Programming in JAVA

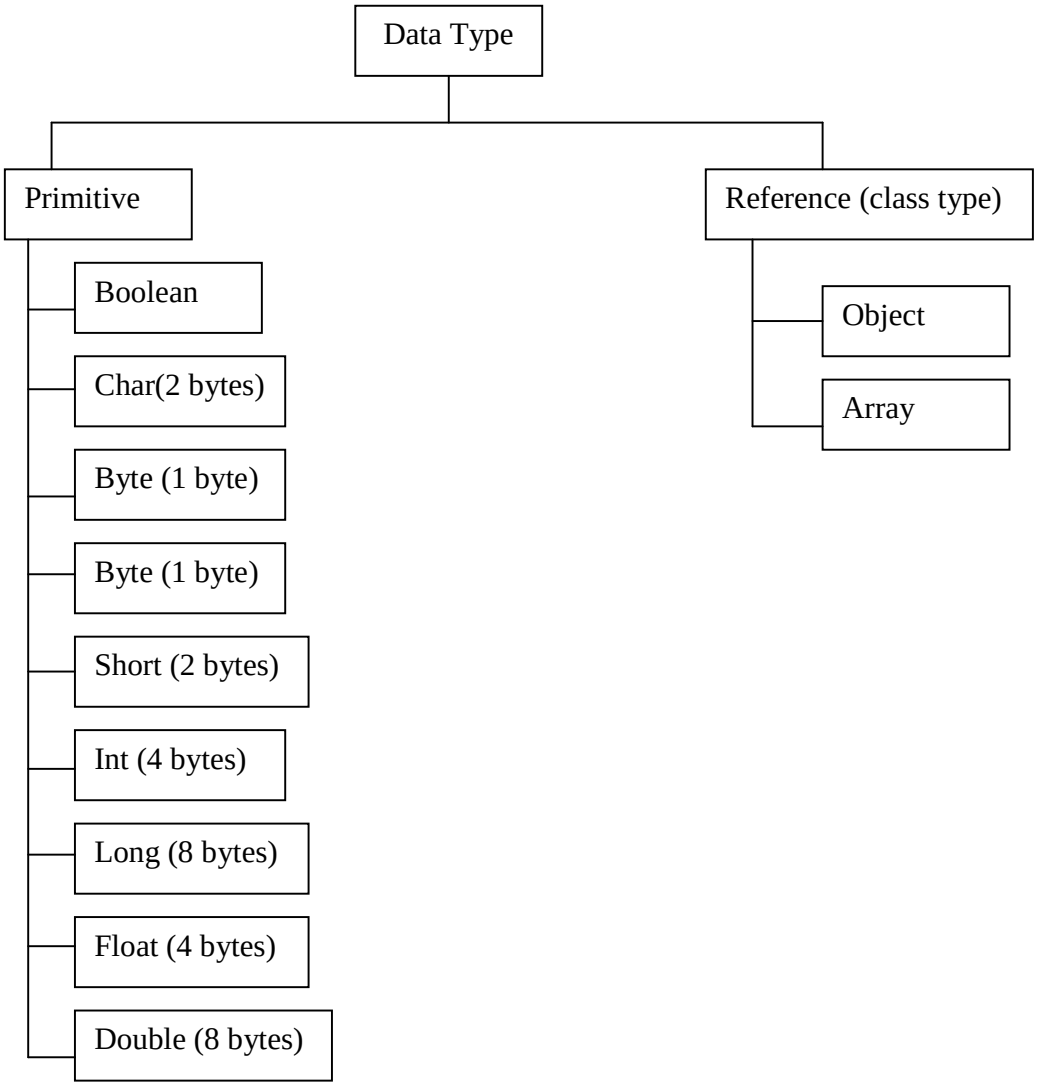
Unit-1

Basic features: - Basic features of Java are

1. Simple : - Java gets its simplicity by omission of some of the complexities such as pointers and memory management.
2. Object-oriented: - It supports object oriented programming
3. Robust: - A robust programme is reliable, running without crashing due to programming errors, erroneous input, or failures of external devices.
4. Security: - Java implements careful measures when communicating across a network with unknown and potentially untrustworthy sources.
5. Architecture neutral and portable: - Being architecture neutral means a compiled java program will run on a variety of processors using various operating systems.
6. Multitasking: - Java threads allow a java program to perform multiple tasks simultaneously.

Java virtual machine concept: - Java is both a compiled and an interpreted language. Java source code is turned into simple binary instructions. Java source is compiled into a universal format – instructions for a virtual machine.

Data types: - Java data types fall into two categories – Primitive type and Reference type.



Variables: - Variables are declared inside of methods or classes. Variables can optionally be initialized with an appropriate expression. Variables that are declared as instance variables in a class are set to default values if they are not initialized. Numeric types default to the appropriate flavor of zero, characters are set to the null character (\0), and Boolean variables have the values false. The syntax is *data type variablename = value*;

Java keyword: - There are 50 keywords defined in the java language. For example abstract, default, goto, if, return, super, native etc.

Java Operator: - An operator is a symbol that are used in programs to compute values and test logical expressions. Java operators can be classified into :

i) Arithmetical operator (+, -, *, /, %)

ii) Assignment operators (=)

iii) Comparison operator (<, >, <=, >=, ==, !=)

iv) Unary operator (++, --)

v) Shift operator (Left shift (<<), Right shift (>>), Right shift with zero fill (>>>)

vi) Bitwise operator (AND(&), OR(|), XOR(^), NOT(~))

vii) Logical operator (AND(&&), OR(||),

viii) Conditional operator (? :)

ix) Instanceof and member selection operator

x) New operator

Instanceof operator: - This operator tests whether an instance derive from a particular class or interface. Instanceof is the reserved word in java. The return type is Boolean. General form is

Object instanceof Class;

Here, Object is an instance of a Class and class is a name of the class type. If object is of the specified type, then instanceof return true, otherwise, its return false. It is also known as runtime operator.

Member selection operator: - The class consists of member variable and member methods. It can be access through the member selection operator (**.**) dot operator. General form is

Object.member variable

Object.member method

new () operator: - When we allocate memory to an object, we must use the new operator. It is in two ways

a) Declare the object and then allocate memory using the new operator. For example

Classname objectname;

Objectname = new classname()

b) Declare the object as well as the allocation of memory in a single step. For example

Classname objectname = new classname();

Class & Object: - Classes are the building blocks of a java application. A class can contain methods, variables, initialization code and even other classes. We can declare a class with *class* keyword.

```
class classname
{
        datatype variablename;
         -----   ------
```

```
        return type methodname(argument list)
        {
            statement block;
        }
}
```

Two types of variables can be defined in a class – instance variables and static variable. Every object has its own set of instance variables. If we don't initialize an instance variable during declaration, then it takes a default value appropriate for its type.

As with variables, methods defined in a class may be instance methods or static methods. An instance method is associated with an instance of the class, but each instance does not really have its own copy of the method

Static members: - Members that are declared with the static modifier live in the class and are shared by all instances of the class. Any modification on static variable in one instance, will affect other instances of the same class. Since static members exist in the class itself, independent of any instance, we can also access them directly through the class.

```
classname.variablename
```

Method: - Method appears inside class bodies. They contain local variable declaration and statement. A method in a java always specifies a return type. There is no default return type.

Static method belongs to the class and not to an individual instance of the class. It can be invoked by name, through the class name, without any objects around. A static method can directly access only other static members of the class. It can not directly see any instance variable or call any instance methods.

Object: - Creating an object of a class in Java is a two-step process. First we must declare a variable of the class type. This variable does not define an object instead, it is simply a variable that can refer to an object. Second, we must acquire an actual, physical copy of the object and assign it to that variable. We can do this using the new operator.

The new operator dynamically allocates memory for an object and returns a reference to it.

```
Classname objectname = new classname( ) ;
```

Array: - An array is a group of similar type of data that are referred to by a common name. Array of any type can be created and may have one or more dimensions. A specific element in an array is accessed by its index. In Java, array index start from zero.

The declaration statement of an array can not allocate any memory space to the array. We must allocate space to array by using new operator.

One-dimensional array: - The general form of one-dimensional array is

```
datatype  arrayname[ ];
arrayname = new datatype [size];
```

Multidimensional array: - In Java, multidimensional arrays are actually arrays of arrays. The general form is

```
datatype  arrayname[ ][ ];
arrayname = new datatype [size][size];
```

String: - A string is a sequence of characters. Java implements strings as objects of type String. After creating a String object, we can not change its contents. If it is necessary to modify, then we have to create a new string object.

Vector: - The Jave.util package has a class called vector. It permits an array to store different elements that are of different class. We need to include the import statement ***import java.util.**** to use the vector class.

It is also an expandable array of objects. The array grows larger as more elements are added to it. The array may also be reduced in size after some of its elements have been deleted. We can not directly store simple(primitive) data type in a vector. We can only store objects. Vector automatically increases its size when needed.

Constructors in vector: - The vector class has three constructors. They are

i) Vector( ): - This constructor creates a vector object with 10 elements initially. Whenever a new element that would exceed this capacity is added, the size of the vector is doubled.

ii) Vector(int size): -This constructor creates a vector whose initial value is specified by size.

iii) Vector(int size, int increment): - This constructor creates a vector whose initial size is specified by size and whose increment is specified by increment value.

Methods in vector: -

i) AddElement( ): - This method is used to add element into the vector at the last position.

ii) elementAt(int i) : - This method returns the object at position I in the vector.

iii) size(): - It returns the number of object present in the vector.

iv) capacity( ): - It gives the number of spaces it has allocated for object references.

v) removeElement(object 0): - This method remove the first occurrence of the object 0 in the vector if and only if its is found.

this keyword: - this keyword has different type of use which are

i) It is used to indicate the implicit object inside a method. For example

```
F1(int a, int b)
{
    this.x = a;
    this.y = b;
}
```

ii) In overloading constructor, sometime it becomes necessary to call one constructor inside another. This is accomplished by using another form of ***this*** keyword. When this( ) is executed, the overloaded constructor that matches the parameter list specified by argument is executed first. Then, if there are any statements inside the original constructor, they are executed. The call to ***this( )*** must be the first statement within the constructor. For example

```
class xyz
{
   int a;
   int b;
   xyz(int i, int j)
   {
         a = i;
         b = j;
   }
   xyz( int j)
   {
         this(j, j);  // call xyz(j,j)
    }
   xyz( )
   {
        this(0); // call xyz(0 , 0)
   }
}
```

Garbage collection: - Java uses a technique known as garbage collection to remove objects that are no longer needed. It finds and watches the object and periodically counts references. When all references to an object are gone, so it's no longer accessible, the garbage-collection mechanism reclaims it and returns the space to the available pool of resources. There is no explicit need to destroy objects.

Finalize method: - Sometimes an object will need to perform some action when it is destroyed. To handle such type of performance, java provides a mechanism called finalization. By using finalization, we can define specific actions that will occur when an object is just about to be reclaimed by the garbage collector.

To add a finalizer to a class, we simply define the ***finalize( )*** method. The java run time calls that method whenever it is about to recycle an object of that class, inside the ***finalize( )*** method, we will specify those actions that must be performed before an object is destroyed. The finalise( ) method is only called just prior to garbage collection.

The general form of finalize( ) method is

```
Protected void finalize( )
{
        statement;
}
```

Inheritance: - Inheritance is the process of creating a new class with the characteristics of an existing class, along with additional characteristics to the new class. The class that is inherited is called a superclass. The class that does the inheriting is called the subclass. To inherit a class we have to use the keyword ***extends.*** Subclass do not inherit the constructor of the superclass.

Although a subclass includes all of the members of its superclass, it cannot access those members of the superclass that have been declared as ***private***.

A superclass variable can reference a subclass object. That superclass variable can only access those member of the subclass which has been inherited from superclass. Not its own members.

Type of inheritance: -

1. Single inheritance: - In this inheritance only one super class and many subclasses.
2. Multilevel inheritance: - In this inheritance a subclass is created from a super class which is again a subclass of another super class.

Super keyword: - Whenever a subclass needs to refer to its immediate superclass, it can do so by use of the keyword super. A subclass can call a constructor defined by it superclass by use of the following form of super super(parameter list);

Another use of the super keyword is to refer to the member of the super class as

super.membername

Multilevel Inheritance: -

Method overriding: - In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its superclass, then the method in the subclass is said to override the method in the superclass. The overriding method may have different operation in the subclass. When an overridden method is called from with a subclass, it will always refer to the version of that method defined by the subclass. The version of the method defined by the superclass will be hidden. If we want to access the base class version of an overridden function, we can use super keyword.

A public method can be overridden only by another public method. Overriding member variable is similar to overriding methods.

Access specifiers: - Access specifiers determined which feature of a class may be used by other classes also. The access specifiers are

i) Public: - The member with public access specifier can be accessible to all other classes. The inner class,(the class declared inside another class) cannot have the public access specifier. The default access specifier is public. *Only one public class can be defined in each source file. The name of the java source file is same name as the public class defined in the .java file*

ii) Private: - The member with private specifier is not accessible to the other class. They can not be inherited to subclass.

iii) Protected: - It is same with private but can be inherited.

Abstract class: - Abstract class is a super class that only defines a generalized form of member which will be shared by all of its subclasses. Each subclass will fill in detail definition of the member. Abstract class can be created by using ***abstract modifier***. The method within the abstract class must be declared with the ***abstract modifier***.

We can not declare any object of abstract class but can create object references. It must be possible to create a reference to an abstract class so that it can be used to point to a subclass object. Also we cannot declare abstract constructor or abstract static methods. The general form of abstract class is

```
abstract class classname
{
    abstract returntype method( )
    {
    }
}
```

The Final keyword: - The keyword final has three uses.

1. It is used to declare a constant.
2. A method declared with final keyword can not be overridden.
3. A class declared with final keyword can not be inherited.

The general form is:

For method

```
Class classname
{
   final returntype method( )
   {
        statement;
   }
}
```

for class

```
final class classname
{
   returntype method( )
   {
        statement;
   }
}
```

Multithreaded Programming: - A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread. Multithreading is a specialized form of multitasking. In java the threads are created by extending from *Thread class*. It is in *java.lang* package.

Threads in java have five characteristics.

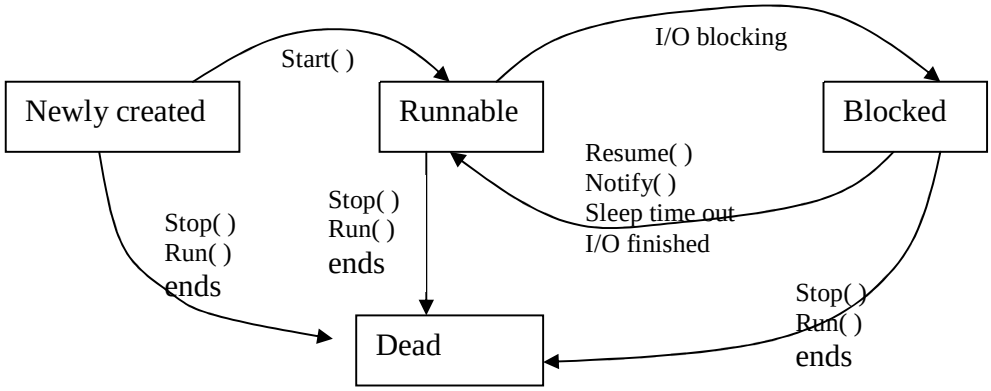
1. **Thread body**: - This is the sequence of instructions for the thread to perform. We create a thread by instantiating an object of type Thread. Java defines two ways in which this can be accomplished:

i) We can implement the Runnable interface: - Runnable abstracts a unit of executable code. To implement Runnable, a class need only implement a single method called run( ).

Instead run( ), we will define the code that constitutes the new thread. This thread will end when run( ) returns.

ii) We can extend the Thread class, itself. : - We can extend the Thread class to create a new thread. Then create a instance (object) of that class. The extending class must override the run( ) method. We must also call start( ) to begin execution of the new thread.

2. **Thread States**: - Every thread, after creation and before destruction, will always be in one of the four states – newly created, runnable, blocked and dead.



i) Newly created: - A thread enters the newly created by using a new operator. It is in the new state or born state immediately after creation.

ii) Runnable: - Once we invoke the start( ) method, the thread is runnable.

iii) Blocked: - The blocked state is entered when one of the following events occurs -

a) when call the suspend( ) method
b) call for wait( ) method
c) waiting for I/O operation
d) the thread will join( ) another thread

iv) Dead: - A thread is dead for any one of the two reasons -

a) it dies a natural death because the run method exits normally
b) it dies abruptly because an uncaught exception terminates the run method.

3. **Thread priority**: - Every thread in java is assigned a priority value. When more than one thread is competing for CPU time, the thread with highest priority value is given preference. We can also use the Thread class constants as follows

Thread.MIN_PRIORITY = 1

Thread.MAX_PRIORITY = 10

Thread.NORM_PRIORITY = 5

4. **Daemon Thread**: - This denotes that a thread is a "server" thread. A server thread is a thread that services client request. They normally enter an endless loop waiting for clients requesting services.

5. **Thread Group**: - Java allows us to group similar threads and manage them as a group. Every thread instance is a member of exactly one thread group.

Methods of Thread class: - The Thread class defines several methods as follows:

1. getName -> Obtain a thread's name
2. getPriority -> Obtain a thread's priority
3. isAlive -> Determine if a thread is still running
4. join -> wait for a thread to terminate
5. run -> Entry point for the thread
6. sleep -> Suspend a thread for a period of time
7. start -> Start f thread by calling its run method.

I/O in Java: -

I/O Basic: -

Streams: - java programs perform I/O through streams. A stream is a flow of data or a channel of communication. A stream is an abstraction that either produces or consumes information. A stream is linked to a physical device by the java I/O system. An input stream can abstract many different kinds of input from a disk file, a keyboard, or a network socket. An output stream may refer to the console, a disk file or a network connection. Java implements streams within class hierarchies defined in the ***java.io*** package.

Byte Streams and Character Streams: - Java defines two types of streams – byte and character.

Byte streams provide a convenient means for handling input and output of bytes. These are used for reading or writing binary data.

Character streams provide a convenient means for handling input and output of characters.

i) The Byte Stream Classes: - Byte streams are defined by using two class hierarchies. At the top are two abstract classes – **InputStream** and **OutputStream.** Each of these abstract classes has several subclasses to handle different devices like disk files, network connection etc. two of the most important methods are read( ) and write( ) for reading and writing bytes of data.

ii) The Character Stream Classes: - Character streams are defined by using two class hierarchies. At the top are two abstract classes, **Reader** and **Writer**. These classes handle Unicode character streams. These classes define several key methods out of which are **read( )** and **write( )** to read and write characters of data.

The Predefined Streams: - All java programs automatically import the **java.lang** package. This package defines a class called System. This system class contains three predefined stream variables – **in**, **out** and **err**. These fields are declared as public, static and final within system.

System.out refers to the standard output stream. By default, this is the console. ***System.in*** refers to standard input, which is the keyboard by default. ***System.err*** refers to the standard error stream, which also is the console by default.

System.in is an object of type InputStream, System.out and System.err are objects of type PrintStream.

Reading Console Input: - In java, console input is accomplished by reading from **System.in**. To obtain a character-based stream that is attached to the console, wrap **System.in** in a **BufferedReader** object. This object supports a buffered input stream. Its most commonly used constructor is

BufferedReader(Reader inputReader)

Here Reader is an abstract class. One of its subclasses is InputStreamReader which converts bytes to characters. To obtain an InputStreamReader object, we have to use the following constructor

InputStreamReader(InputStream *inputStream*)

The main statement is

BufferedReader br = new BufferedReader (new InputStreamReader (System.in));

Reading characters: - Characters from keyboard can be read as follows:

BufferedReader br = new BufferedReader (new InputStreamReader (System.in));

Char c = (char) br.read();

Each time the read( ) is called, it reads a character from the input stream and returns it as an integer value. We are using (char) to convert it to character type.

Reading String: - To read a string from the keyboard, we can use the version of readLine( ) which is the member of the BufferedReader class.

String readLine( ) throws IOException

The statement is :

```
BufferedReader br = new BufferedReader (new InputStreamReader ( System.in));
String str;
Str = br.readLine( );
```

Reading and Writing Files: - Java provides a number of classes and methods that allow to read and write files. In java all files are byte-oriented.

Two of the most often-used stream classes are FileInputStream and FileOutputStream, which create byte streams linked to file. To open a file, we simply create an object of one of these classes, specify the name of the file as an argument to the constructor. The general form is

FileInputStream(String filename) throws FileNotFoundException

FileOutputStream(String filename) throws FileNotFoundxception

Here, filename specifies the name of the file that we want to open. When we create an input stream, if the file does not exist, then FileNotFoundException is thrown. For output streams, if the file can not be created, then FileNotFoundexception is thrown.

When an output file is opened, any pre-existing file by the same name is destroyed.

Reading from File:- To read from a file, we can use a version of read( ) as follows:

Int read( ) throws IOException

Each time that it is called, it reads a single byte from the file and returns the byte as an integer value. The read( ) returns -1 when the end of the file is encountered. The syntax is

```
FileInputStream fin;
        fin = new FileInputStream("filename");
        int x = fin.read( );
```

Writing to file: - To write to a file, we can use the write( ) method as follows

Void write(int byteval)throws IOException

This method writes the byte specified by byteval to the file. Although byteval is declared as an integer, only the low-order eight bits are written to the file. The syntax is

```
FileOutputStream fout;
        fin = new FileOutputStream("filename");
        fout.write(integer value );
```

Closing a file: - We can close a file by using close( ) method. The syntax is

Void close( ) throws IOException;

UNIT – II

Java Applet: - An Applet is a special windows-based program that we can embed in a web page. Applets do not have a main method as an entry point, but instead, have several methods to control specific aspects for applet execution. Every applet that we can create must be a subclass of Applet class. The applet class is contained in the **java.applet** package.

The Applet Class: - The applet class defines all necessary method for applet execution, such as starting and stopping. It also provides methods that load and display images, and methods that load and play audio clips.

Applet extends the AWT (Abstract Window Toolkit) class **Panel** to provide necessary support for windows based activities.

Applet architecture: - An applet is a window-based program. The key concept are:

First, applets are event driven. An applet resembles a set of interrupt service routines. An applet waits until an even occurs. The run-time system notifies the applet about an event by calling an event handler that has been provided by the applet. Once this happens, the applet must take appropriate action and then quickly return.

Second, the user initiates interaction with an applet. These interactions are sent to the applet as events to which the applet must respond.

An Applet Skeleton: -

Initialization and termination: - When an applet begins, the following methods are called in the following sequence –

1. init( ) : This method is the first method to be called. Here we should initialize the variables. This method is called only once during the run time of the applet.
2. start( ): - This method is called after init( ). It is also called to restart an applet after it has been stopped. It is called each time an applet's HTML document is displayed on screen.
3. paint( ): - This method is called each time our applet's output must be redrawn. It is also called when the applet begins execution. It has one parameter of type **Graphics**. This parameter will contain the graphics context, which describes the graphics environment in which the applet is running.

When an applet is terminated, the following sequence of method calls takes place –

1. stop( ): - This method is called when a web browser leaves the HTML document containing the applet or when it goes to another page.
2. destroy( ): - This method is called when the environment determines that our applet needs to be removed completely from memory. At this point, we should free up any resources the applet may be using.

Handling events: -

The HTML APPLET TAG: - The APPLET tag can be used to start an applet from both an HTML document and from an applet viewer. The syntax for a fuller form of the APLET tag is as follows:

```
<       APPLET
        CODEBASE = codebaseURL
        CODE = appletFile
        ALT = alternate Text
        NAME = appletInstanceName
        WIDTH = pixels HEIGHT = pixels
        ALIGN = alignment
        VSPACE = pixel HSPACE = pixels
>
<PARAM NAME = attributeName VALUE = AttributeValue>
<PARAM NAME = AttributeName2 VALUE = AttributeValue>
</APPLET>
```

CODEBASE: - It is an optional attribute that specifies the base URL of the applet code, which is the directory that will be searched for the applet's executable class file specified by the CODE tag.

CODE: - It is a required attribute that gives the name of the file containing our applets's compiled **.class** file.

ALT: - This tag is an optional attribute used to specify a short text message that should be displayed if the browser recognizes the APLET tag but can not currently run java applets.

NAME: - Name is an optional attribute used to specify a name for the applet instance

WIDTH and HEIGHT: - WIDTH and HEIGIT are required attributes that give the size of the applet display area.

ALIGN: - It is an optional attribute that specifies the alignment of the applet.

VSPACE and HSPACE: - These attributes are optional. VSPACE specifies the space above and below the applet. HSPACE specifies the space on each side of the applet.

PARAM NAME and VALUE: - The PARAM tag use to specify applet-specific arguments in an HTML page.

UNIT - III

Networking:

Socket Overview: - Sockets are a low-level programming interface for networked communication. They are at the foundation of modern networking. A socket allows a single computer to serve many different clients at once as well as to serve many different types of information. They send streams of data between applications that may or may not be on the same host. This is accomplished through the use of a port, which is a numberd socket on a particular machine. Socket communication takes place via a protocol.

Java provides different kinds of sockets to support three different distinct class of underlying protocols.

1. Java's basic Socket class, which uses a connection-oriented protocol.
2. The DatagramSocket class, that uses a connectionless protocol.
3. A MulticastSocket that can be used to send data to multiple recipients.

The networking features are support by **java.net** package.

TCP/IP Client Sockets: - TCP/IP sockets are used to implement reliable, bidirectional, persistent, point-to-point, stream-based connections between hosts on the Internet. A socket can be used to connect java's I/O system to other programs that may reside either on the local machine or on any other machine on the Internet.

There are two kinds of TCP sockets in java. One is for servers and the other is for clients. The **ServerSocket** class is designed to be a "listener", which waits for clients to connect before doing anything. Thus **ServerSocket** is for servers. The **Socket** class is for clients. It is designed to connect to server sockets and initiate protocol exchanges.

The creation of a Socket object implicitly establishes a connection between the client and server. The constructors used to create client sockets are:

1. Socket(String hostname, int port) throws UnknownHostException, IOException

   It is used to create socket connected to the name host and port.
2. Socket(InetAddress ipAddress, int port) throws IOException

   It is used to create a socket using a preexisting InetAddress object and a port.

Socket defines several instance methods which are:

1. InetAddress getInetAddress( ) – It returns the InetAddress associated with the Socket object. It returns null if the socket is not connected.
2. int getPort( ) – It returns the remote port to which the invoking Socket object is connected. It returns 0 if the socket is not connected.
3. int getLocalPort( ) – It returns the local port to which the invoking Socket object is bound. It returns -1 if the socket is not bound.

We can gain access to the input and output streams associated with a Socket by use of the getInputStream( ) and getOutputStream( ) methods. Some other methods are:

1. connect( ) – Allows us to specify a new connection.
2. isConnected( ) – return true if the socket is connected to a server
3. isBound( ) – returns true if the socket is bound to an address
4. isClosed( ) – returns true if the socket is closed.

Datagrams: - 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.

Java implements datagrams on top of the UDP protocol by using two classes: -

1. The DatagramPocket object 2. DatagramSocket

1. DatagramSocket: - It defines four public constructors. They are –

i) DatagramSocket( ) throws SocketException – It creates a DatagramSocket bound to any unused port on the local computer.

ii) DatagramSocket(int port) throws SocketException – It creates a DatagramSocket bound to the port specified by port.

iii) DatagramSocket(int port, InetAddress ipAddress) throws SocketException – It constructs a DatagramSocket bound to the specified port and InetAddress.

iv) DatagramSocket(SocketAddress address) throws SocketException – It constructs a DatagramSocket bound to the specified SocketAddress.

DatagramSocket defines many methods which are:

i) void send (DatagramPacket packet) throws IOException – It send a packet to the port specified by packet.

ii) Void receive(DatagramPacket packet)throws IOException – It wait for a packet to be received from the port specified by packet and returns the result

Internet Addressing

DNS

URL

Event Handling: - The modern approach to handling events is based on the *delegation event model*. It defines standard and consistent mechanisms to generate and process events. The main concepts are:

a) Events: - An event is an object that describes a state change in a source. It can be generated as a consequence of a person interacting with the elements in a graphical user interface. Some of the activities that cause events to be generated are pressing a button, entering a character via the keyboard, selecting an item in a list, and clicking the mouse.

b) Event Sources: - A source is an object that generates an event. This occurs when the internal state of that object changes in some way.

A source must register listeners in order for the listeners to receive notifications about a specific type of event. Each type of event has its own registration method. The general form is

Public void addTypeListener(TypeListener el)

c) Event Listeners: - A listener is an object that is notified when an event occurs. It has two major requirements. First, it must have been registered with one or more sources to receive notification about specific type of events. Second, it must implement methods to receive and process these notifications.

UNIT – IV

Java Database Connectivity

Establishing a connection: - Java Database Connectivity (JDBC) is defined as a set of java classes and methods to interface with database. It also provides uniform access to a wide range of relational databases.

We can use JDBC calls to retrieve and update information from a database using JDBC. This involves the following functions:

a) Opening and establish a database connection
b) Send SQL statements
c) Process the returned result set.

d) Close the database connection.

JDBC provides a database programming API for java programs. Java programmes cannot directly communicate with the ODBC driver. Sun Microsystem provides a JDBC-ODBC bridge that translate JDBC to ODBC. There are several type of JDBC drivers which are:

1. JDBC-ODBC bridge+ODBC driver
2. Native API partly java Driver
3. JDBC-Net pure java driver
4. Native protocol pure java driver.

Obtaining a Connection: - The java.sql.Driver class acts as a pass-through driver by forwarding IDBC requests to the real database JDBC Driver. The JDBC driver class is loaded with a call to Class.forName(drivername). The syntax is

Class.forName("sun.jdbc.odbc.JdncOdbcDriver"); or

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

The Connection Object: - It represents a connection with the database. We may have several connection object in an application that connects one or more database. A database conection can be established with a call to the DriverManager.getConnection method.

Connection con = DriverManager.getConnection(url); or

Connection con = DriverManager.getConnection(url, "user", "password");

The Statement Object: - The statement object is created by calling the createStatement() method from the connection object. The statement object is used to execute simple quries. It has three methods that can be used for the purpose of querying.

1. executeQuery(): - This is used to execute simple select query and return a single ResultSet object.
2. executeUpdate(): - This is used to execute SQL, Insert, Update and Delete statements.
3. execute(): - It is used to execute SQL statement that may return multiple values.

Working with ResultSet: - When a data base query is executed, the results of the query are returned as a table of data organized according to rows and columns. The ResultSet interface is to provide access to the table data.

Initially the pointer is positioned before the first row. So we use the next() method to move the cursor to the next row. We can access the data from the ResultSet rows by calling the getXXX() methods, the XXX is the datatype of the parameter such as String, Int etc.

The column in a ResultSet are acesed beginning with index 1.