

# Module-I

## Assignment-1

### Problem Definition:

Design a system with the help of advance data structures in Java and enhance the system using collections and generics.

### 1.1 Prerequisite:

1. Java and Advance Java Programming.
2. Data Structures.

### 1.2 Learning Objectives:

1. Understand the concepts of advance Java.
2. Implementation of data structures.

### 1.3 Theory

#### 1.3.1 Introduction

This article will offer a brief overview of data structures, including what they do, how they work, why users need to be aware of them, and what users can do to with implementation of any software.

#### 1.3.2 What is Data Structure?

A data structure is a particular way of organizing data in a computer so that it can be used effectively. The idea is to reduce the space and time complexities of different tasks. The data structures provided by the Java utility package are very powerful and perform a wide range of functions. These data structures consist of the following interface and classes-

1. Enumeration
2. BitSet
3. Vector
4. Stack
5. Dictionary
6. Hashtable
7. Properties

All these classes are now legacy and Java-2 has introduced a new framework called Collections Framework.

#### 1.4 Need of Data Structures.

It is the same intuitions that led to the rise of data structures and algorithms to arrange and process the data in the memory for efficient storage, access and processing. Data structure is a particular way of storing and organizing information in a computer so that it can be retrieved and used most productively.

#### 1.5 Interfaces and Classes in Java Data Structures.

##### 1) The Enumeration

The Enumeration interface isn't itself a data structure, but it is very important within the context of other data structures. The Enumeration interface defines a means to retrieve successive elements from a data structure. The Enumeration interface defines the methods by which you can enumerate (obtain one at a time) the elements in a collection of objects.

This legacy interface has been super-ceded by Iterator. Although not deprecated, Enumeration is considered obsolete for new code. However, it is used by several methods defined by the legacy classes such as Vector and Properties, is used by several other API classes, and is currently in widespread use in application code.

The methods declared by Enumeration

| Sr. No | Method Description  |
|--------|---|
| 1      | boolean hasMoreElements( )- When implemented, it must return true while there are still more elements to extract, and false when all the elements have been enumerated. |
| 2      | Object nextElement( )- This returns the next object in the enumeration as a generic Object reference.   |

## 2) The BitSet

The BitSet class implements a group of bits or flags that can be set and cleared individually. This class is very useful in cases where you need to keep up with a set of Boolean values; you just assign a bit to each value and set or clear it as appropriate. The BitSet class creates a special type of array that holds bit values. The BitSet array can increase in size as needed. This makes it similar to a vector of bits.

| Sr. No | Constructor & Description   |
|--------|---|
| 1      | BitSet( )- This constructor creates a default object.   |
| 2      | BitSet(int size)- This constructor allows you to specify its initial size, i.e., the number of bits that it can hold. All bits are initialized to zero. |

## 3) The Vector

The Vector class is similar to a traditional Java array, except that it can grow as necessary to accommodate new elements. Like an array, elements of a Vector object can be accessed via an index into the vector. The nice thing about using the Vector class is that you don't have to worry about setting it to a specific size upon creation; it shrinks and grows automatically when necessary.

Vector implements a dynamic array. It is similar to ArrayList, but with two differences – Vector is synchronized and Vector contains many legacy methods that are not part of the collections framework.

Vector proves to be very useful if you don't know the size of the array in advance or you just need one that can change sizes over the lifetime of a program.

| Sr. No. | Constructor and Description   |
|---------|---|
| 1       | Vector( )- This constructor creates a default vector, which has an initial size of 10.  |
| 2       | Vector(int size)- This constructor accepts an argument that equals to the required size, and creates a vector whose initial capacity is specified by size.  |
| 3       | Vector(int size, int incr)- This constructor creates a vector whose initial capacity is specified by size and whose increment is specified by incr. The increment specifies the number of elements to allocate each time that a vector is resized upward. |
| 4       | Vector(Collection c)- This constructor creates a vector that contains the elements of collection c.   |

Apart from the methods inherited from its parent classes, Vector defines the following methods.

| Sr. No | Method Description   |
|--------|--|
| 1      | void add(int index, Object element)- Inserts the specified element at the specified position in this Vector.             |
| 2      | void addElement(Object obj)- Adds the specified component to the end of this vector, increasing its size by one.         |
| 3      | int capacity()- Returns the current capacity of this vector.   |
| 4      | Enumeration elements()- Returns an enumeration of the components of this vector.   |
| 5      | Object firstElement()- Returns the first component (the item at index 0) of this vector.                                 |
| 6      | int hashCode()- Returns the hash code value for this vector.   |
| 7      | String toString()- Returns a string representation of this vector, containing the String representation of each element. |

#### 4) The Stack

The Stack class implements a last-in-first-out (LIFO) stack of elements. we can think of a stack literally as a vertical stack of objects; when we add a new element, it gets stacked on top of the others. When we pull an element off the stack, it comes off the top. In other words, the last element we added to the stack is the first one to come back off.

Stack is a subclass of Vector that implements a standard last-in, first-out stack. Stack only defines the default constructor, which creates an empty stack. Stack includes all the methods defined by Vector, and adds several of its own.

Apart from the methods inherited from its parent class Vector, Stack defines the following methods

| Sr. No | Method Description  |
|--------|---|
| 1      | boolean empty()- Tests if this stack is empty. Returns true if the stack is empty, and returns false if the stack contains elements.                  |
| 2      | Object peek( )- Returns the element on the top of the stack, but does not remove it.  |
| 3      | Object pop( )- Returns the element on the top of the stack, removing it in the process.   |
| 4      | Object push(Object element)- Pushes the element onto the stack. Element is also returned.   |
| 5      | int search(Object element)- Searches for element in the stack. If found, its offset from the top of the stack is returned. Otherwise, -1 is returned. |

## 5) The Dictionary

The Dictionary class is an abstract class that defines a data structure for mapping keys to values. This is useful in cases where you want to be able to access data via a particular key rather than an integer index.

Since the Dictionary class is abstract, it provides only the framework for a key-mapped data structure rather than a specific implementation.

The abstract methods defined by Dictionary are

| Sr. No | Method Description   |
|--------|--|
| 1      | Enumeration elements( )- Returns an enumeration of the values contained in the dictionary.   |
| 2      | Object get(Object key)- Returns the object that contains the value associated with the key. If the key is not in the dictionary, a null object is returned.  |
| 3      | Enumeration keys( )- Returns an enumeration of the keys contained in the dictionary.   |
| 4      | Object put(Object key, Object value)- Inserts a key and its value into the dictionary. Returns null if the key is not already in the dictionary; returns the previous value associated with the key if the key is already in the dictionary. |
| 5      | Object remove(Object key)- Removes the key and its value. Returns the value associated with the key. If the key is not in the dictionary, a null is returned.  |
| 6      | int size( )- Returns the number of entries in the dictionary.  |

## 6) The Hashtable

The Hashtable class provides a means of organizing data based on some user-defined key structure. For example, in an address list hash table you could store and sort data based on a key such as ZIP code rather than on a person's name.

The specific meaning of keys with regard to hash tables is totally dependent on the usage of the hash table and the data it contains.

Hashtable was part of the original java.util and is a concrete implementation of a Dictionary. Java 2 re-engineered Hashtable so that it also implements the Map interface. Thus, Hashtable is now integrated into the collections framework. It is similar to HashMap, but is synchronized.

| Sr. No | Constructor & Description   |
|--------|---|
| 1      | Hashtable( )- This is the default constructor of the hash table it instantiates the Hashtable class.  |
| 2      | Hashtable(int size)- This constructor accepts an integer parameter and creates a hash table that has an initial size specified by integer value size.   |
| 3      | Hashtable(int size, float fillRatio)- This creates a hash table that has an initial size specified by size and a fill ratio specified by fillRatio. This ratio must be between 0.0 and 1.0, and it determines how full the hash table can be before it is resized upward. |
| 4      | Hashtable(Map < ? extends K, ? extends V > t)- This constructs a Hashtable with the given mappings.   |

Apart from the methods defined by Map interface, Hashtable defines the following methods.

| Sr. No | Method Description  |
|--------|---|
| 1      | void clear( )- Resets and empties the hash table.   |
| 2      | Object clone( )- Returns a duplicate of the invoking object.  |
| 3      | Enumeration elements( )- Returns an enumeration of the values contained in the hash table.  |
| 4      | Object get(Object key)- Returns the object that contains the value associated with the key. If the key is not in the hash table, a null object is returned. |
| 5      | Enumeration keys( )- Returns an enumeration of the keys contained in the hash table.  |
| 6      | int size( )- Returns the number of entries in the hash table.   |
| 7      | String toString( )- Returns the string equivalent of a hash table.  |

## 7) The Properties

Properties is a subclass of Hashtable. It is used to maintain lists of values in which the key is a String and the value is also a String.

The Properties class is used by many other Java classes. For example, it is the type of object returned by System.getProperties( ) when obtaining environmental values.

Properties define the following instance variable. This variable holds a default property list associated with a Properties object.

| Sr. No | Constructor & Description  |
|--------|--|
| 1      | Properties( )- This constructor creates a Properties object that has no default values.  |
| 2      | Properties(Properties propDefault)- Creates an object that uses propDefault for its default values. In both cases, the property list is empty. |

Apart from the methods defined by Hashtable, Properties define the following methods-

| Sr. No | Method Description   |
|--------|--|
| 1      | String getProperty(String key)- Returns the value associated with the key. A null object is returned if the key is neither in the list nor in the default property list.                           |
| 2      | String getProperty(String key, String defaultProperty)- Returns the value associated with the key; defaultProperty is returned if the key is neither in the list nor in the default property list. |
| 3      | Enumeration propertyNames( )- Returns an enumeration of the keys. This includes those keys found in the default property list, too.  |
| 4      | Object setProperty(String key, String value)- Associates value with the key. Returns the previous value associated with the key, or returns null if no such association exists.                    |
| 5      | void store(OutputStream streamOut, String description)- After writing the string specified by description, the property list is written to the output stream linked to streamOut.                  |

- **Student should develop program for above problem definition**
- **Student supposed to attach output of implemented program**

**Conclusion:** Hence we conclude that after implementation of all data structures the data can be organized in computer in particular manner.

## Assignment-2

### Problem Definition:

Enhance the Message Chat system with the help of socket programming use client server architecture.

### 1.1 Prerequisite:

1. Java and Advance Java Programming.
2. Data Structures.
3. Socket Programming.

### 1.2 Learning Objectives:

1. Understand the concepts of advance Java.
2. Understand the process of writing client-server applications in java.
3. Illustrate the writing of connection-oriented server, and the corresponding client, in java.

### 1.3 New Concepts:

Client Server architecture using socket programming.

### 1.4 Theory

#### 1.4.1 Introduction

This section will answer the most frequently asked questions about programming sockets in Java. Sockets provide the communication mechanism between two computers 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.

#### 1.4.2 Client Server Architecture?

At a basic level, network-based systems consist of a server , client , and a media for communication. A computer running a program that makes a request for services is called client machine. A computer running a program that offers requested services from one or more clients is called server machine. The media for communication can be wired or wireless network.



### 1.4.3 Socket Programming and JAVA.NET Class

A socket is an endpoint of a two-way communication link between two programs running on the network.

Socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent. Java provides a set of classes, defined in a package called java.net, to enable the rapid development of network applications. Key classes, interfaces, and exceptions in java.net package simplifying the complexity involved in creating client and server programs are:

**Table: 2.1 The Classes of Socket**

| Sr. No | Class              | Sr. No | Class            |
|--------|--------------------|--------|------------------|
| 1      | ContentHandler     | 8      | ServerSocket     |
| 2      | DatagramPacket     | 9      | Socket           |
| 3      | DatagramSocket     | 10     | SocketImpl       |
| 4      | DatagramSocketImpl | 11     | URL              |
| 5      | HttpURLConnection  | 12     | URLConnection    |
| 6      | InetAddress        | 13     | URLEncoder       |
| 7      | MulticastSocket    | 14     | URLStreamHandler |

**Table: 2.2 The Interfaces**

| Sr. No | Interface             | Sr. No | Interface               |
|--------|-----------------------|--------|-------------------------|
| 1      | ContentHandlerFactory | 3      | FileNameMap             |
| 2      | SocketImplFactory     | 4      | URLStreamHandlerFactory |

**Table: 2.3 The Exceptions**

| Sr. No | Exceptions             | Sr. No | Exceptions              |
|--------|------------------------|--------|-------------------------|
| 1      | BindException          | 5      | ProtocolException       |
| 2      | ConnectException       | 6      | SocketException         |
| 3      | MalformedURLException  | 7      | UnknownHostException    |
| 4      | NoRouteToHostException | 8      | UnknownServiceException |

### 1.4.4 TCP/IP Socket Programming

The two key classes from the java.net package used in creation of server and client programs are:

1. ServerSocket
2. Socket

A server program creates a specific type of socket that is used to listen for client requests (server socket), In the case of a connection request, the program creates a new socket through which it will exchange data with the client using input and output streams. The socket abstraction is very similar to the file concept: developers have to open a socket, perform I/O, and close it. Figure 13.5 illustrates key steps involved in creating socket-based server and client programs.

#### A simple Server Program in Java

The steps for creating a simple server program are:

1. Open the Server Socket:

```
ServerSocket server = new ServerSocket( PORT );
```

2. Wait for the Client Request:

```
Socket client = server.accept();
```

3. Create I/O streams for communicating to the client

```
DataInputStream is = new DataInputStream(client.getInputStream());
```

```
DataOutputStream os = new DataOutputStream(client.getOutputStream());
```

4. Perform communication with client

```
Receive from client: String line = is.readLine();
```

```
Send to client: os.writeBytes("Hello\n");
```

5. Close socket:

```
client.close();
```

#### A simple Client Program in Java

The steps for creating a simple client program are:

1. Create a Socket Object:

```
Socket client = new Socket(server, port_id);
```

2. Create I/O streams for communicating with the server.

```
is = new DataInputStream(client.getInputStream());
```

```
os = new DataOutputStream(client.getOutputStream());
```

3. Perform I/O or communication with the server:

Receive data from the server: `String line = is.readLine();`

Send data to the server: `os.writeBytes("Hello\n");`

4. Close the socket when done:

```
client.close();
```

- **Student should develop program for above problem definition**
- **Student supposed to attach output of implemented program**

**Conclusion:** Hence we conclude that message chat system is implemented using client server architecture and socket programming.

---

## Assignment-3

### Problem Definition:

Enhance Message Chat system by using JDBC, Multithreading, concurrency, synchronous and asynchronous callbacks, ThreadPools using ExecutorService.

### 1.1 Prerequisites:

1. Java and Advance Java Programming.
2. MySQLData Base.
3. Socket Programming.

### 1.2 Learning Objectives:

1. Understand the concepts of advance Java.
2. Understand the process of writing client-server applications in java.
3. Illustrate the writing of connection-oriented server, and the corresponding client, in java.
4. Understanding the process of inserting data into database.

### 1.3 New Concepts:

1. Client Server architecture using socket programming.
2. Connectivity of Java with MySQL.

### 1.4 Theory

#### 1.4.1 Introduction

This section will answer the most frequently asked questions about programming sockets in Java. Sockets provide the communication mechanism between two computers 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 java and My SQL are connected and the data is inserted into the database.

#### 1.4.2 Client Server Architecture?

At a basic level, network-based systems consist of a server, client, and a media for communication. A computer running a program that makes a request for services is called client machine. A computer running a program that offers requested services from one or more clients is called server machine. The media for communication can be wired or wireless network.

The client/server architecture significantly decreased network traffic by providing a query response rather than total file transfer. It allows multi-user updating through a GUI front end to a shared database. Remote Procedure Calls (RPCs) or standard query language (SQL) statements are typically used to communicate between the client and server.

**The basic characteristics of client/server architectures are:**

- 1) Combination of a client or **front-end portion** that interacts with the user, and a server or **back-end portion** that interacts with the shared resource. The client process contains **solution-specific logic** and provides the interface between the user and the rest of the application system. The server process acts as a **software engine** that manages shared resources such as databases, printers, modems, or high powered processors.
- 2) The front-end task and back-end task have fundamentally different requirements for computing resources such as processor speeds, memory, disk speeds and capacities, and input/output devices.
- 3) The environment is typically **heterogeneous** and multivendor. The hardware platform and operating system of client and server are not usually the same. Client and server processes communicate through a well-defined set of standard application program interfaces (API's) and RPC's.
- 4) An important characteristic of client-server systems is scalability. They can be scaled horizontally or vertically. Horizontal scaling means adding or removing client workstations with only a slight performance impact. Vertical scaling means migrating to a larger and faster server machine or multiservers.

### **1.4.3 Socket Programming and JAVA.NET Class**

A socket is an endpoint of a two-way communication link between two programs running on the network. Socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent. Java provides a set of classes, defined in a package called java.net, to enable the rapid development of network applications. Key classes, interfaces, and exceptions in java.net package simplifying the complexity involved in creating client and server programs are:

**Table: 2.1 The Classes of Socket**

| Sr. No | Class              | Sr. No | Class            |
|--------|--------------------|--------|------------------|
| 1      | ContentHandler     | 8      | ServerSocket     |
| 2      | DatagramPacket     | 9      | Socket           |
| 3      | DatagramSocket     | 10     | SocketImpl       |
| 4      | DatagramSocketImpl | 11     | URL              |
|        |                    |        |                  |
| 5      | HttpURLConnection  | 12     | URLConnection    |
| 6      | InetAddress        | 13     | URLEncoder       |
| 7      | MulticastSocket    | 14     | URLStreamHandler |

**Table: 2.2 The Interfaces**

| Sr. No | Interface             | Sr. No | Interface               |
|--------|-----------------------|--------|-------------------------|
| 1      | ContentHandlerFactory | 3      | FileNameMap             |
| 2      | SocketImplFactory     | 4      | URLStreamHandlerFactory |

**Table: 2.3 The Exceptions**

| Sr. No | Exceptions             | Sr. No | Exceptions              |
|--------|------------------------|--------|-------------------------|
| 1      | BindException          | 5      | ProtocolException       |
| 2      | ConnectException       | 6      | SocketException         |
| 3      | MalformedURLException  | 7      | UnknownHostException    |
| 4      | NoRouteToHostException | 8      | UnknownServiceException |

#### 1.4.4 TCP/IP Socket Programming

The two key classes from the java.net package used in creation of server and client programs are:

1. ServerSocket
2. Socket

A server program creates a specific type of socket that is used to listen for client requests (server socket), In the case of a connection request, the program creates a new socket through which it will exchange data with the client using input and output streams. The socket abstraction is very similar to the file concept: developers have to open a socket, perform I/O, and close it.

### 1.4.5 ServerSocket Class Methods

The `java.net.ServerSocket` class is used by server applications to obtain a port and listen for client requests. The `ServerSocket` class has four constructors are:

| Sr. No. | Method & Description   |
|---------|--|
| 1       | <b>public ServerSocket(int port) throws IOException-</b> Attempts to create a server socket bound to the specified port. An exception occurs if the port is already bound by another application.  |
| 2       | <b>public ServerSocket(int port, int backlog) throws IOException-</b> Similar to the previous constructor, the backlog parameter specifies how many incoming clients to store in a wait queue.   |
| 3       | <b>public ServerSocket(int port, int backlog, InetAddress address) throws IOException-</b> Similar to the previous constructor, the <code>InetAddress</code> parameter specifies the local IP address to bind to. The <code>InetAddress</code> is used for servers that may have multiple IP addresses, allowing the server to specify which of its IP addresses to accept client requests on. |
| 4       | <b>public ServerSocket() throws IOException-</b> Creates an unbound server socket. When using this constructor, use the <code>bind()</code> method when you are ready to bind the server socket.   |

If the `ServerSocket` constructor does not throw an exception, it means that your application has successfully bound to the specified port and is ready for client requests. Following are some of the common methods of the `ServerSocket` class are:

| Sr. No. | Method & Description  |
|---------|---|
| 1       | <b>public int getLocalPort()-</b> Returns the port that the server socket is listening on. This method is useful if you passed in 0 as the port number in a constructor and let the server find a port for you. |
| 2       | <b>public Socket accept() throws IOException-</b> Waits for an incoming client. This method   |

|   |   |
|---|---|
|   | blocks until either a client connects to the server on the specified port or the socket times out, assuming that the time-out value has been set using the <code>setSoTimeout()</code> method. Otherwise, this method blocks indefinitely.                        |
| 3 | <b>public void setSoTimeout(int timeout)-</b> Sets the time-out value for how long the server socket waits for a client during the <code>accept()</code> .  |
| 4 | <b>public void bind(SocketAddress host, int backlog)</b><br>Binds the socket to the specified server and port in the <code>SocketAddress</code> object. Use this method if you have instantiated the <code>ServerSocket</code> using the no-argument constructor. |

When the `ServerSocket` invokes `accept()`, the method does not return until a client connects. After a client does connect, the `ServerSocket` creates a new `Socket` on an unspecified port and returns a reference to this new `Socket`. A TCP connection now exists between the client and the server, and communication can begin.

#### 1.4.6 Socket Class Methods

The `java.net.Socket` class represents the socket that both the client and the server use to communicate with each other. The client obtains a `Socket` object by instantiating one, whereas the server obtains a `Socket` object from the return value of the `accept()` method.

The `Socket` class has five constructors that a client uses to connect to a server –

| Sr.No. | Method & Description   |
|--------|--|
| 1      | <b>public Socket(String host, int port) throws UnknownHostException, IOException.</b><br>This method attempts to connect to the specified server at the specified port. If this constructor does not throw an exception, the connection is successful and the client is connected to the server. |
| 2      | <b>public Socket(InetAddress host, int port) throws IOException</b><br>This method is identical to the previous constructor, except that the host is denoted by an <code>InetAddress</code> object.  |
| 3      | <b>public Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException.</b><br>Connects to the specified host and port, creating a socket on the local host at the specified address and port.  |
| 4      | <b>public Socket(InetAddress host, int port, InetAddress localAddress, int localPort)</b>  |



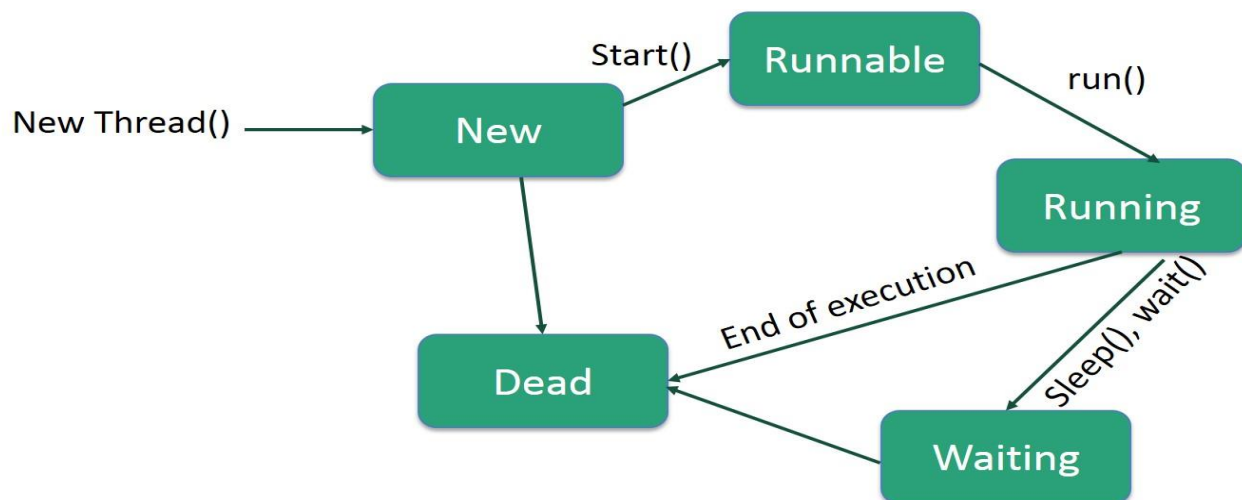
|   |  |
|---|--|
|   | <p><b>throws IOException.</b></p> <p>This method is identical to the previous constructor, except that the host is denoted by an InetAddress object instead of a String.</p> |
| 5 | <p><b>public Socket()-</b> Creates an unconnected socket. Use the connect() method to connect this socket to a server.</p>   |

### 1.4.7 Multithreading:

Multithreading in java is a process of executing multiple threads simultaneously. Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking. But we use multithreading than multiprocessing because threads share a common memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process. Java Multithreading is mostly used in games, animation etc.

#### *Life Cycle of a Thread*

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. The following diagram shows the complete life cycle of a thread.



Following are the stages of the life cycle –

- **New** – A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a **born thread**.
- **Runnable** – After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.

- **Waiting** – Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- **Timed Waiting** – A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- **Terminated (Dead)** – A runnable thread enters the terminated state when it completes its task or otherwise terminates.

### Advantages of Java Multithreading

- 1) It doesn't block the user because threads are independent and you can perform multiple operations at same time.
- 2) You can perform many operations together so it saves time.
- 3) Threads are independent so it doesn't affect other threads if exception occur in a single thread.

### 1.4.8 Java with mysql Database Connection

For connecting java application with the mysql database, you need to follow 5 steps to perform database connectivity.

In this example we are using MySQL as the database. So we need to know following informations for the mysql database:

1. **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.
2. **Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/chatroom** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, you need to replace the sonoo with your database name.
3. **Username:** The default username for the mysql database is **root**.
4. **Password:** Password is given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

Let's first create a table in the mysql database, but before creating table, need to create database first.

#### //Create Database

1. student@hwl-pc22:~\$ mysql -u root -p
2. mysql> CREATE DATABASE chatroom;
3. mysql> show databases;
4. mysql> use chatroom;

5. Database changed

6. `mysql> create table chatroom(u char(40), m char(40), time char(40));`

**//Build Database**

1. Go to Eclipse
2. Right Click on Project
3. Build Path
4. Configure build path
5. Add External Jars for Mysql Connector
6. Add mysql-connector-java-3.1.14-bin.jar
7. Click OK.

**//for Checking in table values**

Select \* from chatroom;

- **Student should develop program for above problem definition**
- **Student supposed to attach output of implemented program**

**Conclusion:** Hence we conclude that message chat system using JDBC, Multithreading is implemented using client server architecture and socket programming.

## Assignment-4

### Problem Definition:

Transform the Message Chat system from command line system to GUI based application

### 1.1 Prerequisites:

1. Java and Advance Java Programming.
2. MySQLData Base.
3. Socket Programming.

### 1.2 Learning Objectives:

1. Illustrate how a given java application can be converted into a java applet.
2. Explain the applet life cycle, with examples.
3. Illustrate the embedding of multiple applets in the same HTML page.
4. Demonstrate the execution of more complex applets.

### 1.3 New Concepts:

1. Client Server architecture using socket programming.
2. Connectivity of Java with MySQL.
3. Applet programming.

### 1.4 Theory

#### 1.4.1 Introduction

This section applet programming is introduced to form the application program into GUI in Java. 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. It is fairly straightforward to convert a

graphical Java application into an applet. Why so? Both the Applet and the Frame classes descend from Container. Thus the same methods may be used to add the user interface components.

#### 1.4.2 Client Server Architecture?

At a basic level, network-based systems consist of a server, client, and a media for communication. A computer running a program that makes a request for services is called client machine. A computer running a program that offers requested services from one or more clients is called server machine. The media for communication can be wired or wireless network.

The client/server architecture significantly decreased network traffic by providing a query response rather than total file transfer. It allows multi-user updating through a GUI front end to a shared database. Remote Procedure Calls (RPCs) or standard query language (SQL) statements are typically used to communicate between the client and server.

### **1.4.3 Socket Programming and JAVA.NET Class**

A socket is an endpoint of a two-way communication link between two programs running on the network. Socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent. Java provides a set of classes, defined in a package called `java.net`, to enable the rapid development of network applications. Key classes, interfaces, and exceptions in `java.net` package simplifying the complexity involved in creating client and server programs are:

### **1.4.4 TCP/IP Socket Programming**

The two key classes from the `java.net` package used in creation of server and client programs are:

1. `ServerSocket`
2. `Socket`

A server program creates a specific type of socket that is used to listen for client requests (server socket). In the case of a connection request, the program creates a new socket through which it will exchange data with the client using input and output streams. The socket abstraction is very similar to the file concept: developers have to open a socket, perform I/O, and close it.

### **1.4.5 Java Applets**

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.

There are some important differences between an applet and a standalone Java application, including the following –

1. An applet is a Java class that extends the `java.applet.Applet` class.
2. A `main()` method is not invoked on an applet, and an applet class will not define `main()`.
3. Applets are designed to be embedded within an HTML page.
4. When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.

5. A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
6. The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
7. 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.

### **Life Cycle of an Applet**

Four methods in the Applet class gives you the framework on which you build any serious applet

1. `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.
2. `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.
3. `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.
4. `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.
5. `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`.

### **The Applet Class**

Every applet is an extension of the `java.applet.Applet` class. The base Applet class provides methods that a derived Applet class may call to obtain information and services from the browser context. These include methods that do the following –

1. Get applet parameters
2. Get the network location of the HTML file that contains the applet
3. Get the network location of the applet class directory
4. Print a status message in the browser
5. Fetch an image

6. Fetch an audio clip
7. Play an audio clip
8. Resize the applet

Additionally, the Applet class provides an interface by which the viewer or browser obtains information about the applet and controls the applet's execution. The viewer may

1. Request information about the author, version, and copyright of the applet
2. Request a description of the parameters the applet recognizes
3. Initialize the applet
4. Destroy the applet
5. Start the applet's execution
6. Stop the applet's execution

The Applet class provides default implementations of each of these methods. Those implementations may be overridden as necessary.

### **java.awt.Component class**

The Component class provides 1 life cycle method of applet.

1. `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.

### **Graphics in Applet**

`java.awt.Graphics` class provides many methods for graphics programming.

### **JApplet class in Applet**

As we prefer Swing to AWT. Now we can use JApplet that can have all the controls of swing. The JApplet class extends the Applet class

### **Applet Communication**

`java.applet.AppletContext` class provides the facility of communication between applets. We provide the name of applet through the HTML file. It provides `getApplet()` method that returns the object of Applet.

Syntax: `public Applet getApplet(String name){ }`

### 1.4.6 Java Swing Tutorial

Java Swing tutorial 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.

#### What is JFC

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

#### Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

| Sr. 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.  |

#### Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.



| Sr. NO | Method                                    | Description   |
|--------|---|---|
| 1      | public void add(Component c)              | add a component on another component.                         |
| 2      | public void setSize(int width,int height) | sets size of the component.                                   |
| 3      | public void setLayout(LayoutManager m)    | sets the layout manager for the component.                    |
| 4      | public void setVisible(boolean b)         | sets the visibility of the component. It is by default false. |

### Advantages of Applet

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

### Drawback of Applet

1. Plugin is required at client browser to execute applet.

### 1.4.8 Steps to convert graphical Java application into an applet

1. Create a HTML file with an APPLETTAG tag

which specifies the name of the applet class file, applet window size, and other relevant information.

2. Drop the main method.

- In a Java application, the main method usually contains code to create a new frame object.
- In an applet, however, creation of an applet object is done by the browser automatically.
- The main method defines the frame size. For the applet, the size information is provided by the WIDTH and HEIGHT attributes of APPLETTAG tag.

3. Instead of deriving the class from Frame, derive it from Applet.

4. Replace the constructor of the Java application with a method called init().

- After the browser creates an object of the applet class, the init() method is automatically called.

5. Take special care regarding the default layout manager.

- Java applications use BorderLayout manager as default, while applets use FlowLayout.
- The following must be included in init() method: setLayout (new BorderLayout());

6. Applets do not have title bars, and so any call to setTitle method must be omitted.

7. Add the following line at the beginning of the program:

**import java.applet.\***

- **Student should develop program for above problem definition**
- **Student supposed to attach output of implemented program**

**Conclusion:** Hence we conclude that message chat system can be converted using applet and swing into GUI based Program.

## Assignment-5

### Problem Definition:

Download Install and Configure Android Studio on Linux/windows platform.

### 1.1 Prerequisite:

1. Java and Advance Java Programming.
2. Android Studio.

### 1.2 Learning Objectives:

1. Understand the concepts Android Studio.

### 1.3 Theory

#### 1.3.1 Introduction

Android is an open source and Linux-based Operating System for mobile devices such as smartphones and tablet computers. Android was developed by the Open Handset Alliance, led by Google, and other companies. Android offers a unified approach to application development for mobile devices which means developers need to develop only for Android, and their applications should be able to run on different devices powered by Android. The first beta version of the Android Software Development Kit (SDK) was released by Google in 2007, whereas the first commercial version, Android 1.0, was released in September 2008. On June 27, 2012, at the Google I/O conference, Google announced the next Android version, 4.1 Jelly Bean. Jelly Bean is an incremental update, with the primary aim of improving the user interface, both in terms of functionality and performance. Android is a mobile operating system that is based on a modified version of Linux. It was originally developed by a startup of the same name, Android, Inc. In 2005, as part of its strategy to enter the mobile space, Google purchased Android and took over its development work (as well as its development team). Google wanted Android to be open and free; hence, most of the Android code was released under the open-source Apache License, which means that anyone who wants to use Android can do so by downloading the full Android source code. Moreover, vendors (typically hardware manufacturers) can add their own proprietary extensions to Android and customize Android to differentiate their products from others. This simple development model makes Android very attractive and has thus piqued the interest of many vendors. The main advantage of adopting Android is that it offers a unified approach to application

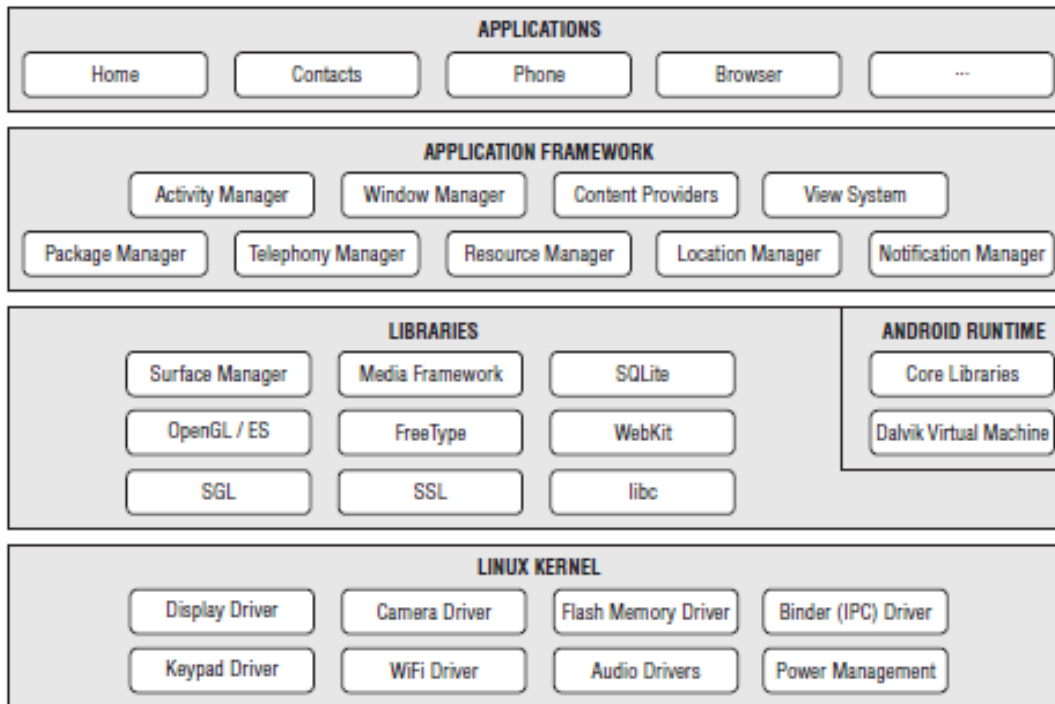
development. Developers need only develop for Android, and their applications should be able to run on numerous different devices, as long as the devices are powered using Android. In the world of smartphones, applications are the most important part of the success chain. Device manufacturers therefore see Android as their best hope to challenge the onslaught of the iPhone, which already commands a large base of applications.

### 1.3.2 Features of Android

| Feature          | Description   |
|------------------|---|
| Beautiful UI     | Android OS basic screen provides a beautiful and intuitive user interface.  |
| Storage          | SQLite, a lightweight relational database, is used for data storage purposes.   |
| Connectivity     | Supports GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth includes A2DP and AVRCP), WiFi, LTE, and WiMAX. .                     |
| Messaging        | Supports both SMS and MMS   |
| Web browser      | Based on the open-source WebKit layout engine, coupled with Chrome's V8 JavaScript engine supporting HTML5 and CSS3.        |
| Media support    | Includes support for the following media: H.263, H.264 (in 3GP or MP4 container), MPEG-4 SP, AMR, AMR-WB (in 3GP container) |
| Hardware support | Accelerometer Sensor, Camera, Digital Compass, Proximity Sensor, and GPS  |
| Multi-touch      | Android has native support for multi-touch which was initially made available in handsets such as the HTC Hero.             |
| Multi-tasking    | User can jump from one task to another and same time various application can run simultaneously.                            |
| Tethering        | Supports sharing of Internet connections as a wired/wireless hotspot  |

### 1.3.3 Architecture of Android

In order to understand how Android works, take a look at Figure 1-1, which shows the various layers that make up the Android operating system (OS).



The Android OS is roughly divided into five sections in four main layers:

**Linux kernel:** This is the kernel on which Android is based. This layer contains all the low level device drivers for the various hardware components of an Android device.

**Libraries:** These contain all the code that provides the main features of an Android OS. For example, the SQLite library provides database support so that an application can use it for data storage. The WebKit library provides functionalities for web browsing.

**Android runtime:** At the same layer as the libraries, the Android runtime provides a set of core libraries that enable developers to write Android apps using the Java programming language. The Android runtime also includes the Dalvik virtual machine, which enables every Android application to run in its own process; with its own instance of the Dalvik virtual machine (Android applications are compiled into the Dalvik executable). Dalvik is a specialized virtual machine designed specifically for Android and optimized for battery-powered mobile devices with limited memory and CPU.

**Application framework:** The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

**Applications:** At this top layer, we will find applications that ship with the Android device

(such as Phone, Contacts, Browser, etc.), as well as applications that you download and install from the Android Market. Any applications that we write are located at this layer.

### 1.3.4 APPLICATIONS COMPONENT

There are following four main components that can be used within an Android application:

| Components          | Description  |
|---------------------|--|
| Activities          | They dictate the UI and handle the user interaction to the smartphone screen |
| Services            | They handle background processing associated with an application.            |
| Broadcast Receivers | They handle communication between Android OS and applications.               |
| Content Providers   | They handle data and database management issues.                             |

#### Activities:

An activity represents a single screen with a user interface. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and one for reading emails. If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched. An activity is implemented as a subclass of **Activity** class as follows:

```
public class MainActivity extends Activity
{
}
```

#### Services:

A service is a component that runs in the background to perform long-running operations. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity. A service is implemented as a subclass of **Service** class as follows:

```
public class MyService extends Service
{
}
```

#### Broadcast Receivers

Broadcast Receivers simply respond to broadcast messages from other applications or from the system. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action. A broadcast receiver is implemented as a subclass of **BroadcastReceiver** class and each message is broadcasted as an **Intent** object.

```
public class MyReceiver extends BroadcastReceiver
{
}
```

### Content Providers

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the *ContentResolver* class. The data may be stored in the file system, the database or somewhere else entirely. A content provider is implemented as a subclass of **ContentProvider** class and must implement a standard set of APIs that enable other applications to perform transactions.

```
public class MyContentProvider extends ContentProvider
{
}
```

### 1.3.5 Additional Components

There are additional components which will be used in the construction of above mentioned entities, their logic, and wiring between them. These components are:

| Components | Description  |
|------------|--|
| Fragments  | Represent behaviour or a portion of user interface in an Activity.       |
| Views      | UI elements that are drawn onscreen including buttons, lists forms etc.  |
| Layouts    | View hierarchies that control screen format and appearance of the views. |
| Intents    | Messages wiring components together.                                     |
| Resources  | External elements, such as strings, constants and draw able pictures.    |
| Manifest   | Configuration files for the application.                                 |

### How to install Android Studio on Ubuntu

## Installing Java

```
sudo add-apt-repository ppa:webupd8team/java
```

```
sudo apt-get update
```

```
sudo apt-get install oracle-java8-installer
```

After that

```
sudo apt-get install oracle-java8-set-default
```

## Installing Android Studio

1. Download Android Studio, use *All Android Studio Packages*
2. Extract the archive file into an appropriate location for your applications, eg: /opt. Use the filename of your downloaded archive, in my example `android-studio-ide-141.2178183-linux.zip`
3. `sudo unzip android-studio-ide-141.2178183-linux.zip -d /opt`
4. To launch Android Studio, navigate to the `/opt/android-studio/bin` directory in a terminal and execute `./studio.sh`. Or use a desktop file, see below. You may want to add `/opt/android-studio/bin` to your PATH environmental variable so that you can start Android Studio from any directory.

**Conclusion:** Thus we studied installation & basics of Android.



## Assignment-6

### Problem Definition:

Design a mobile app for media player

### 1.1 Prerequisite:

1. Java and Advance Java Programming.
2. Android

### 1.2 Theory

Android provides many ways to control playback of audio/video files and streams. One of this ways is through a class called MediaPlayer. Android is providing MediaPlayer class to access built-in mediaplayer services like playing audio, video e.t.c. In order to use MediaPlayer, we have to call a static Method create ( ) of this class. This method returns an instance of MediaPlayer class. Its syntax is as follows

```
MediaPlayer mediaPlayer = MediaPlayer.create(this, R.raw.song);
```

The second parameter is the name of the song that you want to play. You have to make a new folder under your project with name **raw** and place the music file into it. Once you have created the MediaPlayer object you can call some methods to start or stop the music. These methods are listed below.

```
mediaPlayer.start( );
```

```
mediaPlayer.pause( );
```

On call to start( ) method, the music will start playing from the beginning. If this method is called again after the pause( ) method, the music would start playing from where it is left and not from the beginning. In order to start music from the beginning, you have to call reset( ) method. Its syntax is given below.

```
mediaPlayer.reset( );
```

Apart from the start and pause method, there are other methods provided by this class for better dealing with audio/video files. These methods are listed below:

| Sr.no | Method       | Description   |
|-------|--------------|---|
| 1     | isPlaying( ) | This method just returns true/false indicating the song is playing or not |

|    |  |   |
|----|--|---|
| 2  | seekTo(position)                               | This method takes an integer, and move song to that particular second                     |
| 3  | getCurrentDuration( )                          | This method returns the current position of song in milliseconds                          |
| 4  | getDuration( )                                 | This method returns the total time duration of song in milliseconds                       |
| 5  | reset( )                                       | This method resets the media player   |
| 6  | release( )                                     | This method releases any resource attached with MediaPlayer object                        |
| 7  | setVolume(float leftVolume, float rightVolume) | This method sets the up down volume for this player                                       |
| 8  | setDataSource(FileDescriptor fd)               | This method sets the data source of audio/video file                                      |
| 9  | selectTrack(int index)                         | This method takes an integer, and select the track from the list on that particular index |
| 10 | getTrackInfo( )                                | This method returns an array of track information   |

### Example

Here is an example demonstrating the use of MediaPlayer class. It creates a basic media player that allows you to forward, backward, play and pause a song. To experiment with this example, you need to run this on an actual device to hear the audio sound.

| Steps | Description   |
|-------|---|
| 1     | You will use Android studio IDE to create an Android application under a package com.example.myapplication. |

---

|   |  |
|---|--|
| 2 | Modify src/MainActivity.java file to add MediaPlayer code.   |
| 3 | Modify the res/layout/activity_main to add respective XML components   |
| 4 | Create a new folder under MediaPlayer with name as raw and place an mp3 music file in it with name as song.mp3   |
| 5 | Run the application and choose a running android device and install the application on it and verify the results |

- **Student should develop program for above problem definition**
- **Student supposed to attach output of implemented program**

**Conclusion:** Thus we studied to design a media player using Android Studio.

## Assignment-7

### Problem Definition:

Design a mobile app to store data using internal or external storage.

### Theory

Android provides many kinds of storage for applications to store their data. These storage places are shared preferences, internal and external storage, SQLite storage, and storage via network connection. Internal storage is the storage of the private data on the device memory. By default these files are private and are accessed by only your application and get deleted, when user delete your application.

### Writing file

In order to use internal storage to write some data in the file, call the `openFileOutput()` method with the name of the file and the mode. The mode could be private , public e.t.c. Its syntax is given below:

```
FileOutputStream fOut = openFileOutput("file name here",MODE_WORLD_READABLE);
```

The method `openFileOutput ( )` returns an instance of `FileOutputStream`. So you receive it in the object of `FileInputStream`. After that you can call `write` method to write data on the file. Its syntax is given below:

```
String str = "data";  
fOut.write(str.getBytes());  
fOut.close();
```

### Reading file

In order to read from the file you just created, call the `openFileInput()` method with the name of the file. It returns an instance of `FileInputStream`. Its syntax is given below:

```
FileInputStream fin = openFileInput(file);
```

After that, you can call `read` method to read one character at a time from the file and then you can print it. Its syntax is given below –

```
int c;
```

```
String temp="";
while( (c = fin.read()) != -1){
    temp = temp + Character.toString((char)c);
}
//string temp contains all the data of the file.
fin.close()
```

The methods of write and close, there are other methods provided by the **FileOutputStream** class for better writing files. These methods are listed below:

| Sr.no | Method  | Description  |
|-------|---|--|
| 1     | FileOutputStream(File file, boolean append)         | This method constructs a new FileOutputStream that writes to file.                                   |
| 2     | getChannel()  | This method returns a write-only FileChannel that shares its position with this stream               |
| 3     | getFD()   | This method returns the underlying file descriptor   |
| 4     | write(byte[] buffer, int byteOffset, int byteCount) | This method Writes count bytes from the byte array buffer starting at position offset to this stream |

The methods of read and close, there are other methods provided by the **FileInputStream** class for better reading files. These methods are listed below:

| Sr.no | Method       | Description  |
|-------|--------------|--|
| 1     | available()  | This method returns an estimated number of bytes that can be read or skipped without blocking for more input |
| 2     | getChannel() | This method returns a read-only FileChannel that shares  |

|   |   |   |
|---|---|---|
|   |   | its position with this stream   |
| 3 | getFD()   | This method returns the underlying file descriptor  |
| 4 | read(byte[] buffer, int<br>byteOffset, int byteCount) | This method reads at most length bytes from this stream<br>and stores them in the byte array b starting at offset |

### Example

Here is an example demonstrating the use of internal storage to store and read files. It creates a basic storage application that allows you to read and write from internal storage. You can run this on an actual device or in an emulator.

Steps:

1. You will use Android Studio IDE to create an Android application under a package com.example.myapplication.
2. Modify src/MainActivity.java file to add necessary code.
3. Modify the res/layout/activity\_main to add respective XML components
4. Run the application and choose a running android device and install the application on it and verify the results

- **Student should develop program for above problem definition**
- **Student supposed to attach output of implemented program**

**Conclusion:** Hence, we have learned to store data using internal & external storage.

## Assignment-8

### Problem Definition:

Design a mobile app using Google Map and GPS to trace the location.

### Theory

Google Maps is one of the many applications bundled with the Android platform. Android allows us to integrate Google maps in our application. We can show any location on the map, or can show different routes on the map etc. We can also customize the map according to our choices.

### Google Map: Layout file

Now we have to add the map fragment into xml layout file. Its syntax is given below:

```
<fragment
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.MapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

### Google Map: AndroidManifest file

The next thing you need to do is to add some permission along with the Google Map API key in the AndroidManifest.XML file. Its syntax is given below:

```
<!--Permissions-->
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="com.google.android.providers.gsf.permission.
    READ_GSERVICES" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

<!--Google MAP API key-->
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="AIzaSyDKymeBXNeiFWY5jRUejv6zItpmr2MVyQ0" />
```

### ***Customizing Google Map***

You can easily customize google map from its default view, and change it according to your demand.

### **Adding Marker**

You can place a maker with some text over it displaying your location on the map. It can be done by via addMarker() method. Its syntax is given below:

```
final LatLng TutorialPoint = new LatLng(21 , 57);  
Marker TP = googleMap.addMarker(new MarkerOptions()  
    .position(TutorialPoint).title("TutorialPoint"));
```

### **Channing Map Type**

You can also change the type of the MAP. There are four different types of map and each give different view of the map. These types are Normal, Hybrid, Satellite and terrain. You can use them as below:

```
googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);  
googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);  
googleMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);  
googleMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
```

### **Enable/Disable zoom**

You can also enable or disable the zoom gestures in the map by calling the setZoomControlsEnabled(boolean) method. Its syntax is given below:

```
googleMap.getUiSettings().setZoomGesturesEnabled(true);
```

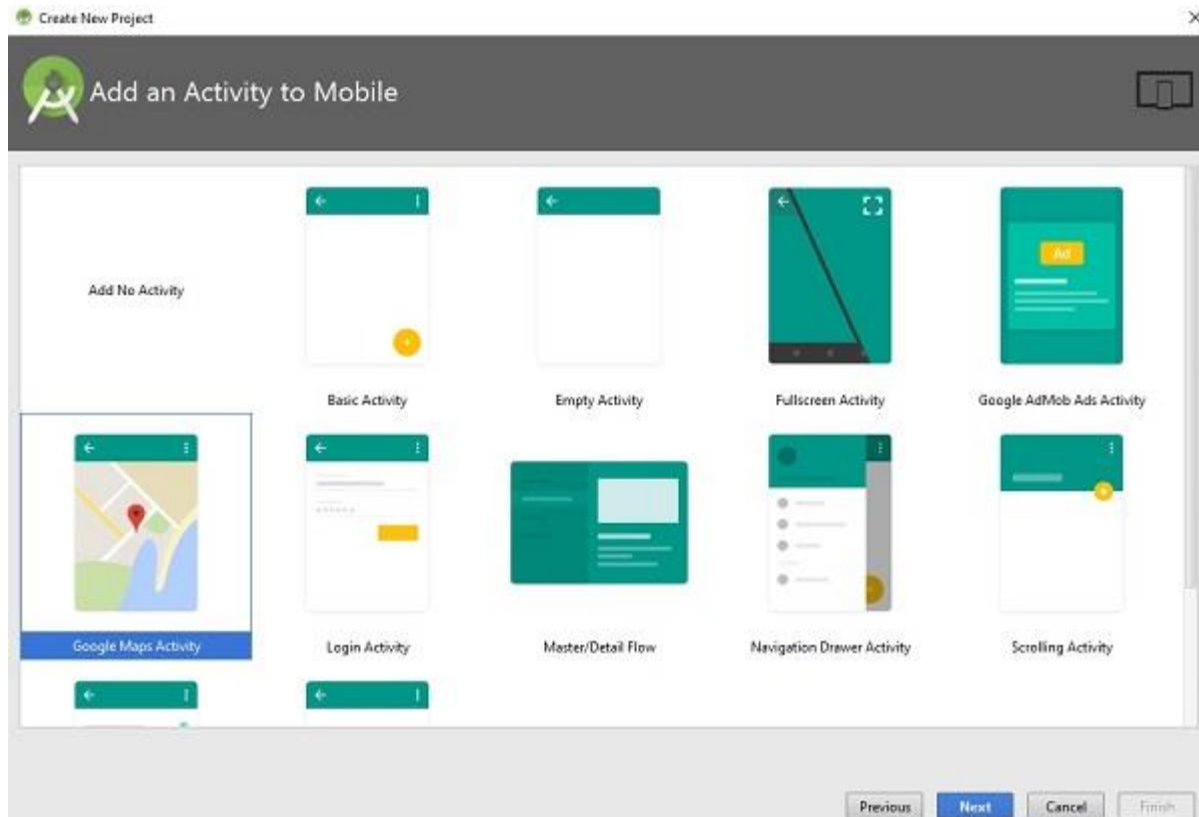
Apart from this customization, there are other methods available in the GoogleMap class, that helps you more customize the map. They are listed below:



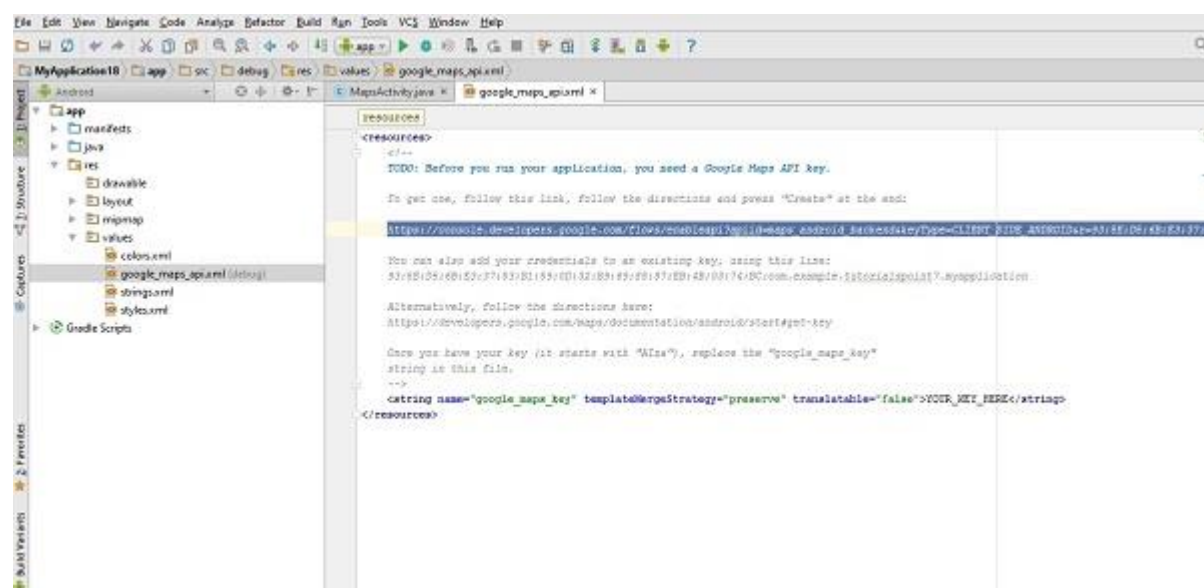
| Sr.no | Method   | Description  |
|-------|--|--|
| 1     | addCircle(CircleOptions options)                   | This method add a circle to the map  |
| 2     | addPolygon(PolygonOptions options)                 | This method add a polygon to the map   |
| 3     | addTileOverlay(TileOverlayOptions options)         | This method add tile overlay to the map  |
| 4     | animateCamera(CameraUpdate update)                 | This method Moves the map according to the update with an animation                    |
| 5     | clear()  | This method removes everything from the map.   |
| 6     | getMyLocation()                                    | This method returns the currently displayed user location.                             |
| 7     | moveCamera(CameraUpdate update)                    | This method repositions the camera according to the instructions defined in the update |
| 8     | setTrafficEnabled(boolean enabled)                 | This method Toggles the traffic layer on or off.                                       |
| 9     | snapshot(GoogleMap.SnapshotReadyCallback callback) | snapshot(GoogleMap.SnapshotReadyCallback callback)                                     |
| 10    | stopAnimation()                                    | This method stops the camera animation if there is one in progress                     |

## Example

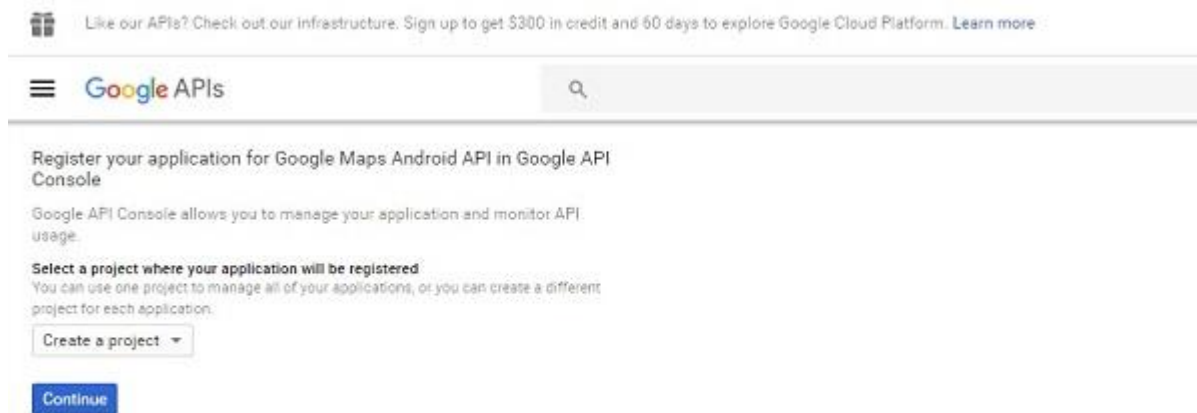
Here is an example demonstrating the use of GoogleMap class. It creates a basic M application that allows you to navigate through the map. To experiment with this example, you can run this on an actual device or in an emulator. Create a project with google maps activity as shown below:



It will open the following screen and copy the console url for API Key as shown below:



Copy this and paste it to your browser. It will give the following screen



Like our APIs? Check out our infrastructure. Sign up to get \$300 in credit and 60 days to explore Google Cloud Platform. [Learn more](#)

Google APIs

Register your application for Google Maps Android API in Google API Console

Google API Console allows you to manage your application and monitor API usage.

Select a project where your application will be registered  
You can use one project to manage all of your applications, or you can create a different project for each application.

Create a project

Continue

Click on continue and click on Create API Key then it will show the following screen



API key created

Use this key in your application by passing it with the `key=API_KEY` parameter.

Your API key

AIzaSyAXhBdyKxUo\_cb-EkSgWJQTdqR0QjLcques

⏏

⚠ Restrict your key to prevent unauthorized use in production.

Close Restrict key

- Student should develop program for above problem definition
- Student supposed to attach output of implemented program

**Conclusion:** We have studied to locate our position using Google Map & GPS.