java.net.ServerSocket Class in Java

Last Updated : 20 Jul, 2021

ServerSocket Class is used for providing system-independent implementation of the server-side of a client/server Socket Connection. The constructor for ServerSocket throws an exception if it can't listen on the specified port (for example, the port is already being used).

It is widely used so the applications of java.net.ServerSocket class which is as follows:

1. In java.nio channel, ServerSocket class is used for retrieving a serverSocket associated with this channel.
2. In java.rmi.Server, ServerSocket class is used to create a server socket on the specified port (port 0 indicates an anonymous port).
3. In javax.net, Server socket is used widely so as to:
   - return an unbound server socket.
   - return a server socket bound to the specified port.
   - return a server socket bound to the specified port, and uses the specified connection backlog.
   - return a server socket bound to the specified port, with a specified listen backlog and local IP.

Let us do go through the methods of this class which is as follows:

Method	Description
accept()	Listens for a connection to be made to this socket and accepts it.
bind(SocketAddress endpoint)	Binds the ServerSocket to a specific address (IP address and port number).
bind(SocketAddress endpoint, int backlog)	Binds the ServerSocket to a specific address (IP address and port number).
close()	Closes this socket
getChannel()	Returns the unique ServerSocketChannel object associated with this socket, if any.
getInetAddress()	Returns the local address of this server socket.

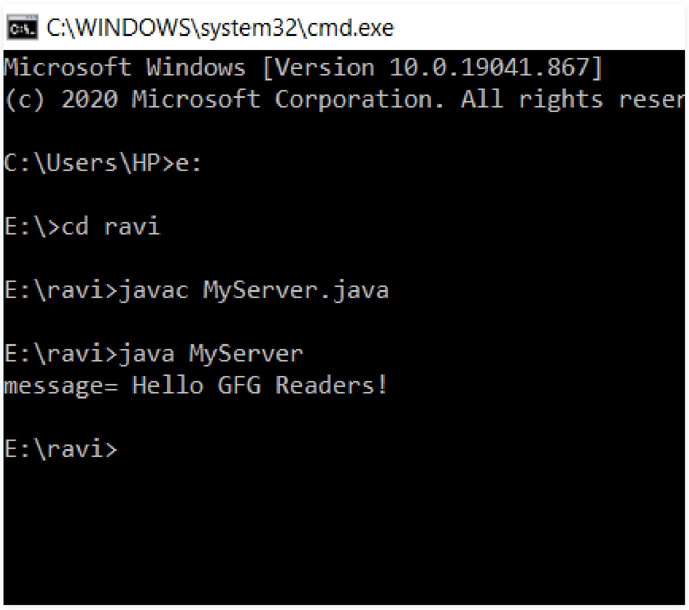
Method	Description
getLocalPort()	Returns the port number on which this socket is listening.
getLocalSocketAddress()	Returns the address of the endpoint this socket is bound to, or null if it is not bound yet.
getReceiveBufferSize()	Gets the value of the SO_RCVBUF option for this ServerSocket, that is the proposed buffer size that will be used for Sockets accepted from this ServerSocket.
getReuseAddress()	Tests if SO_REUSEADDR is enabled.
getSoTimeout()	Retrieve setting for SO_TIMEOUT.
implAccept(Socket s)	Subclasses of ServerSocket use this method to override accept() to return their own subclass of **the** socket.
isBound()	Returns the binding state of the ServerSocket.
isClosed()	Returns the closed state of the ServerSocket.
setPerformancePreferences(int connectionTime, int latency, int bandwidth)	Sets performance preferences for this ServerSocket
Sets performance preferences for this ServerSocket	Sets a default proposed value for the SO_RCVBUF option for sockets accepted from this ServerSocket.
setReuseAddress(boolean on)	Enable/disable the SO_REUSEADDR socket option.
setSocketFactory(SocketImplFactory fac)	Sets the server socket implementation factory for the application.
setSoTimeout(int timeout)	Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.
toString()	Returns the implementation address and implementation port of this socket as a String.

implementation:

Example 1 Server-Side

```java
// Java Program to implement ServerSocket class
// Server Side

// Importing required libraries
import java.io.*;
import java.net.*;

// Main class
public class MyServer {

    // Main driver method
    public static void main(String[] args)
    {

        // Try block to check for exceptions
        try {

            // Creating an object of ServerSocket class
            // in the main() method  for socket connection
            ServerSocket ss = new ServerSocket(6666);

            // Establishing a connection
            Socket soc = ss.accept();

            // Invoking input stream via getInputStream()
            // method by creating DataInputStream class
            // object
            DataInputStream dis
                = new DataInputStream(s.getInputStream());

            String str = (String)dis.readUTF();

            // Display the string on the console
            System.out.println("message= " + str);

            // Lastly close the socket using standard close
            // method to release memory resources
            ss.close();
        }

        // Catch block to handle the exceptions
        catch (Exception e) {

            // Display the exception on the console
            System.out.println(e);
        }
    }
}
```

Output:

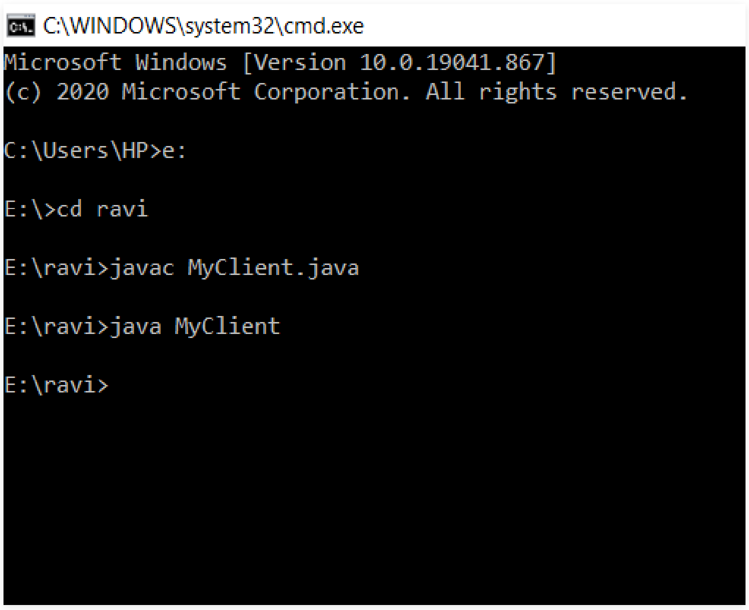


```
Microsoft Windows [Version 10.0.19041.867]
(c) 2020 Microsoft Corporation. All rights reser

C:\Users\HP>e:

E:\>cd ravi

E:\ravi>javac MyServer.java

E:\ravi>java MyServer
message= Hello GFG Readers!

E:\ravi>
```

Example 2 Client-Side

```java
// Java Program to implement ServerSocket class
// Client - side

// Importing required libraries
import java.io.*;
import java.net.*;

// Main class
public class MyClient {

    // Main driver method
    public static void main(String[] args)
    {

        // Try block to check if exception occurs
        try {

            // Creating Socket class object and
            // initializing Socket
            Socket soc = new Socket("localhost", 6666);

            DataOutputStream d = new DataOutputStream(
                soc.getOutputStream());

            // Message to be displayed
            d.writeUTF("Hello GFG Readers!");

            // Flushing out internal buffers,
            // optimizing for better performance
            d.flush();

            // Closing the connections

            // Closing DataOutputStream
            d.close();
            // Closing socket
            soc.close();
        }

        // Catch block to handle exceptions
        catch (Exception e) {

            // Print the exception on the console
            System.out.println(e);
        }
    }
}
```

Output:



```
Microsoft Windows [Version 10.0.19041.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\HP>e:

E:\>cd ravi

E:\ravi>javac MyClient.java

E:\ravi>java MyClient

E:\ravi>
```