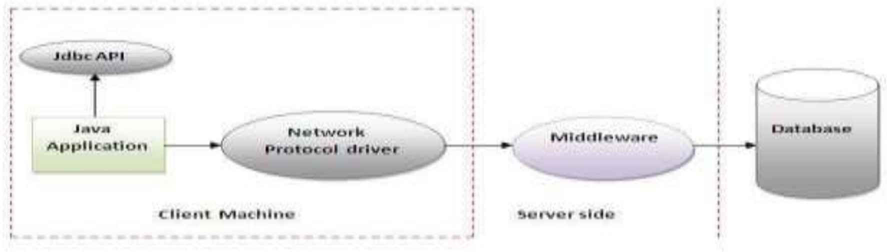


Figure- Network Protocol Driver

Advantage:

- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

4) Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol.

That is why it is known as thin driver. It is fully written in Java language.

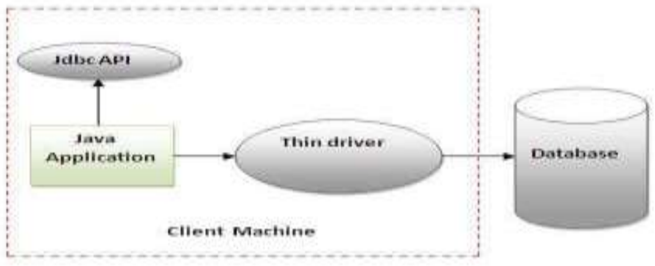


Figure- Thin Driver

Advantage:

- Better performance than all other drivers.
- No software is required at client side or server side.

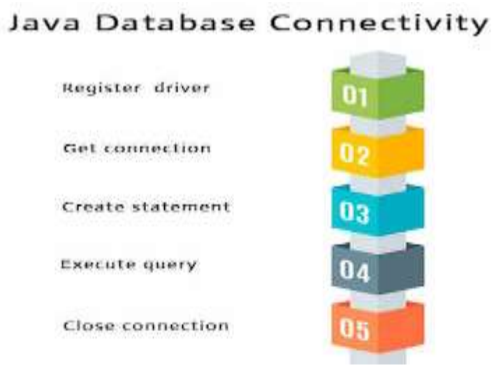
Disadvantage:

- Drivers depend on the Database.

Java Database Connectivity with 5 Steps

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

- Register the Driver class
- Create connection
- Create statement
- Execute queries
- Close connection



1) Register the driver class

The **forName()** method of Class class is used to register the driver class.

This method is used to dynamically load the driver class.

Syntax of forName() method

1. **public static void** forName(String className)**throws** ClassNotFoundException

Example to register the OracleDriver class

Here, Java program is loading oracle driver to esteblish database connection.

1. Class.forName("oracle.jdbc.driver.OracleDriver");

2) Create the connection object

The **getConnection()** method of DriverManager class is used to establish connection with the database.

Syntax of getConnection() method

```
1) public static Connection getConnection(String url)throws SQLException
2) public static Connection getConnection(String url,String name,String pasword)
   throws SQLException
```

Example to establish connection with the Oracle database

```
1. Connection con=DriverManager.getConnection(
2. "jdbc:oracle:thin:@localhost:1521:xe","system","password");
```

3) Create the Statement object

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax of createStatement() method

```
1. public Statement createStatement()throws SQLException
```

Example to create the statement object

```
1. Statement stmt=con.createStatement();
```

4) Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax of executeQuery() method

1. **public** ResultSet executeQuery(String sql)**throws** SQLException

Example to execute query

1. ResultSet rs=stmt.executeQuery("select * from emp");
2. **while**(rs.next()){
3. System.out.println(rs.getInt(1)+" "+rs.getString(2));
4. }

---

5) Close the connection object

By closing connection object statement and ResultSet will be closed automatically.

The close() method of Connection interface is used to close the connection.

Syntax of close() method

1. **public void** close()**throws** SQLException

Example to close connection

1. con.close();

It avoids explicit connection closing step.

Program: Display table

```
package JDBC;
import java.sql.*;

public class sel {
public static void main(String[] args){
try{
        //step1: Register/load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");

//step2: Get/create the connection object
Connection con = DriverManager.getConnection
("jdbc:oracle:thin:@localhost:1521:ORCL","mca","mca");

//step3: Create the statement object
Statement stmt = con.createStatement();

//step4: Execute query
ResultSet rs = stmt.executeQuery("select * from emp");
System.out.println("Connection success");

while(rs.next())
{
System.out.println(rs.getInt(1)+ "            " +rs.getString(2));
}
//step5: close the connection object
con.close();
}
catch(Exception e){}
}
}
```

Java ResultSetMetaData Interface

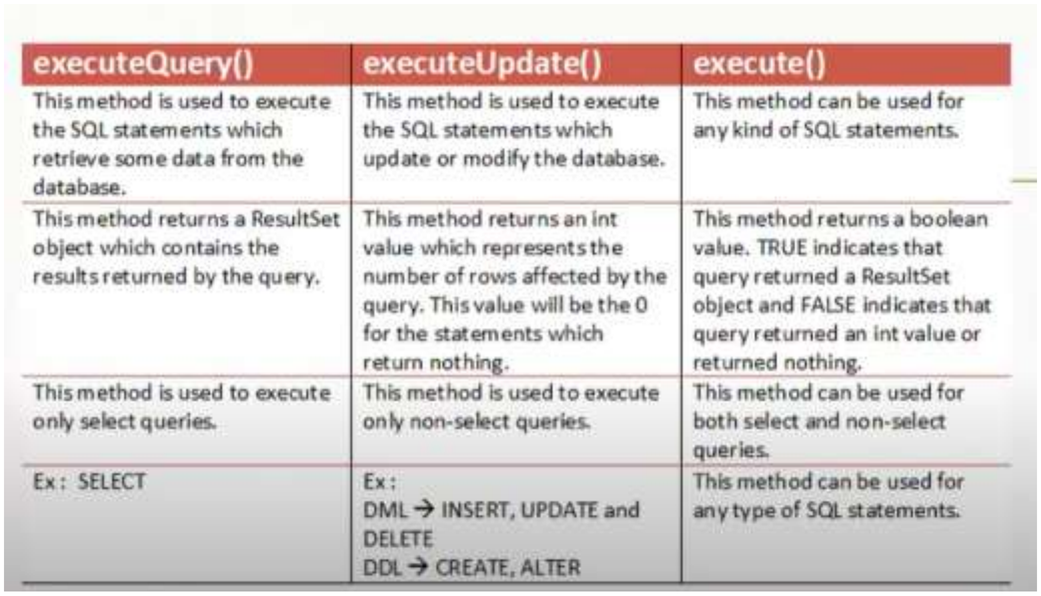
The metadata means data about data i.e. we can get further information from the data.

If you have to get metadata of a table like total number of column, column name, column type etc. , ResultSetMetaData interface is useful because it provides methods to get metadata from the ResultSet object.

Method	**Description**
public int getColumnCount()throws SQLException	it returns the total number of columns in the ResultSet object
public String getColumnName(int index)throws SQLException	it returns the column name of the specified column index.
public String getColumnTypeName(int index)throws SQLException.	it returns the column type name for the specified index.
public String getTableName(int index)throws SQLException	It returns the table name for the specified column index.

Program 2: Table creation and insertion

```
package JDBC;
import java.sql.*;

public class meta {
public static void main(String[] args){
try{
     //step1: Register/load the driver class
    Class.forName("oracle.jdbc.driver.OracleDriver");

    //step2: Get/create the connection object
   Connection con = DriverManager.getConnection
("jdbc:oracle:thin:@localhost:1521:ORCL","mca","mca");

   //step3: Create the statement object
Statement stmt = con.createStatement();

//step4: Execute query
stmt.executeUpdate("create table std(sid int NOT NULL PRIMARY KEY, sname
varchar2(20))");
System.out.println("Table created");

String qry = "insert into std values(1,'Janet')";
stmt.executeUpdate(qry);
String qry1 = "insert into std values(2,'Remya')";
stmt.executeUpdate(qry1);
String qry2 = "insert into std values(3,'Pavitra')";
stmt.executeUpdate(qry2);
String qry3 = "insert into std values(4,'Kavya')";
stmt.executeUpdate(qry3);

ResultSet rs= stmt.executeQuery("select * from std");

ResultSetMetaData rd = rs.getMetaData();
System.out.println("Column  :" +rd.getColumnCount());
System.out.println("Column label : " +rd.getColumnLabel(1));
System.out.println("Column type  :" +rd.getColumnTypeName(1));

for(int i=1;i<=rd.getColumnCount();i++)
{
System.out.println(rd.getColumnLabel(i) + "        ");
}
System.out.println("                    ");
System.out.println("-----------------------------------");
while(rs.next())
{
System.out.println(rs.getInt(1)+ "               " +rs.getString(2));
}

//step5: close the connection object
con.close();
}
catch(Exception e){}
}
}
```



executeQuery()	executeUpdate()	execute()
This method is used to execute the SQL statements which retrieve some data from the database.	This method is used to execute the SQL statements which update or modify the database.	This method can be used for any kind of SQL statements.
This method returns a ResultSet object which contains the results returned by the query.	This method returns an int value which represents the number of rows affected by the query. This value will be the 0 for the statements which return nothing.	This method returns a boolean value. TRUE indicates that query returned a ResultSet object and FALSE indicates that query returned an int value or returned nothing.
This method is used to execute only select queries.	This method is used to execute only non-select queries.	This method can be used for both select and non-select queries.
Ex : SELECT	Ex : DML → INSERT, UPDATE and DELETE DDL → CREATE, ALTER	This method can be used for any type of SQL statements.

Program 3: table creation and insertion using foreign key

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

//Connect table with referential integrity
public class ConnectTableForeignKey {

        public static void main(String[] args) {
            try{

            //Register/Load the driver class
            Class.forName("oracle.jdbc.driver.OracleDriver");
            //Get /Create the connection object
            Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ORCL","mca","
mca");
            //Create the statement object
            Statement stmt = con.createStatement();
            //Execute query
```

```
			stmt.executeUpdate("create table course (cid int not null 
primary key,cname varchar2(20))");
			stmt.executeUpdate("create table stu (sid int not null primary 
key,sname varchar2(20),cid int references course(cid))");
			System.out.println("Tables created");

			stmt.executeUpdate("insert into course values (1,'Java')");
			int i =2;
			String n=”DCN”;
			stmt.executeUpdate("insert into course values (“+i+”,’”+n+"’);

			stmt.executeUpdate("insert into stu values (1,'Abiya',1)");
			stmt.executeUpdate("insert into stu values (2,'Ben',1)");
			stmt.executeUpdate("insert into stu values (3,'Arathi',2)");
			stmt.executeUpdate("insert into stu values (4,'Denia',2)");

			//Close the connection object
			con.close();
			}
			catch (Exception e)
			{
				System.out.println(e);

			}
	}


}
```

PreparedStatement interface

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.

Let's see the example of parameterized query:

1. String sql="insert into emp values(?,?,?)";

Why use PreparedStatement?

- **Improves performance**: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.
- Used for parameterized queries
- Can reuse new parameter values
- Enables easier batch updates
- Accepts input parameters at runtime.

Method	Description
public void setInt(int paramIndex, int value)	sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value)	sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	sets the double value to the given parameter index.
public int executeUpdate()	executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery()	executes the select query. It returns an instance of ResultSet.

Program: insertion using parameterized queries

```
import java.sql.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class ParameterizedQuery {
	public static void main(String[] args) throws Exception{

				//Register/Load the driver class
			Class.forName("oracle.jdbc.driver.OracleDriver");
					//Get /Create the connection object
			Connection con = 
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ORCL","mca","
mca");
					//Create the statement object
			PreparedStatement pst = con.prepareStatement("insert into stutest 
values (?,?)");

			pst.setInt(1,100);
			pst.setString(2,"ABC");
			pst.executeUpdate();


			pst.setInt(1,200);
			pst.setString(2,"CDE");
			pst.executeUpdate();

			System.out.println("Entry Success");
			con.close();

	}

}
```

Program: update

```
import java.sql.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class ParameterizedUpdate {

	public static void main(String[] args) throws Exception{
		// TODO Auto-generated method stub
		Class.forName("oracle.jdbc.driver.OracleDriver");
```

```
                    //Get /Create the connection object
Connection con = 
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ORCL","mca","
mca");
                    //Create the statement object
PreparedStatement pst = con.prepareStatement("update stutest set sname = ? where 
sid=?");

pst.setString(1,"XYZ");
pst.setInt(2,100);
pst.executeUpdate();

System.out.println("Update Success");
con.close();
          }
}
```

Program: Delete

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class ParameterizedDelete {

          public static void main(String[] args) throws Exception{
                    // TODO Auto-generated method stub
                    Class.forName("oracle.jdbc.driver.OracleDriver");
                    //Get /Create the connection object
Connection con = 
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ORCL","mca","
mca");
                    //Create the statement object
PreparedStatement pst = con.prepareStatement("delete from stutest where sid =?");

pst.setInt(1,100);
pst.executeUpdate();

System.out.println("Delete Success");
con.close();
```

```
            }

}
```

Program: insert from keyboard and using parameterized queries

```
import java.io.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class ParameterizedInsert {

                        public static void main(String[] args) throws Exception{

            Class.forName("oracle.jdbc.driver.OracleDriver");
                                    //Get /Create the connection object
            Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ORCL","mca","
mca");
                                    //Create the statement object
            PreparedStatement pst = con.prepareStatement("insert into stutest
values(?,?)");
            BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));

do{
            System.out.println("Enter student ID:");
            int id = Integer.parseInt(br.readLine());
            System.out.println("Enter student name:");
            String name = br.readLine();

            pst.setInt(1,id);
            pst.setString(2,name);
            pst.executeUpdate();

            System.out.println("Continue:Yes/No");
            String s=br.readLine();
If(s.startsWith("n")) {break;}

}while(true);
            System.out.println("Insert Success");
            con.close();
                        }
}
```

Java CallableStatement Interface

CallableStatement interface is used to call the **stored procedures and functions**.

We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.

Suppose you need the get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output.

Stored Procedure	Function
is used to perform business logic.	is used to perform calculation.
must not have the return type.	must have the return type.
may return 0 or more values.	may return only one values.
We can call functions from the procedure.	Procedure cannot be called from function.
Procedure supports input and output parameters.	Function supports only input parameter.
Exception handling using try/catch block can be used in stored procedures.	Exception handling using try/catch can't be used in user defined functions.

How to get the instance of CallableStatement?

The prepareCall() method of Connection interface returns the instance of CallableStatement. Syntax is given below:

1. **public** CallableStatement prepareCall("{ call procedurename(?,?...?)}");

The example to get the instance of CallableStatement is given below:

1. CallableStatement stmt=con.prepareCall("{call myprocedure(?,?)}");

It calls the procedure myprocedure that receives 2 arguments.

Example: Program

Procedure code:

```
Create or replace procedure “INS” (no IN NUMBER, name IN VARCHAR2)

IS

BEGIN

INSERT INTO EMP VALUES (NO, NAME);

END;

/
```

Program:

```
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;

public class CallableIntStoredProc  {
	public static void main(String[] args) throws Exception{
		Class.forName("oracle.jdbc.driver.OracleDriver");
```

```
                    Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ORCL","mca","
mca");
                    CallableStatement cs = con.prepareCall("{call INS(?,?)}");

                    cs.setInt(1,102);
                    cs.setString(2,"MNO");
                    cs.executeUpdate();

                    System.out.println("Procedure call Success");

          }

}
```

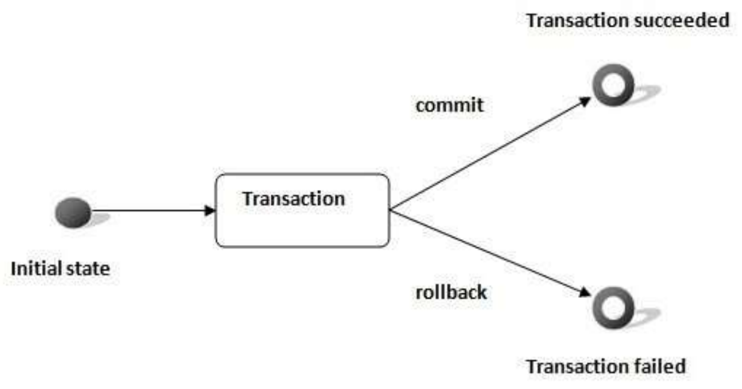
Transaction Management in JDBC

Transaction represents **a single unit of work**.

Advantage of Transaction Management

fast performance It makes the performance fast because database is hit at the time of commit.

- To maintain the integrity of business processes.
- To use distributed transactions



In JDBC, **Connection interface** provides methods to manage transaction.

Method	**Description**
void setAutoCommit(boolean status)	It is true bydefault means each transaction is committed bydefault.
void commit()	commits the transaction.
void rollback()	cancels the transaction.

Commit & Rollback

Commit permanently save changes done in the transaction in tables.It is unable to regain its previous state after the execution. When the transaction is successful, commit is applied.

Once you are done with your changes and you want to commit the changes then call **commit()** method on connection object as follows −

```
conn.commit( );
```

Rollback undo the transactions that have not been saved in the database. It is to undo changes since the last COMMIT. When the transaction is aborted, ROLLBACK occurs.

Otherwise, to roll back updates to the database made using the Connection named conn, use the following code −

```
conn.rollback( );
```

Whenever we use commit and rollback, we should use setAutoCommit(false).

Program 1:

```
import java.io.*;
import java.sql.*;

public class CommitRollBk {

	public static void main(String[] args) throws Exception{

		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection con = 
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ORCL","mca","
mca");

		con.setAutoCommit(false);
		Statement stmt = con.createStatement();
		stmt.executeUpdate("insert into stutest values (102,'PQR')");


		con.rollback();
		//con.commit(); // Optional as by default commit is ON
		con.close();
		System.out.println("Success");
	}


}
```

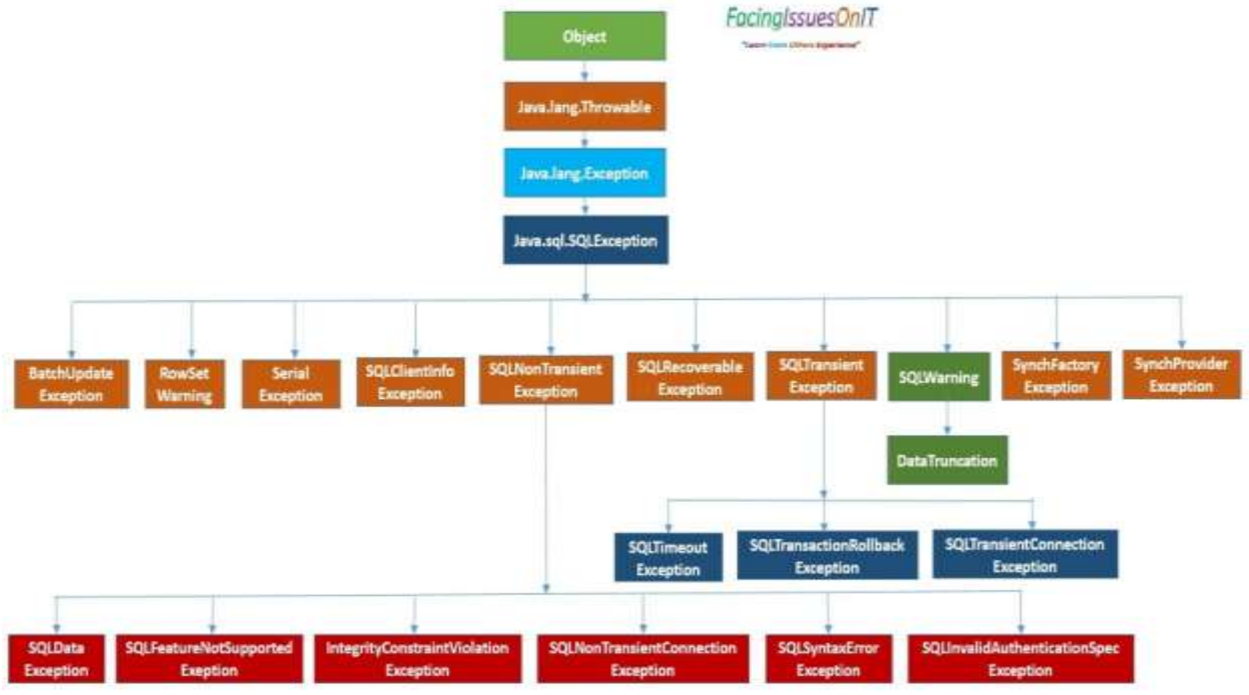
Program 2:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.Statement;

public class SelectiveCommitRollBk {
```

```
public static void main(String[] args) throws Exception{


            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ORCL","mca","
mca");
PreparedStatement pst = con.prepareStatement("insert into stutest values(?,?)");

            con.setAutoCommit(false);
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
            do{
            System.out.println("Enter student ID:");
            int id = Integer.parseInt(br.readLine());
            System.out.println("Enter student name:");
            String name = br.readLine();

            pst.setInt(1,id);
            pst.setString(2,name);
            pst.executeUpdate();


            System.out.println("Enter c for commit/r for rollback");
            String ch = br.readLine();

            if (ch.equals("r"))
                  {con.rollback();}
            else
                  {con.commit(); }// Optional as by default commit is ON
            }while(true);
            con.close();
            System.out.println("Success");
      }
}
```

Statement	PreparedStatement	CallableStatement
It is used to execute normal SQL queries.	It is used to execute parameterized or dynamic SQL queries.	It is used to call the stored procedures.
It is preferred when a particular SQL query is to be executed only once.	It is preferred when a particular query is to be executed multiple times.	It is preferred when the stored procedures are to be executed.
You cannot pass the parameters to SQL query using this interface.	You can pass the parameters to SQL query at run time using this interface.	You can pass 3 types of parameters using this interface. They are – IN, OUT and IN OUT.
This interface is mainly used for DDL statements like CREATE, ALTER, DROP etc.	It is used for any kind of SQL queries which are to be executed multiple times.	It is used to execute stored procedures and functions.
The performance of this interface is very low.	The performance of this interface is better than the Statement interface (when used for multiple execution of same query).	The performance of this interface is high.

JDBC Exception Handling



SQLException Hierarchy

Exception handling allows you to handle exceptional conditions such as program-defined errors in a controlled fashion.

When an exception condition occurs, an exception is thrown. The term thrown means that current program execution stops, and the control is redirected to the nearest applicable catch clause. If no applicable catch clause exists, then the program's execution ends.

JDBC Exception handling is very similar to the Java Exception handling but for JDBC, the most common exception you'll deal with is **java.sql.SQLException.**

SQLException is Checked Exception it encounters an error when interacting with database , executing query on Databases etc. It throws instance of SQLException. **SQLException** contains several kind information on a database access error or other errors which can help to determine the cause of error:

- **Description of Error :** A string describing the error mesage. It can be retrieved by SQLException.getMessage().
- **SQLStateCode :** These codes and their respective meanings have been standardized by ISO/ANSI and Open Group (XOPEN), although some codes have been reserved for database vendors to define for themselves. This String object consists of five alphanumeric characters. Retrieve this code by calling the method SQLException.getSQLState().
- **DatabaseMetaData:** The DatabaseMetaData method getSQLStateType can be used to discover whether the driver returns the XOPEN type or the SQL:2003 type.
- **ErrorCode:** This is an integer value identifying the error that caused the SQLException instance to be thrown. Its value and meaning are implementation-specific and might be the actual error code returned by the underlying data source. Retrieve the error by calling the method SQLException.getErrorCode().
- **Exception Chaining:** If more than one error occurs, the exceptions are referenced through this chain. Retrieve these exceptions by calling the method SQLException.getNextException() on the exception that was thrown.
- **Cause:** A SQLException instance might have a causal relationship, which consists of one or more Throwable objects that caused the SQLException instance to be thrown. To navigate this chain of causes, recursively call the method SQLException.getCause() until a null value is returned.

SQLWarning

SQLWarning objects are a subclass of SQLException that deal with database access warnings. Warnings do not stop the execution of an application, as exceptions they simply alert the user that something did not happen as planned. **For example**, a warning might let you know that a privilege you attempted to revoke was not revoked. Or a warning might tell you that an error occurred during a requested disconnection.
A warning can be reported on a Connection object, a Statement object (including PreparedStatement and CallableStatement objects), or a ResultSet object. Each of these classes has a getWarnings method, which you must invoke in order to see the first warning reported on the calling object. If getWarnings returns a warning, you can call the SQLWarning method getNextWarning on it to get any additional warnings. Executing a statement automatically clears the warnings from a previous statement, so they do not build up. This means, however, that if you want to retrieve warnings reported on a statement, you must do so before you execute another statement.

Example:

```
import java.sql.*;
public class ExpDemo {
public static void main(String[] args){
try{

Class.forName("oracle.jdbc.driver.OracleDriver");

Connection con = DriverManager.getConnection
("jdbc:oracle:thin:@localhost:1521:ORCL","mca","mca");

Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("select * from emp where empname=Devika");

con.close();
}
catch(Exception e)
{
System.out.println("SQL Message: "+e.getMessage());
System.out.println("SQL State: "+e.getSQLState());
System.out.println("SQL Error Code: "+e.getErrorCode());
System.out.println("SQL Cause: "+e.getCause());
e.printStackTrace();
            }
            }
}
```

Output:

```
SQL Message: ORA-00904: "DEVIKA" :invalid identifier
SQL State 42000
SQL Error Code 904
Java.sql.SQLException: ORA-00904: "DEVIKA" :invalid identifier
SQL Cause null
```

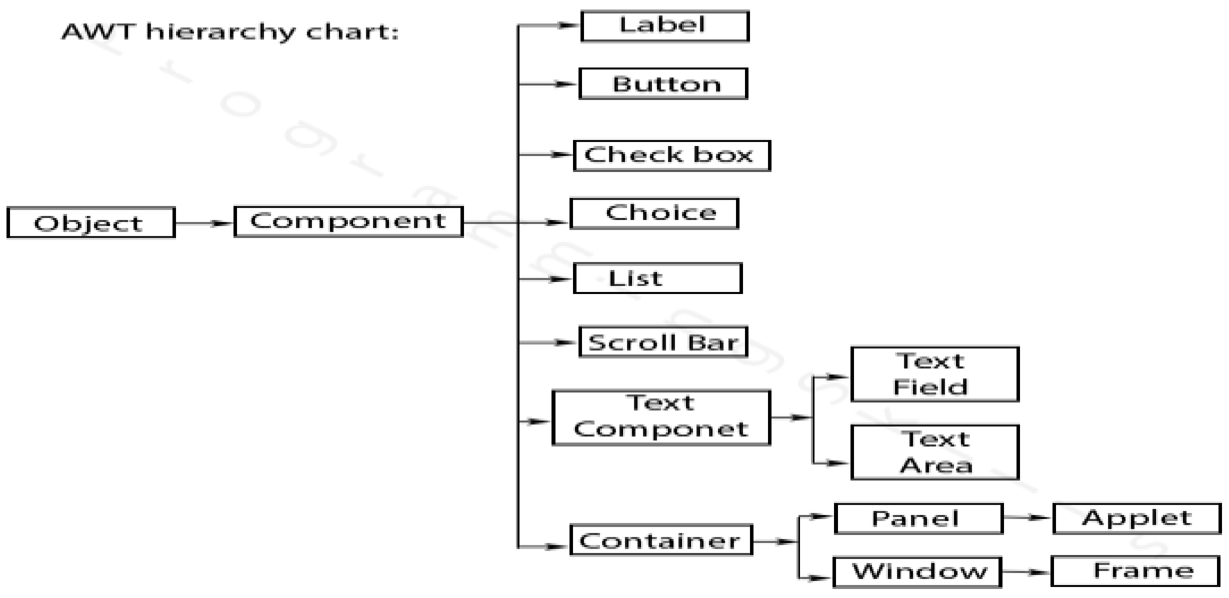
Module 5-AWT,Swing and Applets

AWT in Java

Java AWT (Abstract Window Toolkit) is *an* Application programming interface *API to develop GUI or window-based applications* in java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.

The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.



Frame

The frame is a subclass of Window. Frame defines a top-level Window with Borders and a Menu Bar. It can be defined as a container with components like button, textfield, label, etc. Once defined, a Frame is a container which can contain components.

Useful Methods of Component class

Method	Description
public void add(Component c)	inserts a component on this component.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.

Frame Creation

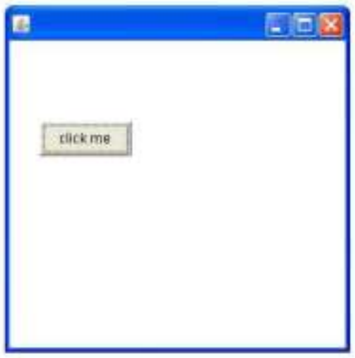
By extending Frame class(inheritance)

By creating the object of Frame class(association).

Program: by inheritance

```
import java.awt.*;
class First extends Frame{
First(){
Button b=new Button("click me");
b.setBounds(30,100,80,30);// setting button position
add(b);//adding button into frame
setSize(300,300);//frame size 300 width and 300 height
setLayout(null);//no layout manager
setVisible(true);//now frame will be visible, by default not visible
}
public static void main(String args[]){
First f=new First();
}}
```

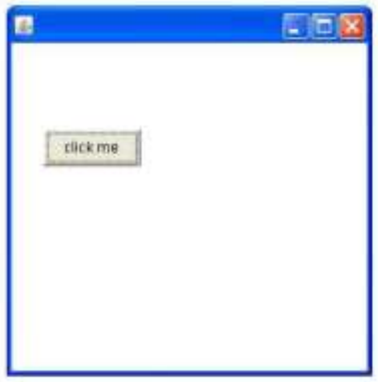
Output:



Program: association

```
import java.awt.*;
class First2{
First2(){
Frame f=new Frame();
Button b=new Button("click me");
b.setBounds(30,50,80,30);
f.add(b);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[]){
First2 f=new First2();
}}
```

Output:



Program: Set ImageIcon

```
import java.awt.*;
class First2{
First2(){
Frame f=new Frame(“My Java Frame”);
f.setSize(300,300);
f.setBackground(Color.red);
f.setForeground(Color.blue);
f.setVisible(true);
Image
i=ToolKit.getDefaultToolKit().getImage(“/E:/backup/Downloads/1,png”);
f.setIconImage(i);
}
public static void main(String args[]){
First2 f=new First2();
}}
```

Java AWT Button

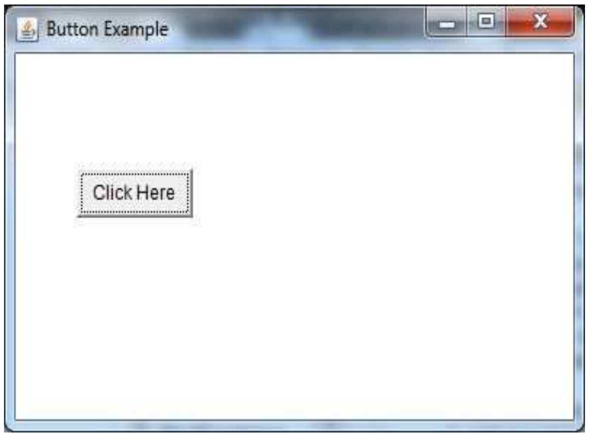
The button class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. When a button is pressed, it notifies its listeners.

A button has **a SetBounds** that has 4 parameters (x-axis,y-axis,width,height).

Program:

```
import java.awt.*;
class First2{
First2(){
Frame f=new Frame(“My Java Frame”);
f.setSize(300,300);
Button b=new Button("Click Here");
b.setBounds(50,100,80,30);
f.add(b);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[]){
First2 f=new First2();
}}
```

Output:



Java AWT Label

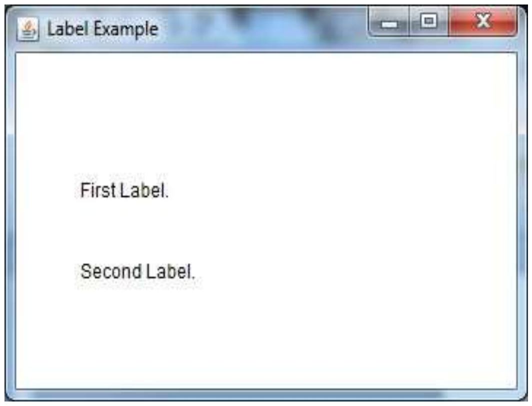
The object of Label class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly.

Program:

```
import java.awt.*;

class First2{
First2(){
Frame f=new Frame(“Label”);
f.setSize(300,300);
Label l1,l2;
   l1=new Label("First Label.");
   l1.setBounds(50,100, 100,30);
   l2=new Label("Second Label.");
   l2.setBounds(50,150, 100,30);
   f.add(l1);
   f.add(l2);  f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[]){
```

```
First2 f=new First2();
}}
```

Output:



First Label.

Second Label.

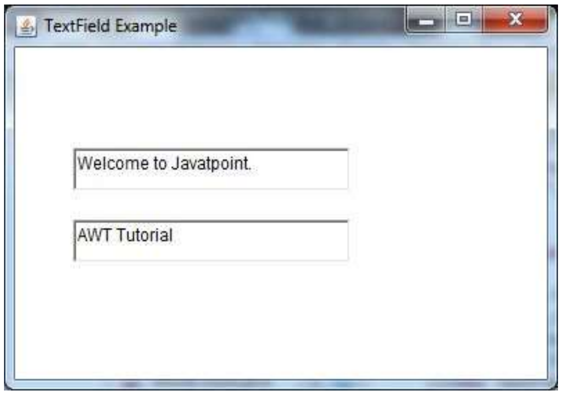
Java AWT TextField

The object of a TextField class is a text component that allows the editing of a single line text. It inherits TextComponent class.

Program:

```
import java.awt.*;

class First2{
First2(){
Frame f=new Frame(“Label”);
f.setSize(300,300);
 TextField t1,t2;
   t1=new TextField("Welcome to Javatpoint.");
   t1.setBounds(50,100, 200,30);
   t2=new TextField("AWT Tutorial");
   t2.setBounds(50,150, 200,30);
   f.add(t1); f.add(t2);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[]){
First2 f=new First2();
}}
```

Output:



```
Welcome to Javatpoint.
AWT Tutorial
```

What is an Event?

Change in the state of an object is known as event i.e. event describes the change in state of source. Events are generated as result of user interaction with the graphical user interface components. For example, clicking on a button, moving the mouse, entering a character through keyboard,selecting an item from list, scrolling the page are the activities that causes an event to happen.

Types of Event

The events can be broadly classified into two categories:

- **Foreground Events -** Those events which require the direct interaction of user.They are generated as consequences of a person interacting with the graphical components in Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard,selecting an item from list, scrolling the page etc.
- **Background Events -** Those events that require the interaction of end user are known as background events. Operating system interrupts, hardware or

software failure, timer expires, an operation completion are the example of background events.

What is Event Handling?

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism have the code which is known as event handler that is executed when an event occurs. Java Uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events.Let's have a brief introduction to this model.

The Delegation Event Model has the following key participants namely:

- **Source -** The source is an object on which event occurs. Source is responsible for providing information of the occurred event to it's handler. Java provide as with classes for source object.
- **Listener -** It is also known as event handler.Listener is responsible for generating response to an event. From java implementation point of view the listener is also an object. Listener waits until it receives an event. Once the event is received , the listener process the event an then returns.

Java Event classes and Listener interfaces

Event Classes	**Listener Interfaces**
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener

ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Steps involved in event handling

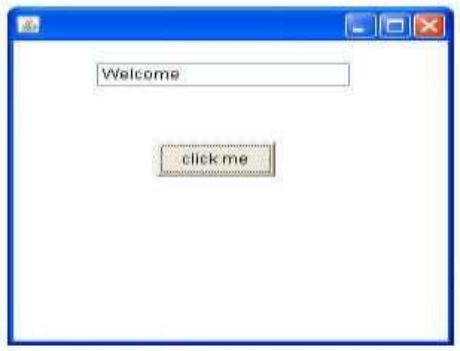
- Register the component with listener
- Perform event handling
  - The User clicks the button and the event is generated.
  - Now the object of concerned event class is created automatically and information about the source and the event get populated with in same object.
  - Event object is forwarded to the method of registered listener class.
  - the method is now get executed and returns.

Example 1:

```
import java.awt.*;
import java.awt.event.*;
class AEvent extends Frame implements ActionListener, WindowListener {
TextField tf;
AEvent(){
```

```
//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);

//register listener
b.addActionListener(this);//passing current instance
f.addWindowListener(this);

//add components and set size, layout and visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void windowClosing(WindowEvent e){
System.exit(0);
}

public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}
public static void main(String args[]){
new AEvent();
}
}
```

Output:

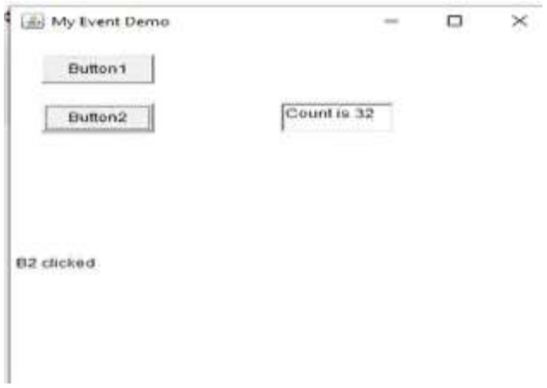


Example 2:

```
import java.awt.*;
import java.awt.event.*;
class AEvent implements ActionListener, WindowListener {
TextField tf;
Button b1,b2;
Label l;
AEvent(){
Frame f=new Frame("Event Demo");
setSize(300,300);
setLayout(null);
setVisible(true);

tf=new TextField();
tf.setBounds(200,100,80,30);
f.add(tf);

b1=new Button("Button1");
b1.setBounds(30,50,80,30);
f.add(b1);

b2=new Button("Button1");
b2.setBounds(30,50,80,30);
f.add(b2);
```

```
   b1.addActionListener(this);
   b2.addActionListener(this);
   f.addWindowListener(this);

   l=new Label("");
   l.setBounds(10,250,100,30);
   f.add(l);
 }
   public void windowClosing(WindowEvent e){
   System.exit(0);
   }

   public void actionPerformed(ActionEvent e){
   if(e.getSource()==b2)
   {l.setText("B2 clicked");}
   Else
   {
   l.setText("B1 clicked");
   String txt=tf.getText();
   Tf.setText("Count is "+txt.length());
   }
   public static void main(String args[]){
   new AEvent();
   }
   }
```

Output:



```
Count is 32
B2 clicked
```

CheckBox

Checkbox control is used to turn an option on(true) or off(false). There is label for each checkbox representing what the checkbox does. The state of a checkbox can be changed by clicking on it.

This class represents a GUI checkbox with a textual label. The Checkbox maintains a Boolean state indicating whether it is checked or not. If a checkbox is added to a CheckBoxGroup, it will behave like a radio button.

Program: check

```
import java.awt.*;
import java.awt.event.*;
public class CheckboxExample
{
    CheckboxExample(){
       Frame f= new Frame("CheckBox Example");
       final Label label = new Label();
       label.setAlignment(Label.CENTER);
       label.setSize(400,100);
       Checkbox checkbox1 = new Checkbox("Java");
       Checkbox1.setBounds(100,150, 50,50);
    Checkbox checkbox2 = new Checkbox("CG",true);
       Checkbox2.setBounds(100,100, 50,50);
Checkbox checkbox3 = new Checkbox("SS");
       Checkbox3.setBounds(100,100, 50,50);

       f.add(checkbox1); f.add(checkbox2);f.add(checkbox3); f.add(label);
    f.setSize(400,400);
       f.setLayout(null);
       f.setVisible(true);
    }
public static void main(String args[])
{
   new CheckboxExample();
}
}
```

Example 2: checkboxgroup

```
import java.awt.*;
import java.awt.event.*;

public class chkbox {
	chkbox()
	{
	Frame f = new Frame("CB DEMO");
	f.setSize(300,300);
	f.setLayout(null);
	f.setVisible(true);

	CheckboxGroup cbg = new CheckboxGroup();

	Checkbox cb1 = new Checkbox("Chicken Biriyani",cbg,false);
	cb1.setBounds(20,50,100,30);
	Checkbox cb2 = new Checkbox("Beef Biriyani",cbg,true);
	cb2.setBounds(20,100,100,30);

	f.add(cb1);
	f.add(cb2);

	}
	public static void main(String[] args) throws Exception{

		chkbox b =  new chkbox();

	}
}
```

Java AWT List

This class is a component which displays a list of strings. The list is scrollable, if necessary. Sometimes called ListBox in other languages. List can be set up to allow single or multiple selections. The list will return an array indicating which Strings are selected. It inherits Component class.

Program:

```
import java.awt.*;
import java.awt.event.*;

public class listdemo{
        TextField t1,t2;
        Button b1;
        List l1;
        listdemo()
        {
                Frame f = new Frame("Hello");
                f.setSize(300,300);
                f.setLayout(null);
                f.setVisible(true);

                l1 = new List(5);
                l1.setBounds(100,100,50,50);
                l1.add("A");
                l1.add("B");
                l1.add("C");
                l1.add("D");
                f.add(l1);

                t1 = new TextField();
                t1.setBounds(50,100,200,30);
                f.add(t1);

                b1 = new Button("Click");
                b1.setBounds(30,150,80,30);
                f.add(b1);
                b1.addActionListener(this);


        }

        public void actionPerformed(ActionEvent e)
        {
                String t = l1.getItem(l1.getSelectedIndex());
                t1.setText(t);
        }

                public static void main(String[] args){

                        listdemo ld =  new listdemo();
                }
        }
```

Java AWT Choice

This class represents a dropdown list of strings. Similar to a list in terms of functionality, but displayed differently. Only one item from the list can be selected at one time and the currently selected element is displayed. The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits Component class. It is non-scrollable.

Program:

```
import java.awt.*;
import java.awt.event.*;

public class listdemo{
        TextField t1,t2;
        Button b1;
        List l1;
        listdemo()
        {
                Frame f = new Frame("Hello");
                f.setSize(300,300);
                f.setLayout(null);
```

```
          f.setVisible(true);

          l1 = new Choice();
          l1.setBounds(100,100,50,50);
          l1.add("A");
          l1.add("B");
          l1.add("C");
          l1.add("D");
          f.add(l1);

          t1 = new TextField();
          t1.setBounds(50,100,200,30);
          f.add(t1);

          b1 = new Button("Click");
          b1.setBounds(30,150,80,30);
          f.add(b1);

          b1.addActionListener(this);


}

public void actionPerformed(ActionEvent e)
{
          String t = l1.getItem(l1.getSelectedIndex());
          t1.setText(t);
}

          public static void main(String[] args){

                    listdemo ld =  new listdemo();

          }

}
```

Program 2: sum of Numbers

```
import java.awt.*;
import java.awt.event.*;

public class sumawt implements ActionListener {
	TextField t1,t2,t3;
	Button b1,b2;
	sumawt()
	{
		Frame f = new Frame("Hello");
		f.setSize(300,300);
		f.setLayout(null);
		f.setVisible(true);

		Label l1 =  new Label("Enter n1");
		l1.setBounds(30,50,80,30);
		f.add(l1);
		Label l2 =  new Label("Enter n2");
		l2.setBounds(30,50,80,30);
		f.add(l2);


		t1= new TextField();
		t1.setBounds(50,50,50,30);
		f.add(t1);

		b1 = new Button("Add");
		b1.setBounds(50,100,200,30);
		f.add(b1);

		t2 = new TextField();
		t2.setBounds(50,100,200,30);
		f.add(t2);

		t3= new TextField();
		t3.setBounds(50,200,200,30);
		f.add(t2);

		b1.addActionListener(this);
}

          public void actionPerfomed(ActionEvent e)
          {
                    int n1 = Integer.parseInt(t1.getText());
                    int n2 = Integer.parseInt(t2.getText());
                    int r = n1+n2;
                    t3.setText("the sum is" +r);
          }

          public static void main(String[] args) throws Exception {

                    sumawt sa =  new sumawt();
          }
}
```

Java AWT MenuBar and Menu

The object of MenuItem class adds a simple labeled menu item on menu. The items used in a menu must belong to the MenuItem or any of its subclass.

The object of Menu class is a pull down menu component which is displayed on the menu bar. It inherits the MenuItem class.

The MenuBar class provides menu bar bound to a frame and is platform specific.

FileDialog

FileDialog control represents a dialog window from which the user can select a file.

- **static int LOAD --** This constant value indicates that the purpose of the file dialog window is to locate a file from which to read.
- **static int SAVE --** This constant value indicates that the purpose of the file dialog window is to locate a file to which to write.

Program:

```
import java.awt.*;
import java.awt.event.*;

public class menu implements ActionListener{
        Label l1;
        menu()
        {
                Frame f = new Frame("MENU DEMO");
                f.setSize(300,300);
                f.setLayout(null);
                f.setVisible(true);

                MenuBar mb = new MenuBar();
                Menu m1 = new Menu("Cars");
                mb.add(m1);
                Menu m2 = new Menu("Bikes");
                mb.add(m2);
                Menu m3 = new Menu("Autos");
                mb.add(m3);

                MenuItem mi1 = new MenuItem("Benz");
                m1.add(mi1);
                MenuItem mi2 = new MenuItem("Audi");
                m1.add(mi2);
                MenuItem mi3 = new MenuItem("Pulsar");
                m2.add(mi3);
                MenuItem mi4 = new MenuItem("BMW");
                m1.add(mi4);

                mi1.addActionListener(this);
                mi2.addActionListener(this);
                mi3.addActionListener(this);
                f.setMenuBar(mb);

                l1 =  new Label();
                l1.setBounds(30,50,80,30);
                f.add(l1);
                l1.addActionListener(this);
        }

        public void actionPerformed(ActionEvent e)
        {
                FileDialog fd=new FileDialog(f,’openfile’,FileDialog.LOAD);
                fd.setVisible(true);
                String dir=fd.getDirectory();
                String f=fd.getFile();
                l1.setText(f);
        }
        public static void main(String args[])
        {
                menu m = new menu();
        }
}
```

Panel

The class **Panel** is the simplest container class. It provides space in which an application can attach any other component, including other panels. It uses FlowLayout as default layout manager. It doesn't have title bar. It inherits the container class.

Program:

```
import java.awt.Button;
import java.awt.Frame;
import java.awt.List;
import java.awt.TextField;

public class panelDemo{

        panelDemo()
        {
                Frame f = new Frame("PANEL DEMO");
                f.setSize(300,300);
                f.setLayout(null);
                f.setVisible(true);
```

```
                 panel p =  new panel();
                p.setBounds(40,80,200,200);
                p.setBackground(Color.gray);


            Button b1=new Button("Button 1");
            b1.setBounds(50,100,80,30);
            b1.setBackground(Color.yellow);
            f.add(b1);
            Button b2=new Button("Button 2");
            b2.setBounds(100,100,80,30);
            b2.setBackground(Color.green);
            f.add(b2);
            p.add(b1); p.add(b2);
            f.add(p);

        }
        Public static void main(String args[])
        {
                new PanelDemo();
         }
}
```

Layout Managers

The layout manager automatically positions all the components within the container. If we do not use layout manager then also the components are positioned by the default layout manager.

Layout Managers enables us to

- control the way in which visual components are arranged in the GUI forms by determining the
    - size
    - position of components within the containers.

Java provide us with various layout manager to position the controls. The properties like size,shape and arrangement varies from one layout manager to other layout manager. When the size of the applet or the application window changes the size, shape and arrangement of the components also changes in response i.e. the layout managers adapt to the dimensions of appletviewer or the application window.
The layout manager is associated with every Container object. Each layout manager is an object of the class that implements the LayoutManager interface.

Layout managers solves

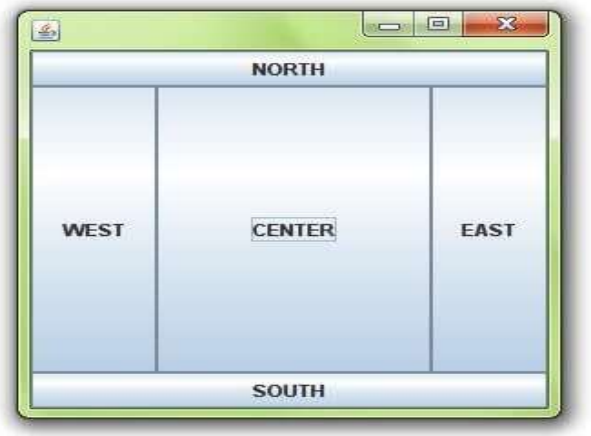
- parameters provided are defined in terms of pixels.
- Pixed sizes may be different (depending on the platform).
- Tend to produce GUIs which will not display properly on all the platforms.

Types of LayoutManagers

- BorderLayout
- GridLayout
- FlowLayout
- BoxLayout
- GridBagLayout

Java BorderLayout

The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

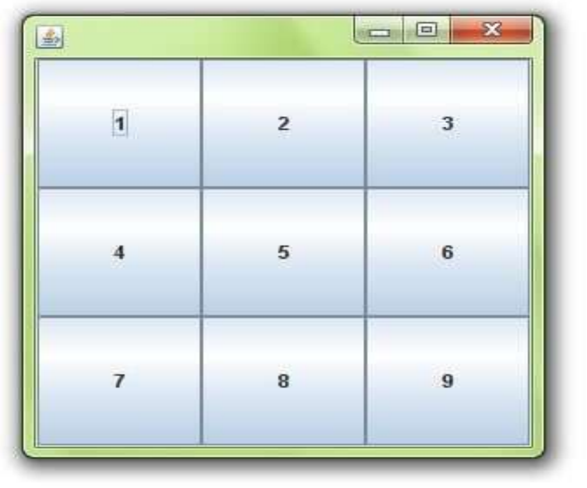


Program:

```
import java.awt.*;
public class Border
{
Frame f;
Button b1,b2,b3,b4,b5;
Border()
{
f=new Frame();
f.setSize(500,500);
//f.setLayout(null);
f.setVisible(true);

b1=new Button("Button 1");
f.add(b1,BorderLayout.NORTH);

b2=new Button("Button 2");
f.add(b2,BorderLayout.SOUTH);

b3=new Button("Button 3");
f.add(b3,BorderLayout.CENTER);

b4=new Button("Button 4");
f.add(b4,BorderLayout.WEST);

b5=new Button("Button 5");
f.add(b5,BorderLayout.EAST);
}

public static void main(String[] args)
{
Border b=new Border();
}

}
```

Java GridLayout

The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.



Program:

```
import java.awt.*;
public class Grid {
        Frame f;
        Button b1,b2,b3,b4,b5;
        Grid(){
                f = new Frame();
                b1 = new Button("B1");
                b2 = new Button("B2");
                b3 = new Button("B3");
                b4 = new Button("B4");
                b5 = new Button("B5");
                f.add(b1);  f.add(b2);  f.add(b3);  f.add(b4);  f.add(b5);
                f.setLayout(new GridLayout(3,4));       //3 rows and 4 columns
                f.setSize(500,500);
                f.setVisible(true);

        }
        public static void main(String args[]) {
                new Grid();

        }

}
```

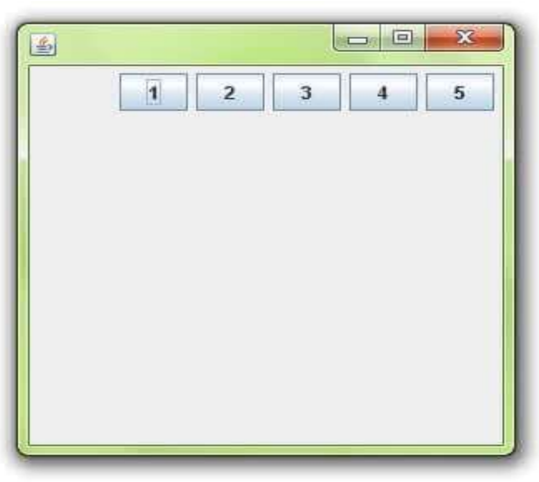
Program 2:

```
import java.awt.*;

public class Grid1 {
        Frame f;
        Button b[];
        Grid1(){
                f = new Frame();
                b = new Button[10];
                for(int i = 0;i<10;i++)
                {
                        b[i] = new Button("B "+(i+1));
                        f.add(b[i]);
                }

                f.setLayout(new GridLayout(3,4));       //3 rows and 4 columns
                f.setSize(500,500);
                f.setVisible(true);
        }
        public static void main(String args[]) {
                new Grid();

        }
}
```

Java FlowLayout

The FlowLayout is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel.

LEFT,RIGHT,CENTER,LEADING,TRAILING



Program:

```
import java.awt.*;
public class flow {
            Frame f;
            Button b[];
            flow(){
                        f = new Frame();
                        b = new Button[5];
                        for(int i = 0;i<5;i++)
                        {
                                    b[i] = new Button("B "+(i+1));
                                    f.add(b[i]);
                        }

                        f.setLayout(new FlowLayout(FlowLayout.CENTER));    //3 rows
and 4 columns
                        f.setSize(500,500);  f.setVisible(true);

            }
            public static void main(String args[]) {
                        new flow();

            }
}
```

Java BoxLayout

The BoxLayout is used to arrange the components either vertically or horizontally. For this purpose, BoxLayout provides four constants.

X-AXIS, Y-AXIS, LINE-AXIS, PAGE-AXIS

Image of y-axis

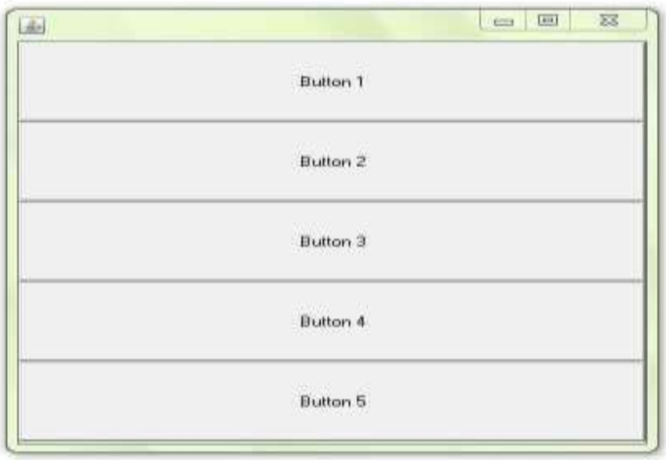
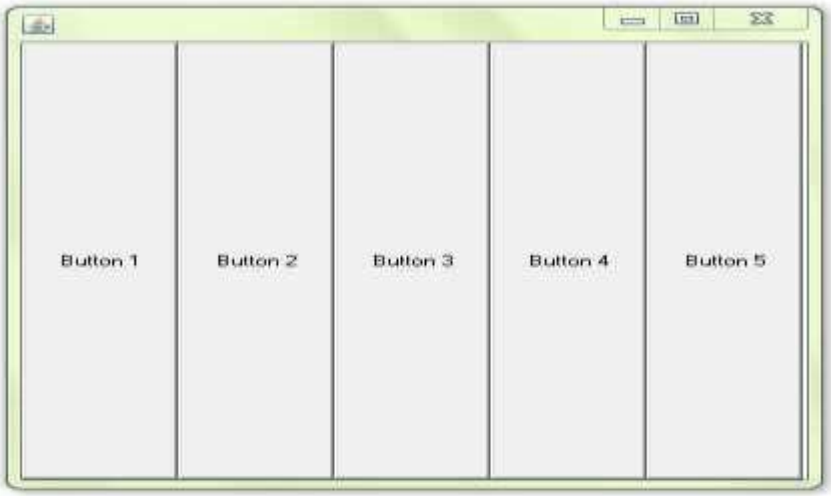


Image of x-axis



Program:

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

public class box extends Frame {
        Button b[];
        box(){
                b = new Button[5];
                for(int i = 0;i<5;i++)
                {
                        b[i] = new Button("B "+(i+1));
                        add(b[i]);
                }

                setLayout(new BoxLayout(this,BoxLayout.PAGE_AXIS));
        //3 rows and 4 columns
//Can be used only with extends frame not with object creation of frame


                setSize(500,500);  setVisible(true);

        }
        public static void main(String args[]) {
                new box();

        }
}
```

Java CardLayout

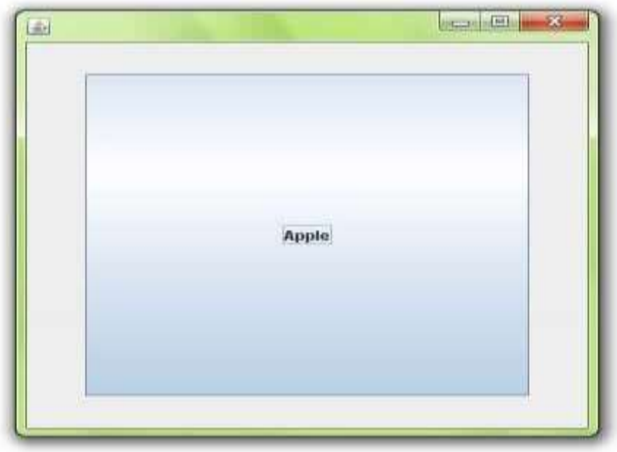
The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout. Only the card at the top of deck is visible at a time.

Java Containers

Containers are integral part of AWT GUI components. A container provides a space where a component can be located. A Container in AWT is a component itself and it adds the capability to add component to itself. Following are noticable points to be considered.

- Sub classes of Container are called as Containter. For example Panel, Frame and Window.
- Container can add only Component to itself.
- A default layout is present in each container which can be overridden using setLayout method.
- Other containers that include windows, frames, dialogs, and panels.

Card Layout only work on containers, not on frames.



Program:

```
import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

public class cardLayout extends JFrame implements ActionListener{
        CardLayout card;
        Container c;
        CardLayout(){
                Card =new CardLayout(40,30);
                C=getContentPane();
                c.setLayout(card);

            Button b1=new Button("Java");
            b1.addActionListener(this);
            add(“a”,b1);

            Button b2=new Button("is");
            b2.addActionListener(this);
            add(“b”,b2);

            Button b3=new Button("Simple");
            b3.addActionListener(this);
            add(“c”,b3);

                setSize(500,500);
                setVisible(true);

        }
        public static void main(String args[]) {
                new cardLayout();

        }
}
```

Java Adapter Classes

Java adapter classes *provide the default implementation of listener* interfaces. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it *saves code*.

The event listener interfaces contain more than one method have a corresponding event adapter class that implements the interface and defines each method in the interface with an empty method body.

The adapter classes are found in **java.awt.event**, **java.awt.dnd** and **javax.swing.event** packages.

java.awt.event Adapter classes

Adapter class	**Listener interface**
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

Uses of Adapter Classes

- Simplifies the creation of an event
- Used to provide the implementation of Listener interfaces
- It saves code.

EventListener interface defines the methods that must be implemented by an event handler for a particular kind of an event.
Event Adapter class provides a default implementation of an **EventListener** interface.

Graphics

The Graphics class is the abstract super class for all graphics contexts which allow an application to draw onto components that can be realized on various devices, or onto off-screen images as well.

A Graphics object encapsulates all state information required for the basic rendering operations that Java supports.

Each component contains a Graphics object which defines a Graphics Context which can be obtained by a call to getGraphics().

Methods:

drawLine	fill3DRect
drawOval	fillArc
drawPolygon	fillPolygon
drawPolyLine	fillRect
drawRect	fillRoundRect
drawRoundRect	setColor
drawString	setFont
draw3DRect	setPaintMode
	drawImage

Program:

```
import java.awt.*;
import java.awt.event.*;

public class mouseadapter extends MouseAdapter{
            Frame f;
            TextArea t1; Label l1; TextField t2;
            mouseadapter()
            {
                        f =  new Frame(“Mouse Listener”);
                        f.setSize(500,500);
                        f.setLayout(null);
                        f.setVisible(true);
                        f.addMouseListener(this);
            }
```

```
        public void mouseClicked(MouseEvent me)
        {
                Graphics  pg = f.getGraphics();
                pg.setColor(Color.RED);
                pg.drawLine(2,2,100,100);
                pg.drawOval(10,80,30,40);
                pg.drawRect(45,45,69,123);
                pg.fillRect(100,80,30,40);
                pg.fillOval(10,80,30,40);
                pg.fill3DRect(200,80,30,40,true);
                pg.drawArc(65,234,50,80,90,10);
                pg.fillArc(65,234,50,80,90,10);
                pg.drawRoundRect(45,78,34,67,57,34);

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

Program 2:

```
import java.awt.*;
import java.awt.event.*;

public class mousead extends MouseMotionAdapter{
        Frame f;
        TextArea t1; Label l1; TextField t2;
        mousead ()
        {
                f =  new Frame("Mouse Motion Listener");
                f.setSize(500,500);
                f.setVisible(true);
                f.addMouseMotionListener(this);
        }

        public void mouseDragged(MouseEvent me)
        {
                Graphics  pg = f.getGraphics();
```

```
                    pg.setColor(Color.RED);
                    pg.drawRect(me.getX(),me.getY(),69,123);
          }
          public static void main(String[] args)
          {
                    mousead ma = new mousead();
          }
}
```

MouseAdapter

- mouseClicked
- mousePressed
- mouseReleased
- mouseEntered
- mouseExited
- mouseDragged

MouseMotionAdapter

- mousemoved
- mouseDragged

Key Adapter

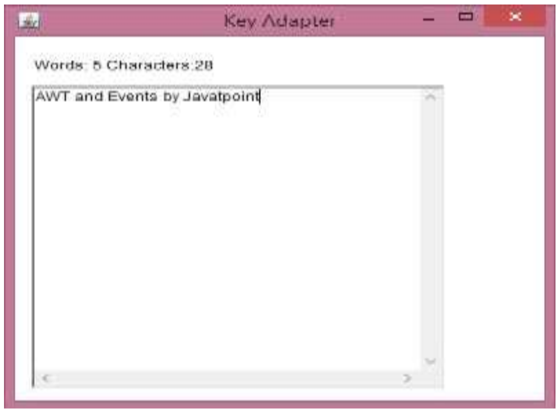
- keypressed
- key Released
- keyTyped

Program:

```
import java.awt.*;
import java.awt.event.*;

public class key extends KeyAdapter {
          Frame f;
```

```
        TextArea ta1;
        TextField t1;
        Label l1;
        key()
        {
                f = new Frame();
                f.setSize(500,500);
                f.setVisible(true);
                f.setLayout(null);

                ta1=new TextArea();
                ta1.setBounds(50,50,200,200);
                f.add(ta1);
                ta1.addKeyListener(this);

                t1=new TextField();
                t1.setBounds(50,300,50,50);
                f.add(t1);

                l1 = new Label("kgfkfgjn" );
                l1.setBounds(50,500,50,50);
                f.add(l1);
        }
        public void keyReleased(KeyEvent e)
        {
                String txt = ta1.getText();
                t1.setText("Words: "+words.length+" Characters:"+text.length());
        }
        public static void main(String[] args) {
        new key();
        }
}
```

Output:



```
Words: 5 Characters:28
AWT and Events by Javatpoint
```

Sources of Events in AWT(all can be clicked)

- Button
- CheckBox
- Choice
- List
- MenuItem
- Text
- Window

Program: DBConnectivity-AWT -Login

```
import java.awt.*;
import java.awt.event.*;

public class AWTDb implements ActionListener{
TextField t1,t2;
Label l1,l2,l3;
Button b;
Frame f;

AWTDb(){
Frame f=new Frame("My Login");

l1=new Label("Enter User ID:");
```

```
l1.setBounds(20,100,100,30);
f.add(l1);

l2=new Label("Enter Password:");
l2.setBounds(20,150,100,30);
f.add(l2);

l3=new Label();
l3.setBounds(20,350,100,30);
f.add(l3);

t1=new TextField();
t1.setBounds(250,100,100,30);
f.add(t1);

t2=new TextField();
t2.setBounds(250,150,100,30);
f.add(t2);
t2.setEchoChar('*');        //for password field

b=newButton("Login");
b.setBounds(250,250,80,30);
f.add(b);
b.addActionListener(this);

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

public void actionPerformed(ActionEvent ae)
{
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","hr","
mca");
```

```
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from login");
String uid=t1.geText();
String pass=t2.getText();
System.out.println(uid);
System.out.println(pass);
while(rs.next()){
if(uid.equals(rs.getString(1)))&&(pass.equals(rs.getString(2)))
{
l3.setText("SUCCESS");
System.out.println("Success");

Dialog d=new Dialog(f,"Success",Dialog.DEFAULT_MODALITY_TYPE);
d.add(new LAbel("GOOD"));
d.setBounds(200,150,200,150);
d.setVisible(true);

break;
else
{l3.setText("FAILURE");
System.out.println("Failure");
}
rs.close();
}catch(Exception e){  System.out.println(e); }
}

public static void main(String args[]){
new AWTDb();
}
```

JAVA SWING

Java Swing is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
2)	AWT components are **heavyweight**.	Swing components are **lightweight**.
3)	AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC**.

Java JProgressBar

The JProgressBar class is used to display the progress of the task. It inherits JComponent class. JProgressBar shows the percentage of completion of specified task.The progress bar fills up as the task reaches it completion.

Program:

```
Import java.awt.*;
Import java.swing.*;

Public class JProgressBar{
Public static void main(String args[]){
Int max=100;
JFrame f=new JFrame("My JProgressBar");
JProgressBar jp=new JProgressBar();
Jp.setMinimum(0);
Jp.setMaximum(max);
Jp.setStringPainted(true);
f.setLayout(new FlowLayout());
f.add(jp);
f.setSize(400,400);
f.setVisible(true);
for(int i-0;i<=max;i++)
{
Try{
Jp.setValue(i);
Thread.sleep(100);
}
Catch(Exception e){ System.out.println(e); }
}
f.dipose();
}
}
```

JFileChooser

JFileChooser is a part of java Swing package. The java Swing package is part of JavaTM Foundation Classes(JFC) . JFC contains many features that help in building graphical user interface in java . Java Swing provides components such as buttons, panels, dialogs, etc. JFileChooser is a easy and an effective dialog window from which the user can choose a file or a directory .

Code:

```
Public void actionPerformed(ActionEvent arg0) {
JFileChooser jc=new JFileChooser("E:");
Jc.setMultiSelectionEnabled(true);
Jc.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
Jc.showOpenDialog(null);

File[] files=jc.getSelectedFiles();
Lb1.setText(jc.getSelectedFile().getAbsolutePath());
For(File x: files)
{
If(x.isDirectory())
{ Sytem.out.println(x.getName()); }
}
}
});
```

JColorChooser

The class **JColorChooser** provides a pane of controls designed to allow a user to manipulate and select a color.

Code:

```
public void actionPerformed(ActionEvent arg0) {
Color jc=JColorChooser.showDialog(null,"Pick any color",Color.WHITE);
//here color is datatype
if(jc==null)
{
jc=Color.WHITE;            //if no color is choosen, this color will be taken by default
}
else
{
//contentPane.setBAckground(jc);
//to give background color for a frame(controlpanel, frame default name )
Button1.setBackground(jc);      // color given to the button1
}
}
});
```

LoginCode:

```
public void actionPerformed(ActionEvent arg0) {
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","hr","
mca");
PreparedStatement Pst=con.prepareStatement("select * from login where id=? and
pass=?");

Pst.setString(1,txtid.getText());
Pst.setString(2,txtpass.getText());

ResultSet rs=pst.executeQuery();

if(rs.next())
{
  System.out.println("Login Successful");
  JOptionPane.showMessageDialog(null,"User Successfully Logged In");
Frame.dispose();
}
Else{
  System.out.println("Invalid Login");
JOptionPane.showMessageDialog(null,"User Invalid");
Frame.dispose();
}
}catch(Exception e){  System.out.println(e);  }
rs.close();
Pst.close();
}
});
```

JTABLE

The JTable class is a part of Java Swing Package and is generally used to display or edit two-dimensional data that is having both rows and columns. It is similar to a spreadsheet. This arranges data in a tabular form.

Java JScrollPane

A JscrollPane is used to make scrollable view of a component. When screen size is limited, we use a scroll pane to display a large component or a component whose size can change dynamically.

JTable ShowDetails Code:

```
public void actionPerformed(ActionEvent arg0) {
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","hr","
mca");
Statement st=con.createStatement();

ResultSet rs=st.executeQuery();
Table.setModel(DbUtils.resultSetToTableModel(rs));
}
}catch(Exception e){  System.out.println(e);  }
}
});
```

Add Details to Table Code:

```
public void actionPerformed(ActionEvent arg0) {
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection(“jdbc:oracle:thin:@localhost:1521:orcl”,”hr”,”
mca”);
Statement st=con.createStatement();

If(id.getText().equals(“”) || name.getText().equals(“”) || age.getText().equals(“”) ||
age.getText.equals(“”) || gen.getText().equals(“”) )
{
        JOPtionPane.showMessageDialog(“Please enter all details”);
}
else
{
String sid=id.getText();
String sname=name.getText();
String sage=age.getText();
String scity=city.getText();
String sgen=String) gen.getSelectedItem();
String sql1="insert into
stu_details(‘”+sid+”’,’”+sname+”’,’”+sage+”’,’”+scity+”’,’”+sgen+”’)”;
St.executeUpdate(sql1);
System.out.println(“Value inserted”);
JOPtionPane.showMessageDialog(null,”Values inserted”);
}
}catch(Exception e){  System.out.println(e);  }
}
});
```

Values from Table to Textmodels Code:

```
Public void mouseClicked(MouseEvent arg0)
{
DefaultTableModel dtm=(DefaultTableModel) table.getModel();

String tid=dtm.getValueAt(table.getSelectedRow(),0).toString();

String tname=dtm.getValueAt(table.getSelectedRow(),1).toString();
String tage=dtm.getValueAt(table.getSelectedRow(),2).toString();

String tcity=dtm.getValueAt(table.getSelectedRow(),3).toString();

String tgen=dtm.getValueAt(table.getSelectedRow(),4).toString();


Id.setText(tid);
Name.setText(tname);
Age.setText(tage);
City.setText(tage);
Gen.setText(tgen);
}
});
```

Edit Code(both on jtable and database):

```
Public void actionPerformed(ActionEvent arg0)
{
String sid=id.getText();
String sname=name.getText();
String sage=age.getText();
String scity=city.getText();
String sgen=String) gen.getSelectedItem();

DefaultTableModel dtm=(DefaultTableModel) table.getModel();

dtm.setValueAt(sid,table.getSelectedRow(),0);
dtm.setValueAt(sname,table.getSelectedRow(),1);
dtm.setValueAt(sage,table.getSelectedRow(),2);
dtm.setValueAt(scity,table.getSelectedRow(),3);
dtm.setValueAt(sgen,table.getSelectedRow(),4);

try{
```

```
String sql=”update stu_details set
                sid=;”+id+”’,sname=’”+name+”’,sage=’”+age+”’,scity=’”+city+”’,sg
                en=’”+gen+”’ where sid=’”+id+”’ ”;
St.executeUpdate(sql);
JOPtionPane.showMessageDialog(null,”Values edited”);
}
Catch(Exception e)
{System.out.println(e);
}
}
});
```

Delete Code:

```
Public void actionPerformed(ActionEvent arg0)
{
try{
String pid=id.getText();
String sql=”Delete from stu_details where sid=’”+tid”’ ”;
st.executeUpdate(sql);
JOPtionPane.showMessageDialog(null,”Values deleted”);
}
catch(Exception e)
{System.out.println(e);
}
}
});
```

Connecting Forms(Flow of forms)

To connect forms, first create object of form2 on form1 code on actionperformed.

Example:

```
Student sd= new Student();
Sd.setVisible(true);
```

Add Images

Put image on the same package of the program.
Create label
Create image icon
Embed imageicon in the label

Code:
ImageIcon im=new ImageIcon(getClass().getResource("images.jpg"));

On label code, add image object on the label object code bracket.
Label lb=new label(im);

Java JRadioButton

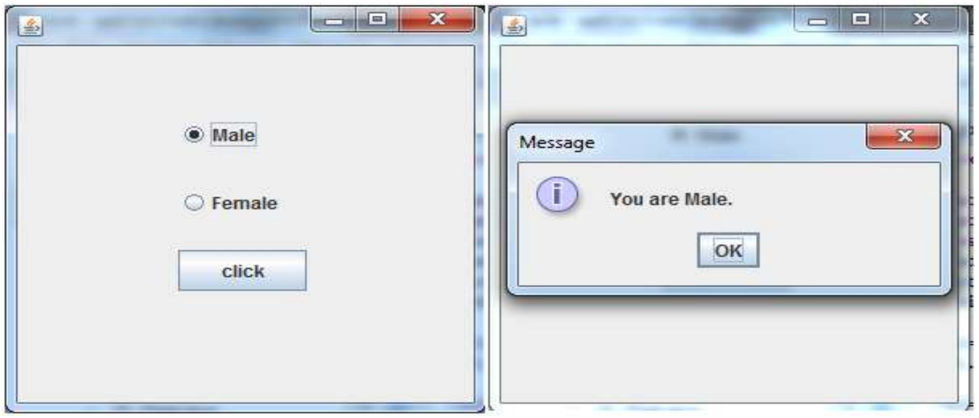
The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.

It should be added in ButtonGroup to select one radio button only.

Code:

```
public void actionPerformed(ActionEvent e){
if(rb1.isSelected()){
JOptionPane.showMessageDialog(this,"You are Male.");
}
if(rb2.isSelected()){
JOptionPane.showMessageDialog(this,"You are Female.");
}
}
```

- JOptionPane.showMessageDialog(null,sel);   //messgedialog
- JOptionPane.showConfirmDialog(null,sel);   //confirmdialog
- JOptionPane.showInputDialog("Enter a value");   //Inputdialog



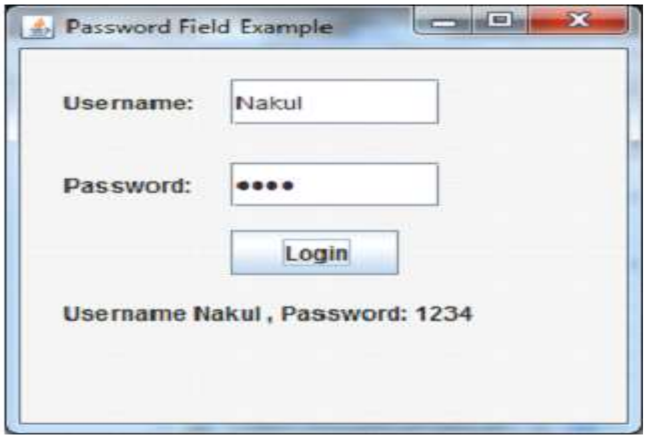
You are Male.

Java JPasswordField

The object of a JPasswordField class is a text component specialized for password entry. It allows the editing of a single line of text. It inherits JTextField class.

Code:

```
public void actionPerformed(ActionEvent e) {
            String data = "Username " + text.getText();
            data += ", Password: "
            + new String(value.getPassword());
            label.setText(data);
          }
        });
```



Username: Nakul

Password: ●●●●

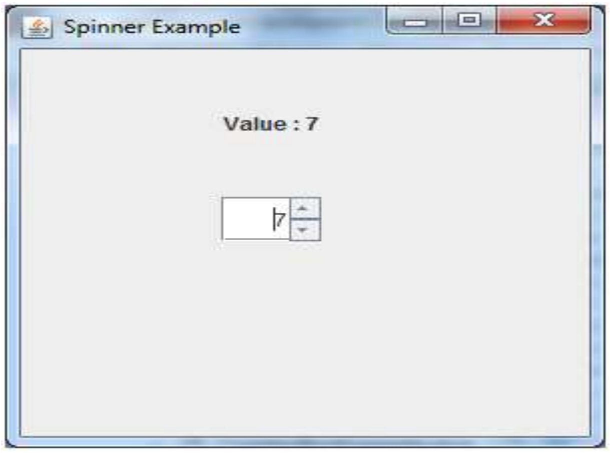
Username Nakul , Password: 1234

Java JSpinner

The object of JSpinner class is a single line input field that allows the user to select a number or an object value from an ordered sequence.

Code:

```
import javax.swing.*;
    import javax.swing.event.*;
    public class SpinnerExample {
        public static void main(String[] args) {
        JFrame f=new JFrame("Spinner Example");
        final JLabel label = new JLabel();
                label.setHorizontalAlignment(JLabel.CENTER);
                label.setSize(250,100);
        SpinnerModel value =
                new SpinnerNumberModel(5, //initial value
                   0, //minimum value
                   10, //maximum value
                   1); //step
        JSpinner spinner = new JSpinner(value);
                spinner.setBounds(100,100,50,30);
                f.add(spinner);  f.add(label);
               f.setSize(300,300);
               f.setLayout(null);
               f.setVisible(true);
               spinner.addChangeListener(new ChangeListener() {
            public void stateChanged(ChangeEvent e) {
             label.setText("Value : " + ((JSpinner)e.getSource()).getValue());
            }
         });
    }
    }
```



Java JSlider

The Java JSlider class is used to create the slider. By using JSlider, a user can select a value from a specific range. The slider can show Major Tick marks and also the minor tick marks between two major tick marks, The knob can be positioned at only those points.

Code:

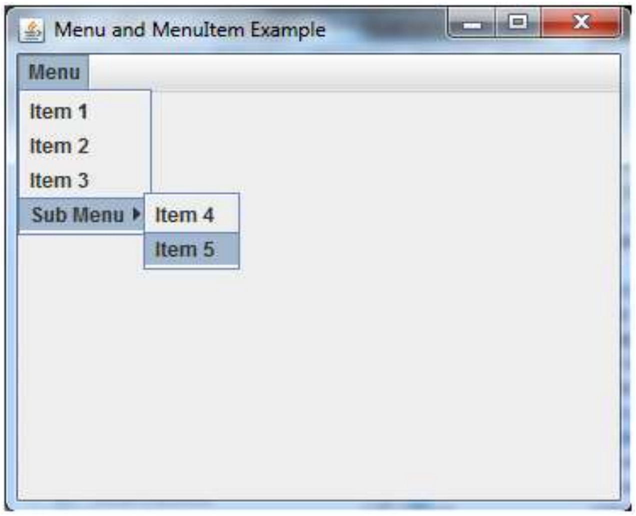
```
l1.setOpaque(true);
Slider.addChangeListener(new ChangeListener() {
    Public void stateChanged(changeEvent e) {
                JSlider js=(JSlider)e.getSource();
                String i=String.valueOf(js.getValue());

                l1.setBackground(new Color(250,js.getVAlue(),50));  }
}
});
```

Java JMenuBar, JMenu and JMenuItem

The JMenuBar class is used to display menubar on the window or frame. It may have several menus.

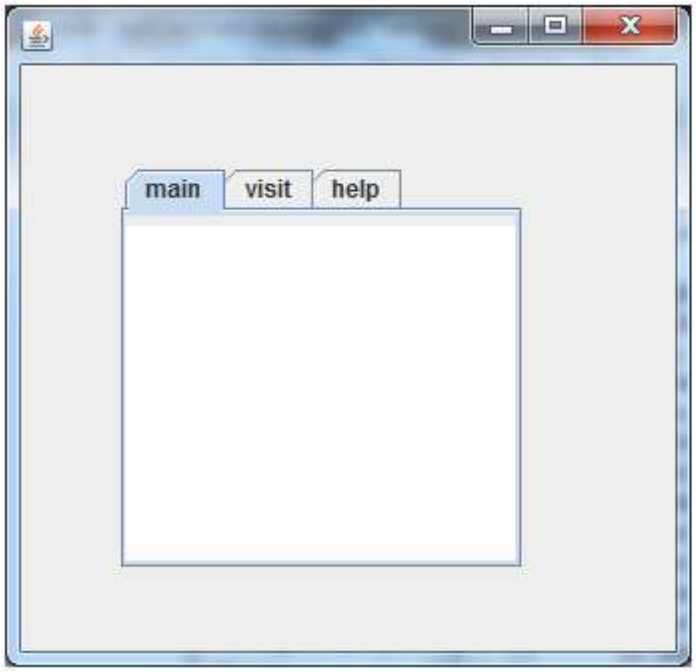
The object of JMenu class is a pull down menu component which is displayed from the menu bar. It inherits the JMenuItem class.

The object of JMenuItem class adds a simple labeled menu item. The items used in a menu must belong to the JMenuItem or any of its subclass.



Java JTabbedPane

The JTabbedPane class is used to switch between a group of components by clicking on a tab with a given title or icon. It inherits JComponent class.



Java Applet

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side. An **applet** is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

- Works at client side
- Fast response time
- Desined to be embedded in a HTML page
- Requires a JVM -> creates an instance of applet class – invokes various applet methods.
- No main method
- When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.
- Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.
- Other classes that the applet needs can be downloaded in a single Java Archive (JAR) file
- Import java.awt.applet.*;

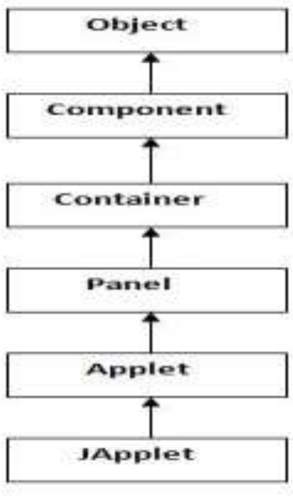
Advantage of Applet

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

Drawback of Applet

- Plugin is required at client browser to execute applet.

Hierarchy of Applet



Applet class extends Panel. Panel class extends Container which is a subclass of Component.

Lifecycle of Java Applet

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.

Applet Lifecycle
1
Applet is initialized.
2
Applet is started.
3
Applet is painted.
4
Applet is stopped.
5
Applet is destroyed.

Implementation

- Using HTML files
- Using ‘Appletviewer’ tools
- Using an IDE

Methods

- **init** − This method is intended for whatever *initialization* is needed for your applet. It is called after the param tags inside the applet tag have been processed.

  public void init(): It is called only once. Once it is called you can see a window.

- **start** − This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.

  public void start(): It is used to start the Applet. It is invoked after the init() method or browser is maximized

- **stop** − This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.

  public void stop(): It is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.

- **destroy** − This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.

  public void destroy(): It is used to destroy the Applet. It is invoked only once.

- **paint** − Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

public void paint(): (Graphics g) is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

Program:

```
import java.applet.*;
import java.awt.*;
/*
<applet code="First.class" width="300" height="300">
</applet>
*/
public class First extends Applet{
public void init()
{
setBackground(Color.yellow);
setForeground(Color.red);
}
public void paint(Graphics g){
g.setFont(new Font("Arial",Font.BOLD,28));
g.drawString("welcome to applet",150,150);
}
}
```

Run on command prompt as:

```
c:\>javac First.java
c:\>appletviewer First.java
```

Displaying Image in Applet

Applet is mostly used in games and animation. For this purpose image is required to be displayed. The java.awt.Graphics class provide a method drawImage() to display the image.

Program:

```
import java.awt.*;
import java.applet.*;
 //ImageIcon →swing..Image--applet

/*<applet code=”DisplayImage” width=”1000” height=”1000”>
</applet>
*/

public class DisplayImage extends Applet {

  Image picture;

public void init() {
setBackground(Color.yellow);
setForeground(Color.red);
}

public void paint(Graphics g) {
picture = getImage(getDocumentBase(),"sonoo.jpg");
g.drawImage(picture, 30,30, this);   //4 args
}
}
```

Adding Audio on Applets

Download any audio clip like .wav file.

Program:

```
import java.awt.*;
import java.applet.*;
```

```
/*<applet code=" DisplayImage" width="1000" height="1000">
</applet>
*/

public class DisplayImage extends Applet implements ActionListener{

  Image picture;
  AudioClip ac;
public void init() {
setBackground(Color.yellow);
setForeground(Color.red);
Button b=new Bjutton("Music");
add(b);
ac=getAudioClip(getCodeBase(),"file_example_WAV_IMG.wav");
b.addActionListener(this);
}

Public void actionPerformed(ActionEvent e)
{
ac.play();
}

public void paint(Graphics g) {
picture = getImage(getDocumentBase(),"sonoo.jpg");
g.drawImage(picture, 30,30, this);   //4 args
}
}
```

Animation –(make images move)

Program:

```
import java.awt.*;
import java.applet.*;
public class AnimationExample extends Applet {

  Image picture;

  public void init() {
    picture =getImage(getDocumentBase(),"bike_1.gif");
  }

  public void paint(Graphics g) {
    for(int i=0;i<=300;i++){
      g.drawImage(picture, i,30, this);

      try{Thread.sleep(100);}
              catch(Exception e){}
    }
  }
}
```

Program 2: Add 2 numbers using applets

```
import java.awt.*;
import java.applet.*;
/*<applet code=”add” wodth=”1000” height=”1000”>
</applet>
*/
public class Add extends Applet {
  TextField t1,t2;
  public void init() {
    setBackground(Color.yellow);
     setForeground(Color.red);
     Button b=new Button("Add");
     t1=new TextField();  add(t1);
      tT2=new TextField();  add(t2);
     add(b);
     b.addActionListener(this);
  }

Public void actionPerformed(ActionEvent e)
{
Int i=Integer.parseInt(t1.getText());
Int j=Integer.parseInt(t2.getText());
Int sum=i+j;
Label l=new Label("");
l.setBounds(50,500,50,50);
add(l);
l.setText(String.valueOf(sum));
}
  public void paint(Graphics g) {
     Image picture =getImage(getCodeBase(), "test.png");
      g.drawImage(picture, i,30, this);
  }
}
```

Parameter Passing in Applets

- **Applet code:**
  Param name="   " value="   "
- **Java Code:**
  getParameter()
    String getParameter(String param_name)

Why used in applets?

- Portability b/w code, html pages, etc.
- You can pass values through web pages into your java program.

Program:

```
import java.applet.Applet;
import java.awt.Graphics;

/*
<applet code=”Param” wodth=”1000” height=”1000”>
<param name=”St_name” value=”Rajitha”>
<param no=”St_RollNo” value=”MSC123”>
<param age=”St_age” value=”23”>
</applet>
*/

public class Param extends Applet{
  String nm,no,age;
Public void init()
{
nm=getParameter(“St_name”);
no=getParameter(“St_RollNo”);
age=getParameter(“St_age”);
}
public void paint(Graphics g){
g.drawString(“Name “+nm,50, 50);
g.drawString(“No“+no,50, 100);
g.drawString(“Age “+age,50, 150);
}
}
```

Program 2:

```
import java.applet.*;
import java.awt.*;

/*
<applet code=”SumParam” wodth=”1000” height=”1000”>
<param name=”A” value=”55”>
<param no=”B” value=”12”>
<param age=”C” value=”23”>
</applet>
*/

public class SumParam extends Applet{
  String a,b,c;
Public void init()
{
a=getParameter(“A”);
b=getParameter(“B”);
c=getParameter(“C”);
//here a,b, and c are strings
}
public void paint(Graphics g){
g.drawString(“A is “+a,50, 50);
g.drawString(“B is “+b,50, 100);
g.drawString(“C is “+c,50, 150);
int ia=Integer.parseInt(a);
int ib=Integer.parseInt(b);
int ic=Integer.parseInt(c);
int sum=ia+ib+ic;
g.drawString(“sum is “+sum,50,250);
}
}
```