UNIT-I

DISTRIBUTED PROGRAMMING USING JAVA: QUICK RECAP AND ADVANCED JAVA PROGRAMMING:

Generics in Java

The **Java Generics** programming is introduced in J2SE 5 to deal with type-safe objects. Before generics, we can store any type of objects in collection i.e. non-generic. Now generics, forces the java programmer to store specific type of objects.

Java **Generic** methods and generic classes enable programmers to specify, with a single method declaration, a set of related methods, or with a single class declaration, a set of related types, respectively.

Why Use Generics?

In a nutshell, generics enable *types* (classes and interfaces) to be parameters when defining classes, interfaces and methods. Much like the more familiar *formal parameters* used in method declarations, type parameters provide a way for you to

re-use the same code with different inputs. The difference is that the inputs to formal parameters are values, while the inputs to type parameters are types.

Advantage of Java Generics:

There are many advantages of generics. They are as follows:

1) Type-safety: We can hold only a single type of objects in generics. It doesn't allow to store other objects.

2) Type casting is not required(Individual Type Casting is not needed): There is no need to typecast the object.

Before Generics, we need to type cast. Programs that uses Generics has got many benefits over non-generic code. If we do not use generics, then, in the above example every-time we retrieve data from ArrayList, we have to typecast it. Typecasting at every retrieval operation is a big headache. If we already know that our list only holds string data then we need not to typecast it every time.

Before Generics, we use Typecasting

```
1.    List list = new ArrayList( );
2.    list.add("hello");
3.    String s = (String) list.get(0);  //typecasting
```

After Generics, we don't need to typecast the object

```
1.    List<String> list = new ArrayList<String>( );
2.    list.add("hello");
3.    String s = list.get(0);
```

3) Compile-Time Checking : It is checked at compile time so problem will not occur at runtime. The good programming strategy says it is far better to handle the problem at compile time than runtime.

Generics make errors to appear compile time than at run time (It's always better to know problems in your code at compile time rather than making your code fail at run time).

```
1.  List<String> list = new ArrayList<String>( );
2.  list.add("hello");
3.  list.add(32);        //Compile Time Error
```

4) Code Reuse: We can write a method/class/interface once and use for any type we want.

GENERIC METHODS:

Like generic class, we can create generic method that can accept any type of argument. Using Java Generic concept, we might write a generic method for sorting an array of objects,

then invoke the generic method with Integer arrays, Double arrays, String arrays and so on, to sort the array elements.

You can write a single generic method declaration that can be called with arguments of different types. Based on the types of the arguments passed to the generic method, the compiler handles each method call appropriately. Following are the rules to define Generic Methods –

· All generic method declarations have a type parameter section delimited by angle brackets (< and >) that precedes the method's return type (< E > in the next example).

· Each type parameter section contains one or more type parameters separated by commas. A type parameter, also known as a type variable, is an identifier that specifies a generic type name.

· The type parameters can be used to declare the return type and act as placeholders for the types of the arguments passed to the generic method, which are known as actual type arguments.

· A generic method's body is declared like that of any other method. Note that type parameters can represent only reference types, not primitive types (like int, double and char).

Example:

Following example illustrates how we can print an array of different type using a single Generic method –

LIST:

A **List** or **sequence** is an abstract data type that represents a countable number of ordered values, where the same value may occur more than once.

```
public class GenericMethodTest {
  // generic method printArray
  public static < E > void printArray( E[  ] inputArray ) {
     // Display array elements
     for(E element : inputArray) {
        System.out.printf("%s ", element);
     }
```

```
    System.out.println( );
  }

  public static void main(String args[ ]) {
    // Create arrays of Integer, Double and Character
    Integer[ ] intArray = { 1, 2, 3, 4, 5 };
    Double[ ] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
    Character[ ] charArray = { 'H', 'E', 'L', 'L', 'O' };

    System.out.println("Array integerArray contains:");
    printArray(intArray);   // pass an Integer array

    System.out.println("\nArray doubleArray contains:");
    printArray(doubleArray);   // pass a Double array

    System.out.println("\nArray characterArray contains:");
    printArray(charArray);   // pass a Character array
  }
}
```

This will produce the following result −

Output

```
Array integerArray contains:
1 2 3 4 5

Array doubleArray contains:
1.1 2.2 3.3 4.4

Array characterArray contains:
H E L L O
```

Type Parameters

The type parameters naming conventions are important to learn generics thoroughly. The commonly type parameters are as follows:

1. T - Type
2. E - Element
3. K - Key
4. N - Number
5. V – Value

Let's see a simple example of java generic method to print array elements. We are using here **E** to denote the element.

```
1.  public class TestGenerics4{
2.      public static < E > void printArray(E[ ] elements) {
3.        for ( E element : elements){
4.          System.out.println(element );
5.        }
6.        System.out.println( );
7.      }
8.      public static void main( String args[ ] ) {
9.        Integer[ ] intArray = { 10, 20, 30, 40, 50 };
10.       Character[ ] charArray = { 'J', 'A', 'V', 'A', 'T','P','O','I','N','T' };
11.
12.       System.out.println( "Printing Integer Array" );
13.       printArray( intArray  );
14.
15.       System.out.println( "Printing Character Array" );
16.       printArray( charArray );
17.     }
```

```
18.         }
```

Output:

```
Printing Integer Array
10
20
30
40
50
Printing Character Array
J
A
V
A
T
P
O
I
N
T
```

Generic class:

A class that can refer to any type is known as generic class. Here, we are using **T** type parameter to create the generic class of specific type.

Let's see the simple example to create and use the generic class.

Creating generic class:

```
1.      class MyGen<T> {
2.      T obj;
3.      void add(T obj){    this.obj=obj;   }
4.      T get( ){
```

```
5.          return obj;
6.          } }
```

The T type indicates that it can refer to any type (like String, Integer, Employee etc.). The type you specify for the class, will be used to store and retrieve the data.

Using generic class:

Let's see the code to use the generic class.

```
1.          class TestGenerics3{
2.          public static void main(String args[  ]){
3.          MyGen<Integer> m=new MyGen<Integer>( );
4.          m.add(2);
5.          System.out.println (m.get( ));
6.          }  }
```

Output:

```
2
```

Full Example of Generics in Java using ArrayList class

Here, we are using the ArrayList class, but you can use any collection class such as ArrayList, LinkedList, HashSet, TreeSet, HashMap, Comparator etc.

```
1.          import java.util.*;
2.          class TestGenerics1 {
3.          public static void main(String args[ ]) {
4.          ArrayList<String> list=new ArrayList<String>( );
5.          list.add("rahul");
6.          list.add("jai");
7.          String s=list.get(1);          //type casting is not required
8.          System.out.println("element is: "+s);
9.          Iterator<String> itr=list.iterator();
10.        while(itr.hasNext()){
11.        System.out.println(itr.next());
```

```
12.         }
13.         } }
```

Output:

element is: jai

rahul

jai

Example of Java Generics using Map class

- A **Map** is also called **associative array** or **symbol table**or **dictionary** is an abstract data type composed of a collection of **<key,value>** pairs, such that each possible key appears at most once in the collection.
- A **set** is an abstract **data** type that can store certain values, without any particular order, and no repeated values. An abstract **data structure** is a collection, or aggregate, of **data**.
- In computer science, an **abstract data type** (**ADT**) is a mathematical model for **data types.**
- Now we are going to use map elements using generics. Here, we need to pass key and value. Let us understand it by a simple example:

```
1.          Import  java.util.*;
2.          class TestGenerics2{
3.          public static void main(String args[  ]) {
4.          Map<Integer,String> map = new HashMap <Integer, String> ( );
5.          map.put(1,"vijay");
6.          map.put(4,"umesh");
7.          map.put(2,"ankit");
8.          //Now use Map.Entry for Set and Iterator
9.          Set<Map.Entry<Integer,String>>     set =  map.entrySet( );
10.       Iterator<Map.Entry<Integer,String>>     itr = set.iterator( );
11.       while( itr.hasNext (  ) ) {
12.       Map.Entry e=itr.next( );          //no need to typecast
13.       System.out.println( e.getKey (  )+" "+e.getValue (  ) );
```

```
14.         } } }
```

Output:

1 vijay

2 ankit

4 umesh

THREADS

Introduction:

The **java.lang.Thread** class is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently.Following are the important points about Thread:

· Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority

· Each thread may or may not also be marked as a daemon.

· There are two ways to create a new thread of execution. One is to declare a class to be a subclass of Thread and,the other way to create a thread is to declare a class that implements the Runnable interface

Class declaration:

Following is the declaration for **java.lang.Thread** class:

```
public class Thread
 extends Object
 implements Runnable
```

Following are the fields for **java.lang.Thread** class:

· **static int MAX_PRIORITY** -- This is the minimum priority that a thread can have.

· **static int NORM_PRIORITY** -- This is the default priority that is assigned to a thread.

Life Cycle of Thread

A thread can be in any of the five following states

1. Newborn State: When a thread object is created a new thread is born and said to be in Newborn state.

2. Runnable State: If a thread is in this state it means that the thread is ready for execution and waiting for the availability of the processor. If all threads in queue are of same priority then they are given time slots for execution in round robin fashion

3. Running State: It means that the processor has given its time to the thread for execution. A thread keeps running until the following conditions occurs

a. Thread give up its control on its own and it can happen in the following situations

i. A thread gets suspended using **suspend( )** method which can only be revived with **resume( )** method

ii. A thread is made to sleep for a specified period of time using s**leep(time)** method, where time in milliseconds

iii. A thread is made to wait for some event to occur using **wait ( )**method. In this case a thread can be scheduled to run again using**notify ( )** method.

b. A thread is pre-empted by a higher priority thread

4. Blocked State: If a thread is prevented from entering into runnable state and subsequently running state, then a thread is said to be in Blocked state.

5. Dead State: A runnable thread enters the Dead or terminated state when it completes its task or otherwise terminates.

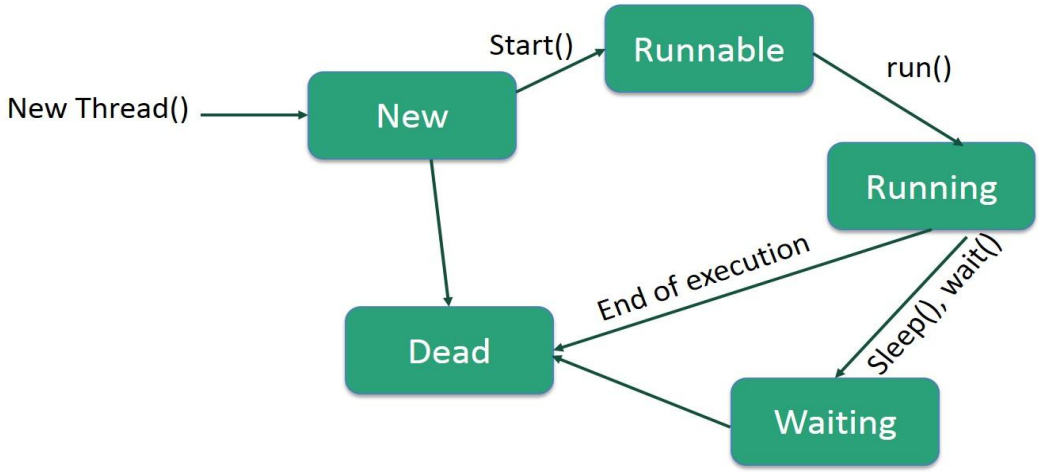

Fig: Life Cycle of Thread

Main Thread

Every time a Java program starts up, one thread begins running which is called as the *main thread* of the program because it is the one that is executed when your program begins.

- Child threads are produced from main thread
- Often it is the last thread to finish execution as it performs various shut down operations

Creating a Thread

Java defines two ways in which this can be accomplished:

- You can implement the Runnable interface.
- You can extend the Thread class, itself.

Create Thread by Implementing Runnable

The easiest way to create a thread is to create a class that implements the Runnable interface. To implement Runnable, a class need only implement a single method called **run( )**, which is declared like this:

public void run( )

You will define the code that constitutes the new thread inside **run( )**method. It is important to understand that **run( )** can call other methods, use other classes, and declare variables, just like the main thread can.

After you create a class that implements Runnable, you will instantiate an object of type Thread from within that class. Thread defines several constructors. The one that we will use is shown here:

Thread(Runnable threadOb, String threadName);

Here threadOb is an instance of a class that implements the Runnable interface and the name of the new thread is specified by threadName. After the new thread is created, it will not start running until you call its **start( )** method, which is declared within Thread. The start( ) method is shown here:

void start( );

Class constructors:

S.N.	Constructor & Description
1	**Thread( )** This allocates a new Thread object.
2	**Thread(Runnable target)** This allocates a new Thread object.
3	**Thread(Runnable target, String name)** This allocates a new Thread object.
4	**Thread(String name)** This constructs allocates a new Thread object.
5	**Thread(ThreadGroup group, Runnable target)** This allocates a new Thread object.
6	**Thread(ThreadGroup group, Runnable target, String name)** This allocates a new Thread object so that it has target as its run object, has the specified name

	name, and belongs to the thread group referred to by group.
7	**Thread(ThreadGroup group, Runnable target, String name, long stackSize)** This allocates a new Thread object so that it has target as its run object, has the specified name name, belongs to the thread group referred to by group, and has the specified stack size.
8	**Thread(ThreadGroup group, String name)** This allocates a new Thread object.

SOCKETS

What Is a Socket?

Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.

On the client-side: The client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.

If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.

On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server.

The client and server can now communicate by writing to or reading from their sockets.

Definition:

A *socket* is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.

- An endpoint is a combination of an IP address and a port number. Every TCP connection can be uniquely identified by its two endpoints. That way you can have multiple connections between your host and the server.
- The java.net package in the Java platform provides a class, Socket, that implements one side of a two-way connection between your Java program and another program on the network. The Socket class sits on top of a platform-dependent implementation, hiding the details of any particular system from your Java program. By using the java.net.Socket class instead of relying on native code, your Java programs can communicate over the network in a platform-independent fashion.
- Additionally, java.net includes the ServerSocket class, which implements a socket that servers can use to listen for and accept connections to clients. This lesson shows you how to use the Socket and ServerSocket classes.
- If you are trying to connect to the Web, the URL class and related classes (URLConnection, URLEncoder) are probably more appropriate than the socket classes. In fact, URLs are a relatively high-level connection to the Web and use sockets as part of the underlying implementation. See Working with URLs for information about connecting to the Web via URLs.
- Java Socket programming is used for communication between the applications running on different JRE.
- Java Socket programming can be connection-oriented or connection-less.
- Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

The client in socket programming must know two information:

1. IP Address of Server, and
2. Port number.

Reading from and Writing to a Socket

- Let's look at a simple example that illustrates how a program can establish a connection to a server program using the Socket class and then, how the client can send data to and receive data from the server through the socket.
- The example program implements a client, EchoClient, that connects to an echo server. The echo server receives data from its client and echoes it back. The example EchoServer implements an echo server. (Alternatively, the client can connect to any host that supports the Echo Protocol.)
- The EchoClient example creates a socket, thereby getting a connection to the echo server. It reads input from the user on the standard input stream, and then forwards that text to the echo server by writing the text to the socket. The server echoes the input back through the socket to the client. The client program reads and displays the data passed back to it from the server.
- Note that the EchoClient example both writes to and reads from its socket, thereby sending data to and receiving data from the echo server.
- Let's walk through the program and investigate the interesting parts. The following statements in the try-with-resources statement in the EchoClient example are critical. These lines establish the socket connection between the client and the server and open a PrintWriter and a BufferedReader on the socket:

```
String hostName = args[0];
Int portNumber = Integer.parseInt(args[1]);
try (
        Socket echoSocket = new Socket(hostName, portNumber);
        PrintWriter out =
        new PrintWriter(echoSocket.getOutputStream( ), true);
        BufferedReader in = new BufferedReader(
        new InputStreamReader(echoSocket.getInputStream( )));
```

BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in))

)

- The first statement in the try-with resources statement creates a new Socket object and names it echoSocket. The Socketconstructor used here requires the name of the computer and the port number to which you want to connect.
- The example program uses the first command-line argument as the name of the computer (the host name) and the second command line argument as the port number.
- When you run this program on your computer, make sure that the host name you use is the fully qualified IP name of the computer to which you want to connect.

  For example, if your echo server is running on the computer echoserver.example.comand it is listening on port number 7, first run the following command from the computer echoserver.example.com if you want to use the EchoServer example as your echo server: **java EchoServer 7**

Afterward, run the EchoClient example with the following command:

java EchoClient echoserver.example.com 7

- The second statement in the try-with resources statement gets the socket's output stream and opens a PrintWriter on it. Similarly, the third statement gets the socket's input stream and opens a BufferedReader on it. The example uses readers and writers so that it can write Unicode characters over the socket.
- To send data through the socket to the server, the EchoClient example needs to write to the PrintWriter. To get the server's response, EchoClient reads from the BufferedReader object stdIn, which is created in the fourth statement in the try-with resources statement. If you are not yet familiar with the Java platform's I/O classes, you may wish to read Basic I/O.
- The next interesting part of the program is the while loop. The loop reads a line at a time from the standard input stream and immediately sends it to the server by writing it to the PrintWriter connected to the socket:

```
String userInput;
while ((userInput = stdIn.readLine( )) != null) {
   out.println(userInput);
   System.out.println("echo: " + in.readLine( ));
}
```

- The last statement in the while loop reads a line of information from the BufferedReader connected to the socket. The readLinemethod waits until the server echoes the information back to EchoClient. When readline returns, EchoClient prints the information to the standard output.
- The while loop continues until the user types an end-of-input character. That is, the EchoClient example reads input from the user, sends it to the Echo server, gets a response from the server, and displays it, until it reaches the end-of-input. (You can type an end-of-input character by pressing **Ctrl-C**.)
- The while loop then terminates, and the Java runtime automatically closes the readers and writers connected to the socket and to the standard input stream, and it closes the socket connection to the server.
- The Java runtime closes these resources automatically because they were created in the try-with-resources statement. The Java runtime closes these resources in reverse order that they were created. (This is good because streams connected to a socket should be closed before the socket itself is closed.)
- This client program is straightforward and simple because the echo server implements a simple protocol. The client sends text to the server, and the server echoes it back. When your client programs are talking to a more complicated server such as an HTTP server, your client program will also be more complicated. **However, the basics are much the same as they are in this program:**

1. Open a socket.
2. Open an input stream and output stream to the socket.
3. Read from and write to the stream according to the server's protocol.

4. Close the streams.

5. Close the socket.

Only step 3 differs from client to client, depending on the server. The other steps remain largely the same.

Socket class :

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

Important methods

Method	Description
1)public InputStream getInputStream( )	returns the InputStream attached with this socket.
2) public OutputStream getOutputStream( )	returns the OutputStream attached with this socket.
3) public synchronized void close( )	closes this socket

ServerSocket class:

The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.

Important methods

Method	Description
1) public Socket accept( )	returns the socket and establish a connection between server and client.
2) public synchronized void close( )	closes the server socket.

Simple client-server programming using java:

Let's see a simple of java socket programming in which client sends a text and server receives it.

File: MyServer.java

```
1.    import java.io.*;
2.    import java.net.*;
3.    public class MyServer {
4.    public static void main(String[  ] args){
5.    try{
6.    ServerSocket ss=new ServerSocket(6666);
7.    Socket s=ss.accept( );                    //establishes connection
8.   DataInputStream dis=new DataInputStream(s.getInputStream( ));
9.    String  str=(String)dis.readUTF( );
10.  System.out.println("message= "+str);
11.  ss.close( );
12.  }catch(Exception e){System.out.println(e);}
13.  }
14.  }
```

File: MyClient.java

```
1.    import java.io.*;
2.    import java.net.*;
3.    public class MyClient {
4.    public static void main(String[  ] args) {
5.    try{
6.    Socket s=new Socket("localhost",6666);
7.   DataOutputStream dout=new DataOutputStream(s.getOutputStream( ));
8.    dout.writeUTF("Hello Server");
9.    dout.flush( );
10.  dout.close( );
11.  s.close( );
12.  }
13.  catch(Exception e) {
```

```
14.     System.out.println(e);
15.     }
16.     }
17.     }
```

DIFFICULTIES IN DEVELOPING DISTRIBUTED PROGRAMS FORLARGE SCALE CLUSTERS

The World Wide Web is used by millions of people everyday forvarious purposes including email, reading news, downloading music, onlineshopping or simply accessing information about anything.

Using a standardweb browser, the user can access information stored on Web serverssituated anywhere on the globe.

Generally, distributed system is defines as "a system in which hardware or software components located at networked computers communicate andcoordinate their actions only by message passing".

Tanenbaum defines distributed systems "A collection of independent computers that appear to the users of the system as a single computer" Leslie Lamport is a famous researcher on timing, message ordering, andclock synchronization in distributed systems once said that "A distributedsystem is one on which I cannot get any work done because some machine Ihave never heard of has crashed" reflecting on the huge number of challengesfaced by distributed system designers.

Despite these challenges, the benefits of distributed systems andapplications are many, making it worthwhile to pursue. Various types ofdistributed systems and applications have been developed and are beingused extensively in the real world.

Various kinds of distributed systems operate today, each aimed at solvingdifferent kinds of problems. The challenges faced in building a distributedsystem vary depending on the requirements of the system. In general,however, most systems will need to handle the following issues

- **Heterogeneity:**

Various entities in the system must be able to interoperate with oneanother, despite differences in hardware architectures, operating systems,communication protocols, programming languages, software interfaces,security models, and data formats.

· **Transparency:**

The entire system should appear as a single unit and the complexityand interactions between the components should be typically hidden fromthe end user.

· **Fault tolerance and failure management:**

Failure of one or more components should not bring down the entire system,and should be isolated.

· **Scalability:**

The system should work efficiently with increasing number of users andaddition of a resource should enhance the performance of the system.

· **Concurrency:**

Shared access to resources should be made possible. · Openness and Extensibility Interfaces should be cleanly separated and publicly available to enableeasy extensions to existing components and add new components.

· **Migration and load balancing:**

Allow the movement of tasks within a system without affecting the operationof users or applications, and distribute load among available resourcesfor improving performance.

· **Security:**

Access to resources should be secured to ensure only known users are able to perform allowed operations.

INTRODUCTION TO CLOUD COMPUTING

An Overview

- Cloud computing is a computing paradigm, where a large pool ofsystems are connected in private or

public networks, to providedynamically scalable infrastructure for application, data and fil estorage. With

the advent of this technology, the cost of computation,application hosting, content storag e and delivery is reducedsignificantly.

- Cloud computing is a practical approach to experience direct costbenefits and it has th e potential to transform a data center from a capital-intensive set up to a variable priced environment.
- The idea of cloud computing is based on a very fundamentalprincipal of „reusability of I T capabilities'. The difference thatcloud computing brings compared to traditional co ncepts of"grid computing","distributed computing", "utility computing", or"autonomic computing" is to broaden horizons across organizational boundaries.

Cloud computing is defined as

"A pool of abstracted, highly scalable, and managed compute infrastructure capable of hosting end-customer applications and billed by consumption."

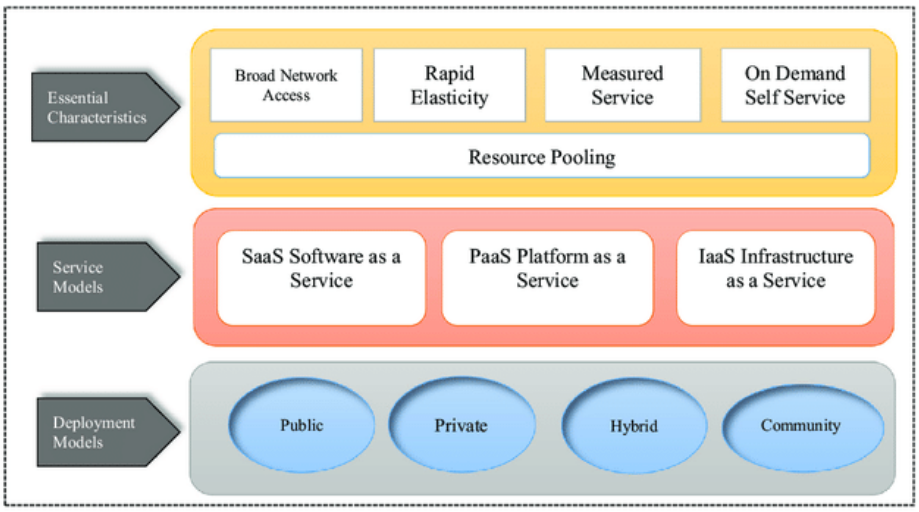

Figure 1: *Conceptual view of cloud computing*

Cloud Computing Models

Cloud Providers offer services that can be grouped into three categories.

1. **Software as a Service (SaaS):** In this model, a completeapplication is offered to the customer, as a service on demand. Asingle instance of the service runs on the cloud & multiple endusers

are serviced. On the customers‟ side, there is no need forupfront investment in servers o r software licenses, while for theprovider, the costs are lowered.

since only a single application needs to be hosted & maintained. Today SaaS is offered b ycompanies such as Google, Salesforce, Microsoft, Zoho, etc.

2. **Platform as a Service (Paas):** Here, a layer of software, ordevelopment environment is encapsulated & offered as a service,upon which other higher levels of service can be built. The customer has the freedom to build his own applications,which run on the provi der‟s infrastructure. To meetmanageability and scalability requirements of the applications ,PaaS providers offer a predefined combination of OS andapplication servers, such as LAMP pl atform (Linux, Apache, MySqland PHP), restricted J2EE, Ruby etc. Google‟s App Engine,Force. com, etc are some of the popular PaaS examples.
3. **Infrastructure as a Service (Iaas):** IaaS provides basic storageand computing capabilities as standardized services over thenetwork. Servers, storage systems, networking equipment, dat a centre space etc. are pooled and made available to handleworkloads. The customer wo uld typically deploy his own softwareon the infrastructure. Some common examples are Am azon, GoGrid, 3 Tera, etc.

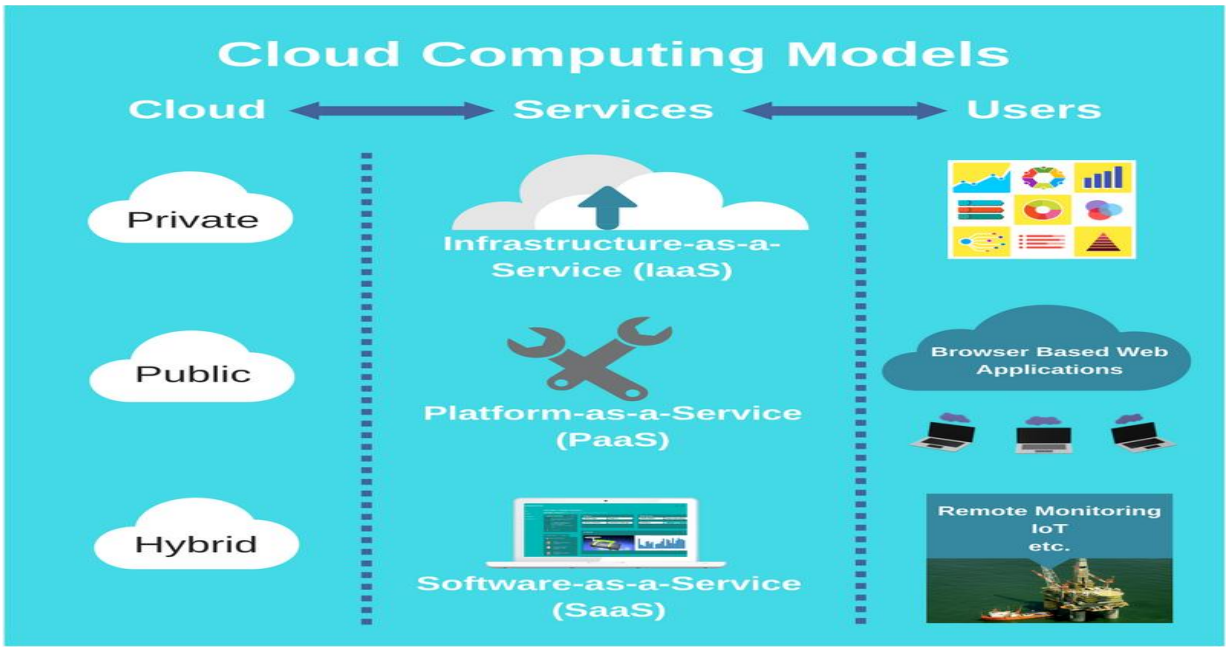

Figure 2: *Cloud*

Understanding Public and Private Clouds

Enterprises can choose to deploy applications on Public,Private or Hybrid clouds. Cloud In tegrators can play a vital partin determining the right cloud path for each organization.

1. **Public Cloud:**

Public clouds are owned and operated by third parties;they deliver superior economies of scale to customers, as the infrastructure costs are spread among a mix of users, givingeach indi vidual client an attractive low-cost, "Pay-as-you go"model. All customers share the same infrastructure pool withlimited configuration, security protections, and availability variances. These are managed and supported by the cloudprovider. One of the advantages of a Public cloud is that theymay be larger than an enterprises cloud, thus providing the ability to scale seamlessly, on demand.]

2. **Private Cloud:**

Private clouds are built exclusively for a single enterprise.They aim to address concerns on data security and offergreater control, which is typically lacking in a public cloud.There are two variations to a private cloud:

- **On-premise Private Cloud:** On-premise private clouds,also known as internal clouds are hosted within one‟sown data center. This model provides a more standardized process and protection, but is limited inaspects of size and scalability. IT de partments would alsoneed to incur the capital and operational costs for the physical resources. This is best suited for applications which require complete control and configurability of the infrastructure and security.
- **Externally hosted Private Cloud:** This type of privatecloud is hosted externally with a cloud provider, wherethe provider facilitates an exclusive cloud environment with full guarantee of privacy. This is best suited forenterprises that don‟t prefer a public cloud due to sharingof physical resources.

3. **Hybrid Cloud:**

Hybrid Clouds combine both public and private cloud models. With a Hybrid Cloud, service providers can utilize 3rd party Cloud Providers in a full or partial manner thus increasingthe flexibility of computing. The Hybrid cloud environment is capable of providing on demand, exte

rnally provisioned scale.The ability to augment a private cloud with the resources of a public cloud can be used to manage any unexpected surges inworkload.

Cloud Computing Benefits :

Enterprises would need to align their applications, so as toexploit the architecture models that Cloud Computing offers.Some of the typical benefits are listed below:

1. **Reduced Cost**

There are a number of reasons to attribute Cloudtechnology with lower costs. The billing model is pay asper usage; the infrastructure is not purchased thuslowering maintenance. Initial expense and recurringexpenses are much lower than traditional computing.

2. **Increased Storage**

With the massive Infrastructure that is offered by Cloudproviders today, storage & maintenance of large volumesof data is a reality. Sudden workload spikes are also managed effectively & efficiently, since the cloud canscale dynamically.

3. **Flexibility**

This is an extremely important characteristic. Withenterprises having to adapt, even more rapidly, tochanging business conditions, speed to deliver iscritical. Cloud computing stresses on getting applicationsto market very quickly, by using the most appropriate building blocks necessary for deployment.

Cloud Computing Challenges

Despite its growing influence, concerns regarding cloudcomputing still remain. In our opinion, the benefits outweighthe drawbacks and the model is worth exploring. Some common challenges are:

1. **Data Protection**

Data Security is a crucial element that warrants scrutiny.Enterprises are reluctant to buy an assurance ofbusiness data security from vendors. They fear losingdata to competition and the data confidentiality ofconsumers. In many instances, the actual storagelocation is not disclosed, adding onto the securityconcerns of enterprises. In

the existing models, firewallsacross data centers (owned by enterprises) protect this sensitive information. In the cloud model, Serviceproviders are responsible for maintaining data security and enterprises would have to rely on them.

2. Data Recovery and Availability

All business applications have Service level agreementsthat are stringently followed. Operational teams play akey role in management of service level agreements and runtime governance of applications. In production environments, operational teams support Appropriate clustering and Fail over Data Replication System monitoring (Transactions monitoring,logs monitoring and others) Maintenance(Runtime Governance) Disaster recovery Capacity and performance management.

If, any of the above mentioned services is under-servedby a cloud provider, the damage & impact could be severe.

3. Management Capabilities

Despite there being multiple cloud providers, themanagement of platform and infrastructure is still in itsinfancy. Features like „Auto-scaling" for example, are a crucial requirement for many enterprises. There is hugepotential to improve on the scalability and load balancing features provided today.

4. Regulatory and Compliance Restrictions

In some of the European countries, Governmentregulations do not allow customer's personal informationand other sensitive information to be physically locatedoutside the state or country. In order to meet suchrequirements, cloud providers need to setup a datacenter or a storage site exclusively within the country tocomply with regulations. Having such an infrastructuremay not always be feasible and is a big challenge for cloud providers.