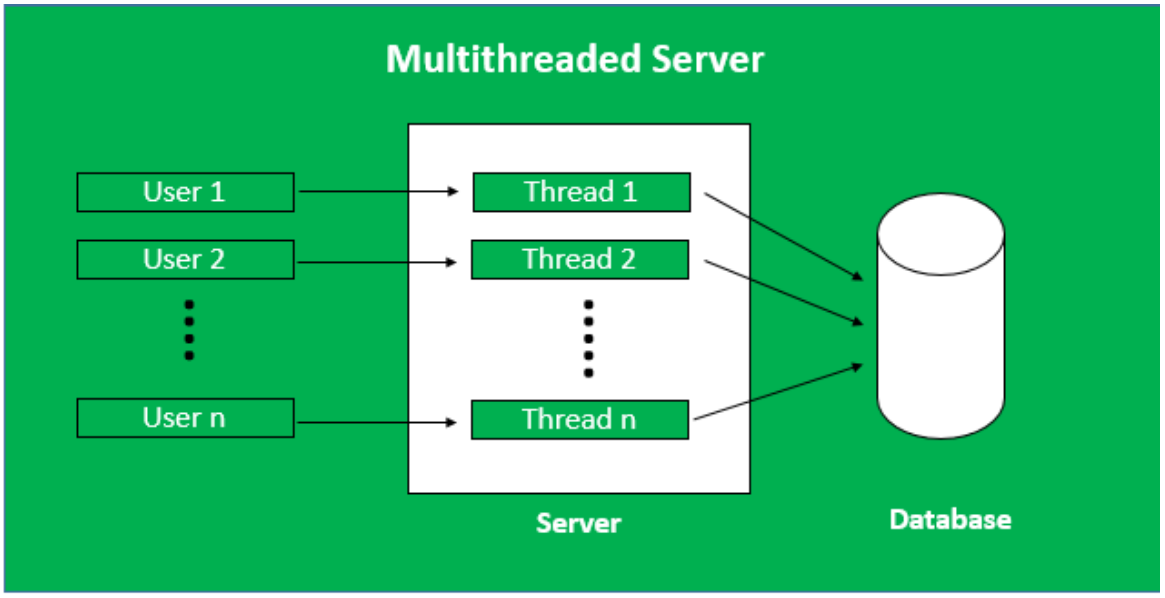
Multithreaded Servers in Java

Last Updated : 23 Jul, 2025

Prerequisites: Socket Programming in Java

Multithreaded Server: A server having more than one thread is known as Multithreaded Server. When a client sends the request, a thread is generated through which a user can communicate with the server. We need to generate multiple threads to accept multiple requests from multiple clients at the same time.



Advantages of Multithreaded Server:

- **Quick and Efficient:** Multithreaded server could respond efficiently and quickly to the increasing client queries quickly.
- **Waiting time for users decreases:** In a single-threaded server, other users had to wait until the running process gets completed but in multithreaded servers, all users can get a response at a single time so no user has to wait for other processes to finish.
- **Threads are independent of each other:** There is no relation between any two threads. When a client is connected a new thread is generated every time.
- **The issue in one thread does not affect other threads:** If any error occurs in any of the threads then no other thread is disturbed, all other processes keep running normally. In a single-

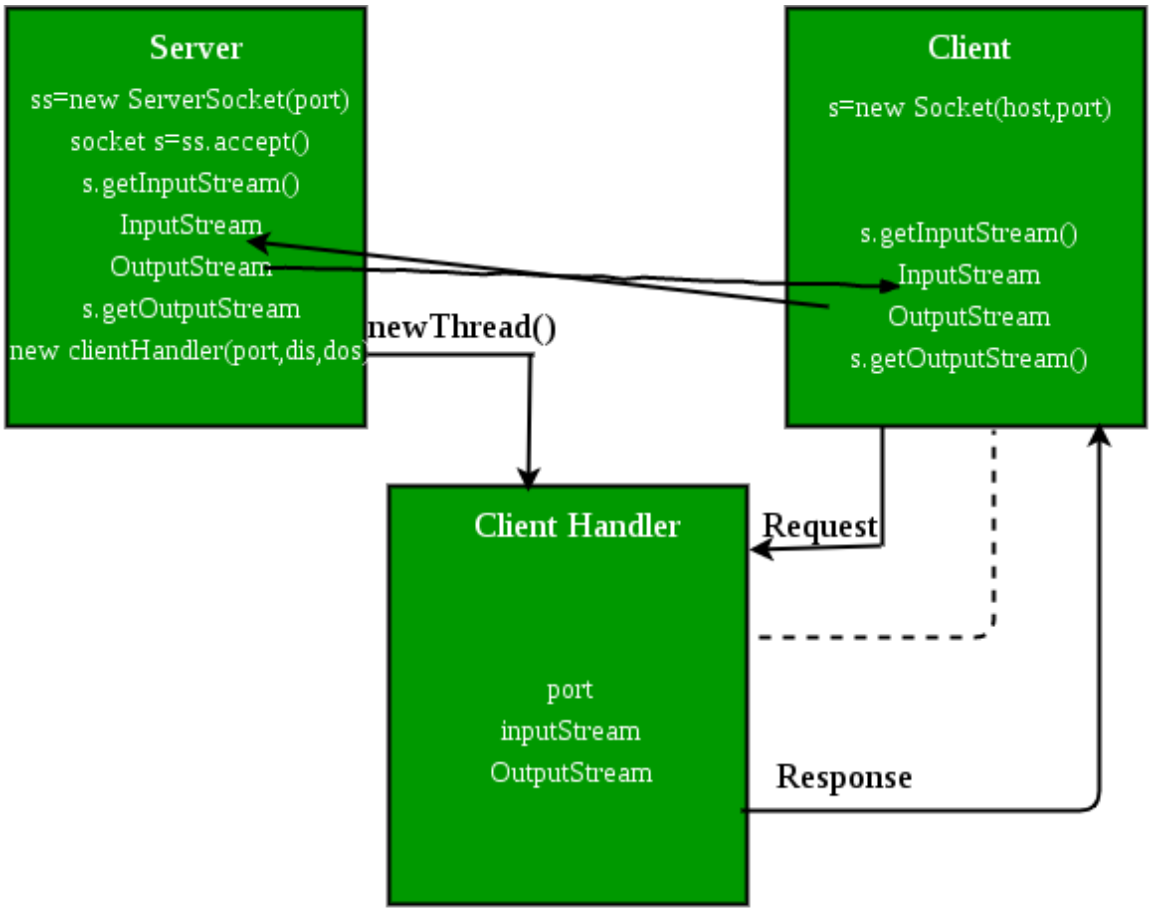
threaded server, every other client had to wait if any problem occurs in the thread.

Disadvantages of Multithreaded Server:

- **Complicated Code:** It is difficult to write the code of the multithreaded server. These programs can not be created easily
- **Debugging is difficult:** Analyzing the main reason and origin of the error is difficult.

Quick Overview

We create two java files, **Client.java** and **Server.java**. Client file contains only one class **Client** (for creating a client). Server file has two classes, **Server**(creates a server) and **ClientHandler**(handles clients using multithreading).



Client-Side Program: A client can communicate with a server using this code. This involves

1. **Establish a Socket Connection**

2. Communication

```
import java.io.*;
import java.net.*;
import java.util.*;

// Client class
class Client {

    // driver code
    public static void main(String[] args)
    {
        // establish a connection by providing host and port
        // number
        try (Socket socket = new Socket("localhost", 1234)) {

            // writing to server
            PrintWriter out = new PrintWriter(
                socket.getOutputStream(), true);

            // reading from server
            BufferedReader in
                = new BufferedReader(new InputStreamReader(
                    socket.getInputStream()));

            // object of scanner class
            Scanner sc = new Scanner(System.in);
            String line = null;

            while (!"exit".equalsIgnoreCase(line)) {

                // reading from user
                line = sc.nextLine();

                // sending the user input to server
                out.println(line);
                out.flush();

                // displaying server reply
                System.out.println("Server replied "
                                   + in.readLine());
            }

            // closing the scanner object
            sc.close();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Server-Side Program: When a new client is connected, and he sends the message to the server.

1. Server class: The steps involved on the server side are similar to the article Socket Programming in Java with a slight change to create the thread object after obtaining the streams and port number.

- **Establishing the Connection:** Server socket object is initialized and inside a while loop a socket object continuously accepts an incoming connection.
- **Obtaining the Streams:** The inputstream object and outputstream object is extracted from the current requests' socket object.
- **Creating a handler object:** After obtaining the streams and port number, a new clientHandler object (the above class) is created with these parameters.
- **Invoking the start() method:** The start() method is invoked on this newly created thread object.

2. ClientHandler class: As we will be using separate threads for each request, let's understand the working and implementation of the **ClientHandler** class implementing Runnable. An object of this class acts as a Runnable target for a new thread.

- First, this class implements Runnable interface so that it can be passed as a Runnable target while creating a new Thread.
- Secondly, the constructor of this class takes a parameter, which can uniquely identify any incoming request, i.e. a **Socket**.
- Inside the **run()** method of this class, it reads the client's message and replies.

```
import java.io.*;
import java.net.*;

// Server class
class Server {
    public static void main(String[] args)
    {
        ServerSocket server = null;

        try {
```

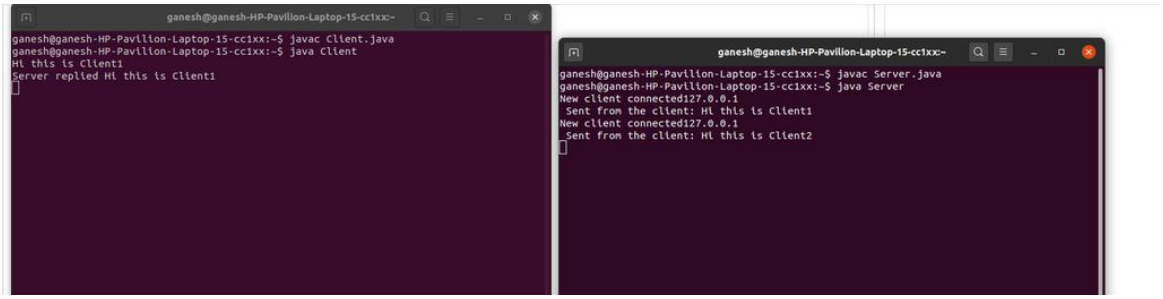
```
            // server is listening on port 1234
            server = new ServerSocket(1234);
            server.setReuseAddress(true);

            // running infinite loop for getting
            // client request
            while (true) {

                // socket object to receive incoming client
                // requests
                Socket client = server.accept();

                // Displaying that new client is connected
                // to server
                System.out.println("New client connected"
                                   + client.getInetAddress()
                                         .getHostAddress());

                // create a new thread object
                ClientHandler clientSock
                    = new ClientHandler(client);

                // This thread will handle the client
                // separately
                new Thread(clientSock).start();
            }
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        finally {
            if (server != null) {
                try {
                    server.close();
                }
                catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }

    // ClientHandler class
    private static class ClientHandler implements Runnable {
        private final Socket clientSocket;

        // Constructor
        public ClientHandler(Socket socket)
        {
            this.clientSocket = socket;
        }

        public void run()
        {
            PrintWriter out = null;
            BufferedReader in = null;
            try {
```

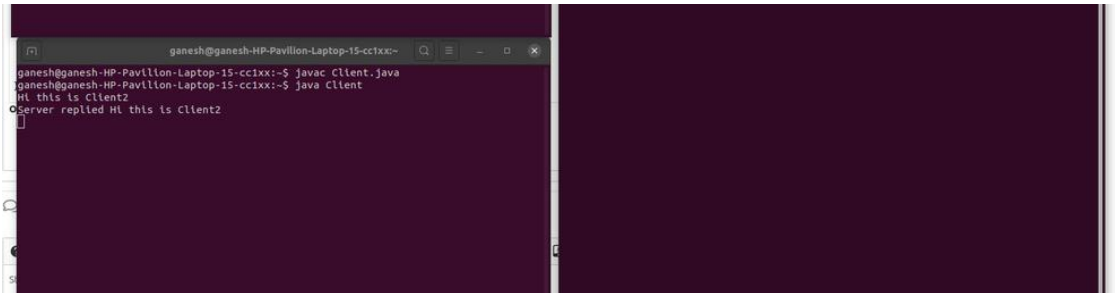
```
                // get the outputstream of client
                out = new PrintWriter(
                    clientSocket.getOutputStream(), true);

                // get the inputstream of client
                in = new BufferedReader(
                    new InputStreamReader(
                        clientSocket.getInputStream()));

                String line;
                while ((line = in.readLine()) != null) {

                    // writing the received message from
                    // client
                    System.out.printf(
                        " Sent from the client: %s\n",
                        line);
                    out.println(line);
                }
            }
            catch (IOException e) {
                e.printStackTrace();
            }
            finally {
                try {
                    if (out != null) {
                        out.close();
                    }
                    if (in != null) {
                        in.close();
                        clientSocket.close();
                    }
                }
                catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Steps:

- Compile both Client and Server programs.
- Run the server first and then the Client.

Output




ganesh@ganesh-HP-Pavilion-Laptop-15-cc1xx:~
ganesh@ganesh-HP-Pavilion-Laptop-15-cc1xx:~$ javac Client.java
ganesh@ganesh-HP-Pavilion-Laptop-15-cc1xx:~$ java Client
Hi this is Client2
Server replied Hi this is Client2