LAB 02: InetAddress and TCP

Contents

The InetAddress Class

- The class $InetAddress$ handles Internet addresses both as host names and as IP addresses.
  https://docs.oracle.com/javase/7/docs/api/java/net/InetAddress.html
- Retrieve the IP address of a given host name

IPFinder

```
Scanner input = new Scanner(System.in);

InetAddress address;
System.out.print("Enter Host Name: ");

String host = input.next();

try {
    address = InetAddress.getByName(host);
    System.out.println("IP address: " + address);
} catch (UnknownHostException ex) {
    System.out.println("Couldn't find the host");
}
```

```
Enter Host Name: www.luxor.edu.eg
IP address: www.luxor.edu.eg/193.227.53.75
```

- Static method $getByName$ of this class uses DNS (Domain Name System) to return the Internet address of a specified host name as an $InetAddress$ object.
- The $getByName$ throws the checked exception $UnknownHostException$ if the host name is not recognized.
- Retrieve the IP address of the current machine.

MyLocalIPAddress

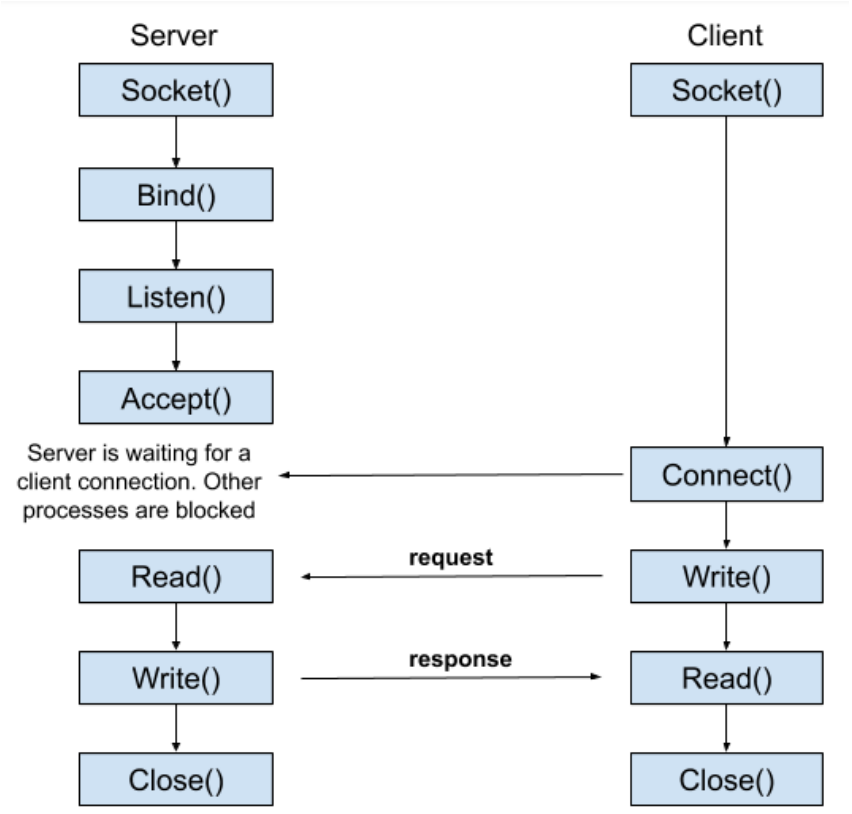
```
InetAddress address = null;
try {
    address = InetAddress.getLocalHost();
    System.out.println(address);
} catch (UnknownHostException e) {
    System.out.println("Could not find the IP address of
the local host");
}
```

TCP Socket

- Java implements both TCP/IP sockets and datagram sockets (UDP sockets).
- To create a TCP socket for the server:

Step	Code
1. Create a *ServerSocket* object.	$ServerSocket\ serverSocket\ =\ new\ ServerSocket(1234);$
2. Put the server into a waiting state.	$Socket\ link\ =\ serverSocket.accept();$
3. Set up input and output streams.	$Scanner\ input\ =\ new\ Scanner(link.getInputStream());$ $PrintWriter\ output\ =\ new\ PrintWriter(link.getOutputStream(), true);$
4. Send and receive data.	$output.println("Awaiting\ data\ \ldots");$ $String\ input\ =\ input.nextLine();$
5. Close the connection.	$link.close();$

- TCP connection



TCP Server

TCPEchoServer

```
public class TCPEchoServer {
    private static ServerSocket serverSocket;
    private static final int PORT = 1234;

    public static void main(String[] args) {
        System.out.println("Opening port ...");
        try {
            serverSocket = new ServerSocket(PORT);
            System.out.println("Port opened successfully...");
        } catch (IOException ioex) {
            System.out.println("Failed to connect to port: " + PORT);
            System.out.println("Please try another port");
            System.exit(1);
        }

        do {
            handleConnection();
        } while (true);
    }

    private static void handleConnection() {
        Socket link = null;

        try {
            link = serverSocket.accept();
            Scanner input = new Scanner(link.getInputStream());
            PrintWriter output = new PrintWriter(link.getOutputStream(), true);

            int num_msgs = 0;
            String msg = input.nextLine();

            while (!msg.equals("***CLOSE***")) {
                System.out.println("Message Received");
                num_msgs++;
                output.println("Message " + num_msgs + ": " + msg);
                msg = input.nextLine();
            }
            output.println(num_msgs + " messages received");
        } catch (IOException ioex) {
            ioex.printStackTrace();
        } finally {
            try {
                System.out.println("\n*Closing Connection...*\n");
                link.close();
            } catch (IOException ioex) {
                System.out.println("Unable to disconnect!");
                System.exit(1);
            }
        }
    }
```

- Step-by-step TCP server code:
  - Create a class named $TCPEchoServer$ with $main()$ method.

```
public class TCPEchoServer {
    public static void main(String[] args) {

    }
}
```

  - Create two static fields: $ServerSocket$ object for waiting for client requests, and a constant integer $PORT$ to define port number for the service.

```
public class TCPEchoServer {
    private static ServerSocket serverSocket;
    private static final int PORT = 1234;

    public static void main(String[] args) {

    }
}
```

  - Inside $main$, setup $try - catch$ block
  - Inside try block, initialize $serverSocket$ object to listen for any incoming connection through port number $PORT$.
  - Catch $IOException$ inside $catch$ block.
  - Print appropriate messages for the user.

```
System.out.println("Opening port ...");
     try {
     serverSocket = new ServerSocket(PORT);
     System.out.println("Port opened successfully...");
 } catch (IOException ioex) {
     System.out.println("Failed to connect to port: " +
PORT);
     System.exit(1);
 }
```

- After $try - catch$ block, define a $do - while$ loop to run the $handleConnection()$ method.

```
public static void main(String[] args) {
    System.out.println("Opening port ...");
    try {
        serverSocket = new ServerSocket(PORT);
        System.out.println("Port opened successfully...");
    } catch (IOException ioex) {
        System.out.println("Failed to connect to port: " +
PORT);
        System.out.println("Please try another port");
        System.exit(1);
    }

    do {
        handleConnection();
    } while (true);
}
```

- Inside the class, define $static$ method for handling incoming connections.

```
public static void main(String[] args) {…}

private static void handleConnection(){

}
```

- Inside $handleConnection$, define $Socket$ object that will be our link between server and client.
- Set up $try - catch - finally$ block.

```
private static void handleConnection(){
    Socket link = null;

    try{

    }catch (){

    }finally {

    }
}
```

- Inside try block, initialize $link$ object to accept incoming connection, create $Scanner$ object to get input stream from $link$, and create $PrintWriter$ object to set output stream to link.
- Create an integer variable to count the number of messages.
- Create a $String$ object to receive client message.

```
try {
    link = serverSocket.accept();
    Scanner input = new Scanner(link.getInputStream());
    PrintWriter output = new
PrintWriter(link.getOutputStream(), true);

    int num_msgs = 0;
    String msg = input.nextLine();
```

- Define a $while$ loop, with the condition that the incoming message is not equal to $*** close ***$.
- Receive a message from the client, increment message counter, send a message to client, and read the incoming message again.
- If the $while$ loop is finished, we print the number of messages received to the server's console.

```
while (!msg.equals("***CLOSE***")) {
    System.out.println("Message Received");
    num_msgs++;
    output.println("Message " + num_msgs + ": " + msg);
    msg = input.nextLine();
}
output.println(num_msgs + " messages received");
```

- In the $catch$ block, catch the exception.
- In the $finally$ block, define a $try - catch$ block to close the link.

```
catch (IOException ioex) {
        ioex.printStackTrace();
    } finally {
        try {
            System.out.println("\n*Closing
Connection...*\n");
            link.close();
        } catch (IOException ioex) {
            System.out.println("Unable to disconnect!");
            System.exit(1);
        }
    }
```

TCP Client

Step	Code
1. Establish a connection to the server.	$Socket\ link\ =\ new\ Socket(InetAddress.getLocalHost()\ ,1234);$
2. Set up input and output streams.	$Scanner\ input\ =\ new\ Scanner(link.getInputStream());$ $PrintWriter\ output\ =\ new\ PrintWriter(link.getOutputStream(),\ true);$
3. Send and receive data.	$output.println();$ $input.nextLine();$
4. Close the connection.	$link.close();$

- Client accepts messages from server.

TCPEchoClient

```
public class TCPEchoClient {
    private static InetAddress host;
    private static final int PORT = 1234;

    public static void main(String[] args) {
        try {
            host = InetAddress.getLocalHost();
        } catch (IOException ioex) {
            System.out.println("Host ID not found!");
            System.exit(1);
        }
        accessServer();
    }

    private static void accessServer() {
        Socket link = null;
        try{
            link = new Socket(host, PORT);
            Scanner input = new Scanner(link.getInputStream());
            PrintWriter output = new PrintWriter(link.getOutputStream(), true);

            Scanner userInput = new Scanner(System.in);
            String msg, respns;

            do {
                System.out.print("Enter a message: ");
                msg = userInput.nextLine();
                output.println(msg);
                respns = input.nextLine();
                System.out.println("\nSERVER> "+respns);
            }while (!msg.equals("***CLOSE***"));
        } catch (IOException e) {
            e.printStackTrace();
        }finally {
            try {
                System.out.println("CLOSING CONNECTION ...");
                link.close();
            } catch (IOException e) {
                System.out.println("Unable to disconnect");
                System.exit(1);
            }
        }
    }
```

- Step-by-Step TCP client code:
  - Create a $TCPEchoClient$ class with $main$ method.
  - Define static variables: $InetAddress$ object that represents IP address of the machine, and integer port number $PORT$.
  - Inside $main$, create $try - catch$ block

```
public class TCPEchoClient {
    private static InetAddress host;
    private static final int PORT = 1234;

    public static void main(String[] args) {
        try {
        } catch () {

        }
    }
}
```

  - Because our own machine will act as both the client and the server, the server address has the same IP address of the client machine, which is the local host machine. Initialize $host$ object to get local host address, then catch the exception.
  - Call the method that will access the server.

```
public static void main(String[] args) {
    try {
        host = InetAddress.getLocalHost();
    } catch (IOException ioex) {
        System.out.println("Host ID not found!");
        System.exit(1);
    }
    accessServer();
}
```

- Create static $accessServer$ method that will be used to connect to TCP server.
- Within $accessServer$ method, create $Socket$ object that will be our connection link with the server.
- Create $try - catch - finally$ block.

```
private static void accessServer() {
    Socket link = null;
    try{

    } catch () {

    }finally {

    }
}
```

- Within try block, initialize $link$ object as a $Socket$ object that gets server address as $host$ variable and service port as $PORT$ variable.
- Create $Scanner$ object to get input stream from $link$ object, then define $PrintWriter$ object to get output stream from $link$ object.
- Create another $Scanner$ object that will get user input from client's console.
- Create two $String$ object, one for sending messages and the other for receiving the message.

```
try{
       link = new Socket(host, PORT);
       Scanner input = new Scanner(link.getInputStream());
       PrintWriter output = new
PrintWriter(link.getOutputStream(), true);

       Scanner userInput = new Scanner(System.in);
       String msg, respns;
```

- Setup a $do - while$ loop in which we accept the user input from console, send the message to the server, accept a response from the server, and print the response on the client's console.
- The loop runs if the user input is not $*** CLOSE ***$.

```
do {
    System.out.print("Enter a message: ");
    msg = userInput.nextLine();
    output.println(msg);
    respns = input.nextLine();
    System.out.println("\nSERVER> "+respns);
}while (!msg.equals("***CLOSE***"));
```

- Catch the exception in $catch$ block.
- Within the $finally$ block create $try - catch$ block to close the $link$ object.

```
catch (IOException e) {
        e.printStackTrace();
    }finally {
        try {
            System.out.println("CLOSING CONNECTION ...");
            link.close();
        } catch (IOException e) {
            System.out.println("Unable to disconnect");
            System.exit(1);
        }
    }
```

- Now run the server then run the client.

Exercise

1. Create a TCP server that accepts a username and password from a client. If the username and password are valid, the server echoes back user messages. Otherwise, the server prompts the client to enter the username and password again.
2. Summarize the methods of $DatagramSocket$ and $DatagramPacket$ classes.
3. What is $Socket$ class.
4. What is $InetAddress$ class is used for?
5. Write a java application for printing the open ports of the local machine.

❖ Create the same server-client application explained above but with GUI.