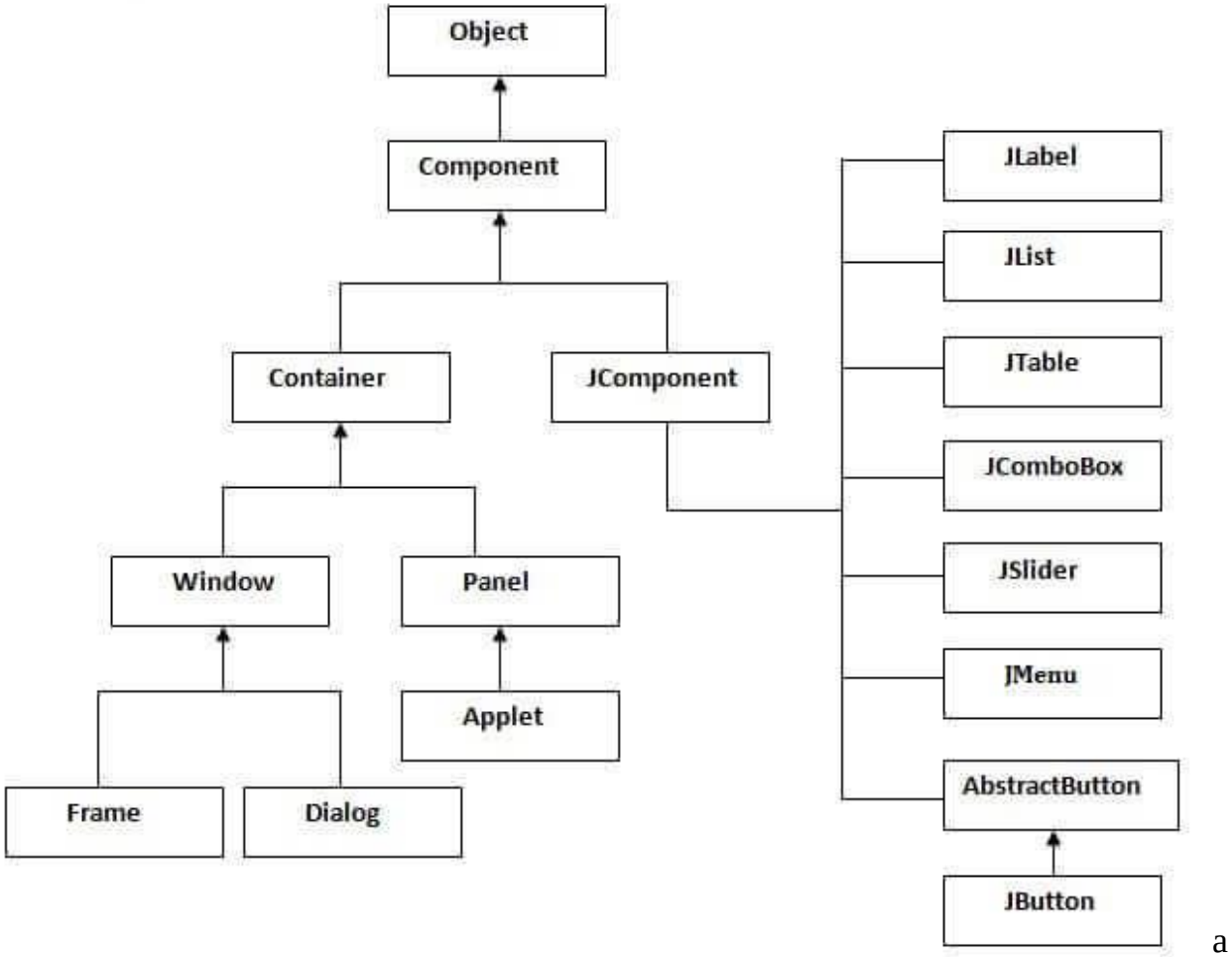
UNIT - IV

Swing in Java is a lightweight GUI toolkit which has a wide variety of widgets for building optimized window based applications. It is a part of the JFC(Java Foundation Classes). It is build on top of the AWT API and entirely written in java. It is platform independent unlike AWT and has lightweight components.

The Java Swing library is built on top of the Java Abstract Widget Toolkit (**AWT**), an older, platform dependent GUI toolkit. You can use the Java simple GUI programming components like button, textbox, etc., from the library and do not have to create the components from scratch.



a

Panel is to organize the components on to a window.

2. **Frame**: It is a fully functioning window with its title and icons.
3. **Dialog**: It can be thought of like a pop-up window that pops out when a message has to be displayed. It is not a fully functioning window like the Frame.

What is GUI in Java?

GUI (Graphical User Interface) in Java is an easy-to-use visual experience builder for Java applications. It is mainly made of graphical components like buttons, labels, windows, etc. through which the user can interact with an application. GUI plays an important role to build easy interfaces for Java applications.

How to Make a GUI in Java with Example

Now in this Java GUI Tutorial, let's understand how to create a GUI in Java with Swings in Java examples.

Step 1) Copy code into an editor

In first step Copy the following code into an editor.

```
import javax.swing.*;
class gui{
   public static void main(String args[]){
      JFrame frame = new JFrame("My First GUI");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setSize(300,300);
      JButton button = new JButton("Press");
      frame.getContentPane().add(button); // Adds Button to content pane of frame
      frame.setVisible(true);
   }
}
```

Java Layout Manager

The Layout manager is used to layout (or arrange) the GUI Java components inside a container. There are many layout managers, but the most frequently used are-

Java BorderLayout

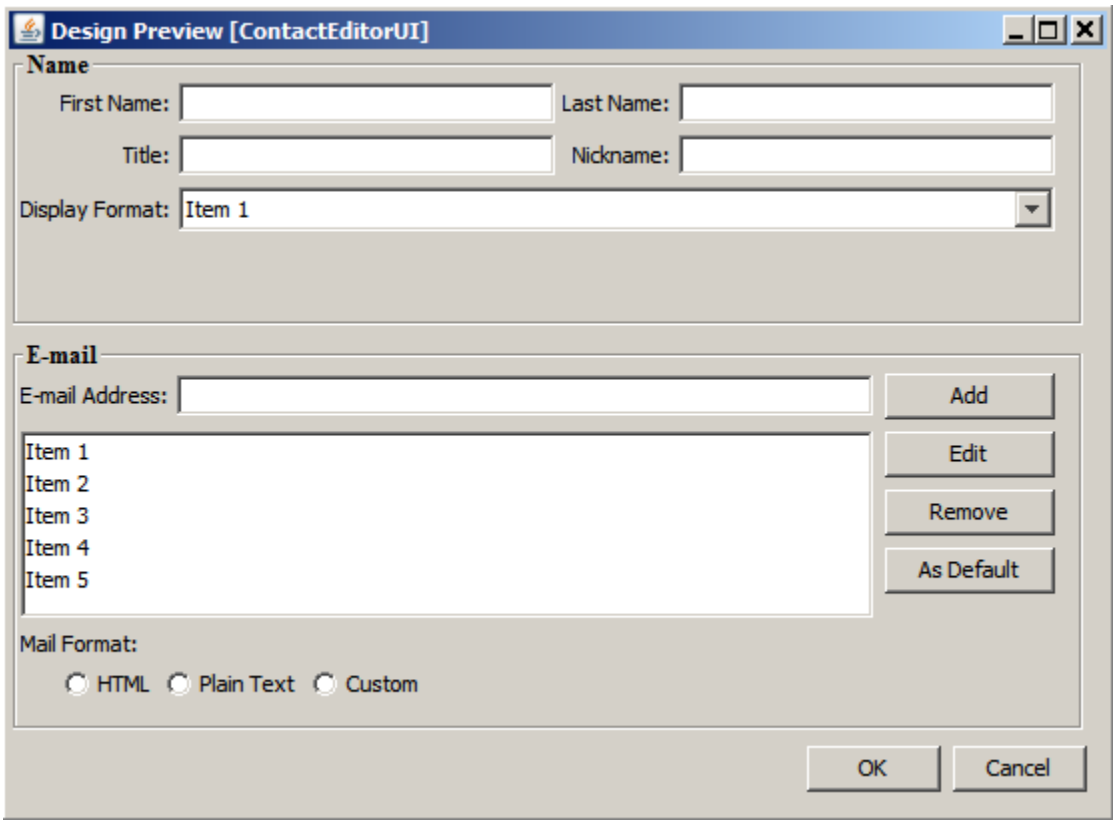
A `BorderLayout` places components in up to five areas: top, bottom, left, right, and center. It is the default layout manager for every java JFrame

Java FlowLayout

`FlowLayout` is the default layout manager for every `JPanel`. It simply lays out components in a single row one after the other.

Java GridBagLayout

It is the more sophisticated of all layouts. It aligns components by placing them within a grid of cells, allowing components to span more than one cell.



IO STREAMS AND OBJECT IN JAVA

An I/O Stream represents an input source or an output destination. A stream can represent many different kinds of sources and destinations, including disk files, devices, other programs, and memory arrays.
Java IO or **Input Output** in **Java** with input **stream**, output **stream**, reader and writer class. The **java.io** package provides api to reading and writing data.

Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.

We can perform **file handling in Java** by Java I/O API.

Stream

A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.

In Java, 3 streams are created for us automatically. All these streams are attached with the console.

1) System.out: standard output stream

2) System.in: standard input stream

3) System.err: standard error stream

Let's see the code to print **output and an error** message to the console.

```
System.out.println("simple message");
System.err.println("error message");
```

Let's see the code to get **input** from console.

```
int i=System.in.read();//returns ASCII code of 1st character
System.out.println((char)i);//will print the character
```

Do You Know?

OutputStream vs InputStream

The explanation of OutputStream and InputStream classes are given below:

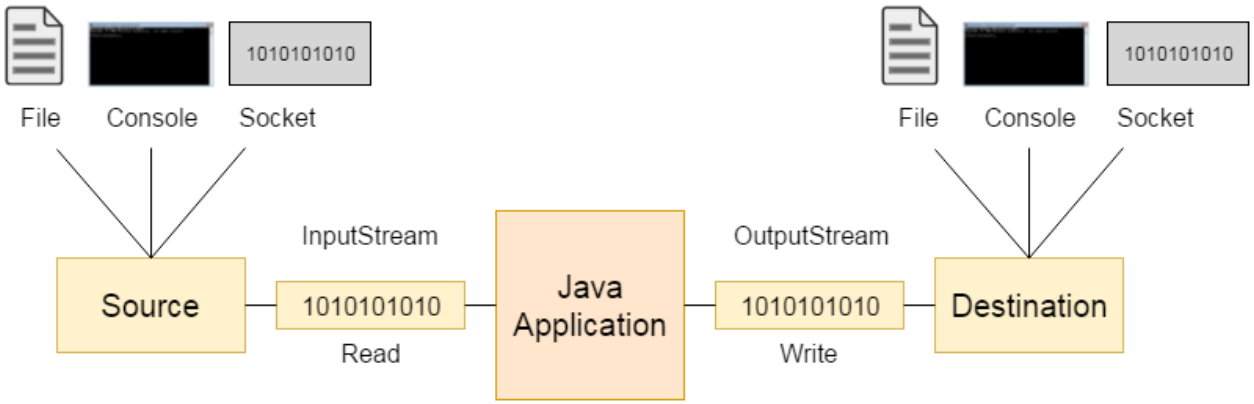
OutputStream

Java application uses an output stream to write data to a destination; it may be a file, an array, peripheral device or socket.

InputStream

Java application uses an input stream to read data from a source; it may be a file, an array, peripheral device or socket.

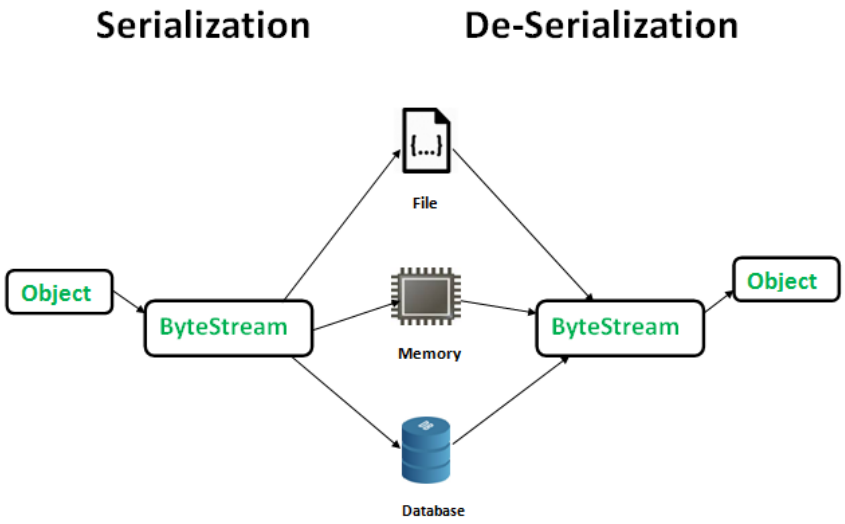
Let's understand the working of Java OutputStream and InputStream by the figure given below.



SERIALIZATION

Serialization in Java is **the concept of representing an object's state as a byte stream**. The byte stream has all the information about the object. Usually used in Hibernate, JMS, JPA, and EJB, serialization in Java helps transport the code from one JVM to another and then de-serialize it there.

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.



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. The ObjectOutputStream class contains **writeObject()** method for serializing an Object.

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 **Serializable** interface must be implemented by the class whose object needs to be persisted.

The String class and all the wrapper classes implement the *java.io.Serializable* interface by default.

Let's see the example given below:

import java.io.Serializable;

public class Student **implements** Serializable

ObjectOutputStream class

The ObjectOutputStream class is used to write primitive data types, and Java objects to an OutputStream.

Constructor

1) public ObjectOutputStream(OutputStream out) throws IOException {}	It creates an ObjectOutputStream that writes to the specified OutputStream.

Important Methods

Method	Description
1) public final void writeObject(Object obj) throws IOException {}	It writes the specified object to the ObjectOutputStream.
2) public void flush() throws IOException {}	It flushes the current output stream.
3) public void close() throws IOException {}	It closes the current output stream.

ObjectInputStream class

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

Constructor

1) public ObjectInputStream(InputStream in) throws IOException {}	It creates an ObjectInputStream that reads from the specified InputStream.

Important Methods

Method	Description
1) public final Object readObject() throws IOException, ClassNotFoundException{}	It reads an object from the input stream.
2) public void close() throws IOException {}	It closes ObjectInputStream.

Generics in Java

The **Java Generics** programming is introduced in J2SE 5 to deal with type-safe objects. It makes the code stable by detecting the bugs at compile time.

Before generics, we can store any type of objects in the collection, i.e., non-generic. Now generics force the java programmer to store a specific type of objects.

Advantage of Java Generics

There are mainly 3 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.

Without Generics, we can store any type of objects.

```
1. List list = new ArrayList();
2. list.add(10);
3. list.add("10");
4. With Generics, it is required to specify the type of object we need to store.
5. List<Integer> list = new ArrayList<Integer>();
6. list.add(10);
7. list.add("10");// compile-time error
```

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

Before Generics, we need to type cast.

```
1. List list = new ArrayList();
2. list.add("hello");
3. String s = (String) list.get(0);//typecasting
4. After Generics, we don't need to typecast the object.
5. List<String> list = new ArrayList<String>();
6. list.add("hello");
7. 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.

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

Syntax to use generic collection

```
1. ClassOrInterface<Type>
```

Example to use Generics in java

1. ArrayList<String>

Java Concurrency

Java Concurrency package covers **concurrency, multithreading**, and **parallelism** on the Java platform. Concurrency is the ability to run several or multi programs or applications in parallel. The **backbone** of Java concurrency is threads (a lightweight process, which has its own files and stacks and can access the shared data from other threads in the same process). java.util.concurrent package.

Main Components/Utilities of Concurrent Package

1. Executor
2. ExecutorService
3. ScheduledExecutorService
4. Future
5. CountDownLatch
6. CyclicBarrier
7. Semaphore
8. ThreadFactory
9. BlockingQueue
10. DelayQueue
11. Lock
12. Phaser

Life cycle of a Thread (Thread States)

A thread is **a thread of execution in a program**. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently. Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority.

In Java, a thread always exists in any one of the following states. These states are:

1. New
2. Active
3. Blocked / Waiting
4. Timed Waiting
5. Terminated

Explanation of Different Thread States

New: Whenever a new thread is created, it is always in the new state. For a thread in the new state, the code has not been run yet and thus has not begun its execution.

Active: When a thread invokes the start() method, it moves from the new state to the active state. The active state contains two states within it: one is **runnable**, and the other is **running**.

- **Runnable:** A thread, that is ready to run is then moved to the runnable state. In the runnable state, the thread may be running or may be ready to run at any given instant of time.
- **Running:** When the thread gets the CPU, it moves from the runnable to the running state. Generally, the most common change in the state of a thread is from runnable to running and again back to runnable.

Blocked or Waiting: Whenever a thread is inactive for a span of time (not permanently) then, either the thread is in the blocked state or is in the waiting state.
Timed Waiting: Sometimes, waiting for leads to starvation.
Terminated: A thread reaches the termination state because of the following reasons:

- When a thread has finished its job, then it exists or terminates normally.
- **Abnormal termination:** It occurs when some unusual events such as an unhandled exception or segmentation fault.

A terminated thread means the thread is no more in the system. In other words, the thread is dead, and there is no way one can respawn (active after kill) the dead thread.

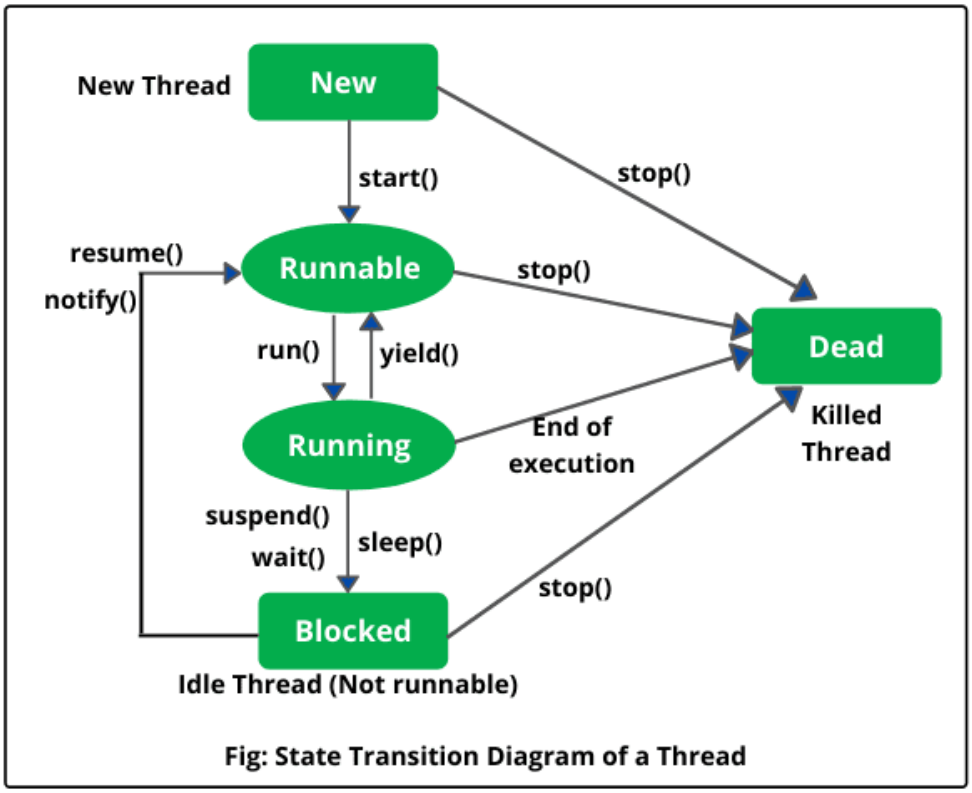


Fig: State Transition Diagram of a Thread

Synchronization in Java

Synchronization in Java is the capability to control the access of multiple threads to any shared resource.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

Why use Synchronization?

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

Types of Synchronization

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

Here, we will discuss only thread synchronization.

Thread Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive
    1. Synchronized method.
    2. Synchronized block.
    3. Static synchronization.
2. Cooperation (Inter-thread communication in java)

Mutual Exclusive

Mutual Exclusive helps keep threads from interfering with one another while sharing data. It can be achieved by using the following three ways:

1. By Using Synchronized Method
2. By Using Synchronized Block
3. By Using Static Synchronization

Concept of Lock in Java

Synchronization is built around an internal entity known as the lock or monitor. Every object has a lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.

From Java 5 the package java.util.concurrent.locks contains several lock implementations.

Java Networking

Java Networking is a concept of connecting two or more computing devices together so that we can share resources.

Java socket programming provides facility to share data between different computing devices.

Advantage of Java Networking

1. Sharing resources
2. Centralize software management

The java.net package supports two protocols,

1. **TCP:** Transmission Control Protocol provides reliable communication between the sender and receiver. TCP is used along with the Internet Protocol referred as TCP/IP.
2. **UDP:** User Datagram Protocol provides a connection-less protocol service by allowing packet of data to be transferred along two or more nodes

The java.net package supports two protocols,

1. **TCP:** Transmission Control Protocol provides reliable communication between the sender and receiver. TCP is used along with the Internet Protocol referred as TCP/IP.
2. **UDP:** User Datagram Protocol provides a connection-less protocol service by allowing packet of data to be transferred along two or more nodes

1. IP Address
2. Protocol

3. Port Number
4. MAC Address(**Media Access Control address)**
5. Connection-oriented and connection-less protocol
6. Socket

1) IP Address

IP address is a unique number assigned to a node of a network e.g. 192.168.0.1 . It is composed of octets that range from 0 to 255.

It is a logical address that can be changed.

2) Protocol

A protocol is a set of rules basically that is followed for communication. For example:

- TCP(**Transmission Control Protocol)**
- FTP(File Transfer Protocol)
- Telnet(Teletype Network,)
- SMTP(**Simple Mail Transfer Protocol**.)
- POP(**Point of Presence Or Post Office Protocol**.) etc.

3) Port Number

The port number is used to uniquely identify different applications. It acts as a communication endpoint between applications.

The port number is associated with the IP address for communication between two applications.

4) MAC Address

MAC (Media Access Control) address is a unique identifier of NIC (Network Interface Controller). A network node can have multiple NIC but each with unique MAC address.

For example, an ethernet card may have a **MAC** address of 00:0d:83::b1:c0:8e.

5) Connection-oriented and connection-less protocol

In connection-oriented protocol, acknowledgement is sent by the receiver. So it is reliable but slow. The example of connection-oriented protocol is TCP.

But, in connection-less protocol, acknowledgement is not sent by the receiver. So it is not reliable but fast. The example of connection-less protocol is UDP.

6) Socket

A socket is an endpoint between two way communications.

Visit next page for Java socket programming.

java.net package

The java.net package can be divided into two sections: (**Application Programming Interface)**

1. **A Low-Level API:** It deals with the abstractions of addresses i.e. networking identifiers, Sockets i.e. bidirectional data communication mechanism and Interfaces i.e. network interfaces.
2. **A High Level API:** It deals with the abstraction of URIs i.e. Universal Resource Identifier, URLs i.e. Universal Resource Locator, and Connections i.e. connections to the resource pointed by URLs.