Lecture 3 - Java Socket Programming and I/O

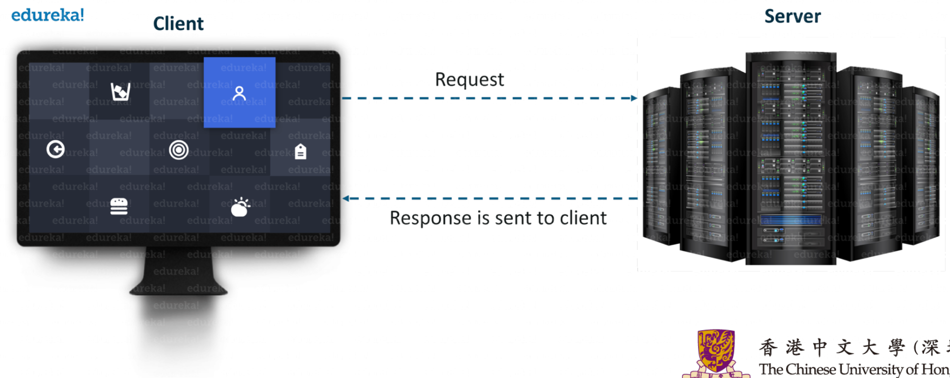
Guiliang Liu

The Chinese University of Hong Kong, Shenzhen

CSC-1004: Computational Laboratory Using Java
Course Page: [Click]

Introduction

Java Socket programming is used for communication between the applications running on different machines.

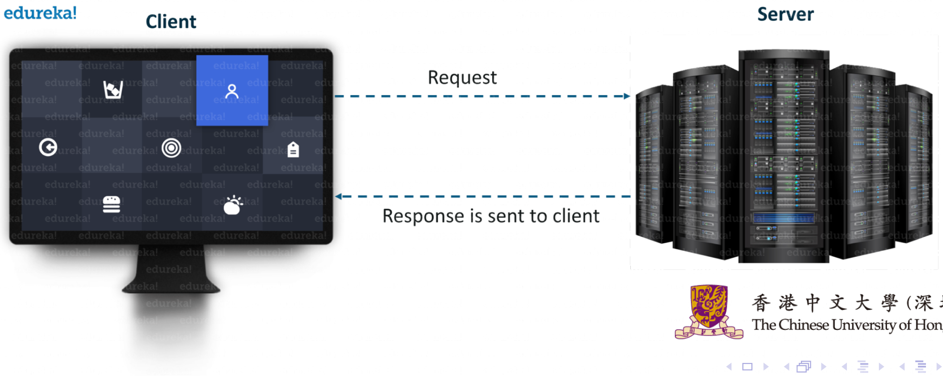


The Chinese University of Hong Kong, Shenzhen
香 港 中 文 大 學 (深 圳)

Introduction

In Java Socket programming, the most important elements are:

- **Socket and ServerSocket** are used for connection-oriented socket programming.
- **The client** must know the IP Address of the server and port number.



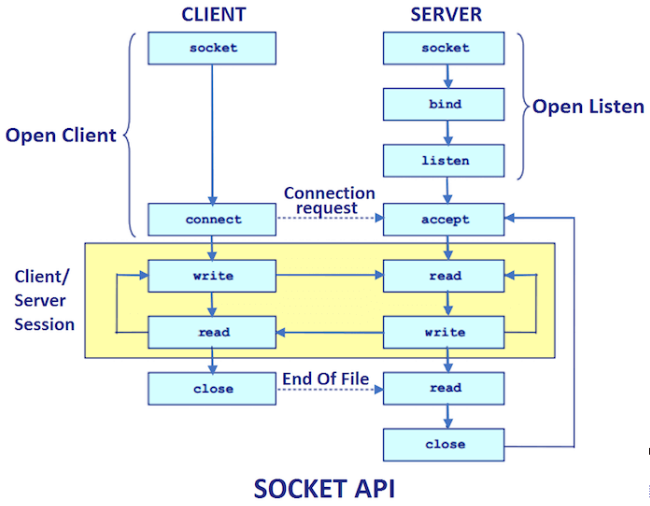
香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Introduction

One-way client and server communication: the client sends a message to the server, and the server reads the message and prints it.

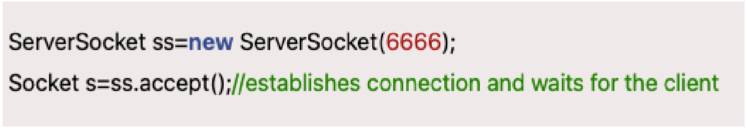
The Socket class communicates client and server.

The ServerSocket class is used on the server side. Its accept() method blocks the console until the client is connected.



Example of Java Socket Programming

- **Creating Server:** 1) We use the 6666 port number for the communication between the client and server. 2) The accept() method waits for the client. If clients connect with the given port number, it returns an instance of Socket.

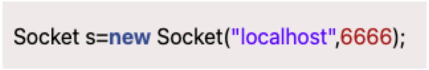


```
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();//establishes connection and waits for the client
```

香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

Example of Java Socket Programming

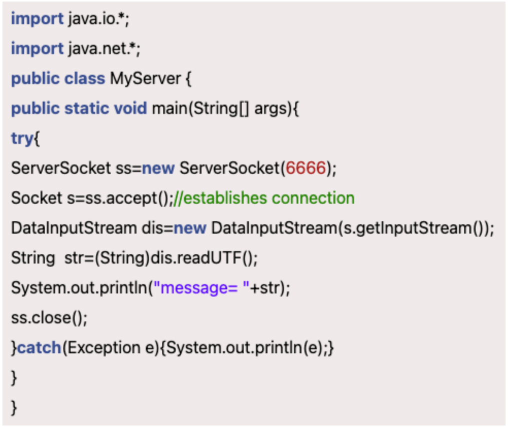
- **Creating Client:** We pass the IP address or hostname of the Server and a port number. Here, we are using "localhost" because our server is running on the same system.



```java
Socket s=new Socket("localhost",6666);
```

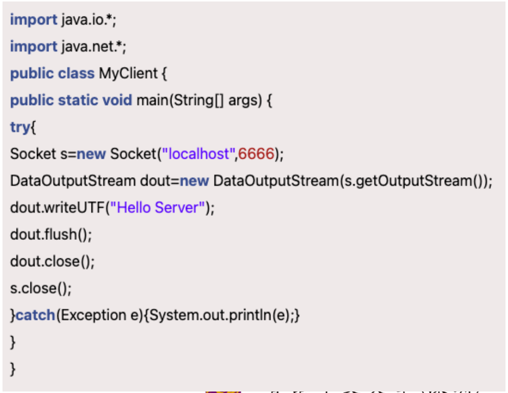
Example of Java Socket Programming

- MyServer.java



```
import java.io.*;
import java.net.*;
public class MyServer {
public static void main(String[] args){
try{
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();//establishes connection
DataInputStream dis=new DataInputStream(s.getInputStream());
String  str=(String)dis.readUTF();
System.out.println("message= "+str);
ss.close();
}catch(Exception e){System.out.println(e);}
}
}
```

- MyClient.java



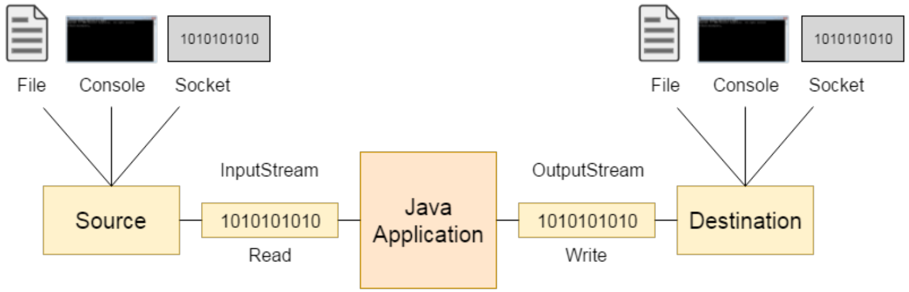
```
import java.io.*;
import java.net.*;
public class MyClient {
public static void main(String[] args) {
try{
Socket s=new Socket("localhost",6666);
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
dout.writeUTF("Hello Server");
dout.flush();
dout.close();
s.close();
}catch(Exception e){System.out.println(e);}
}
}
```

Java I/O (Input and Output)

Basic Knowledge of**Java Input and Output**.

- **Stream:** A stream is a sequence of data. In Java, a stream is composed of bytes.
- **OutputStream:** Java application uses an output stream to write data to a destination; it may be a file, an array or sockets.
- **InputStream:** Java application uses an input stream to read data from a source; it may be a file, an array, or sockets.

Java I/O (Input and Output)



Java I/O (Input and Output)

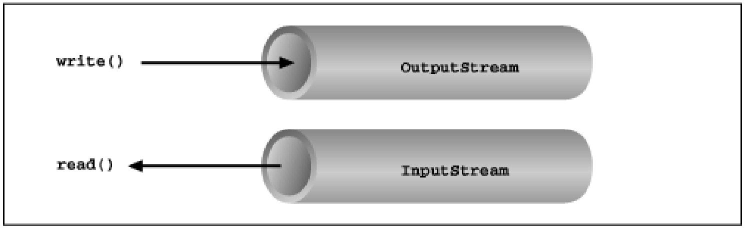
Stream Wrappers

What if we want to do more than read and write a mess of bytes or characters?

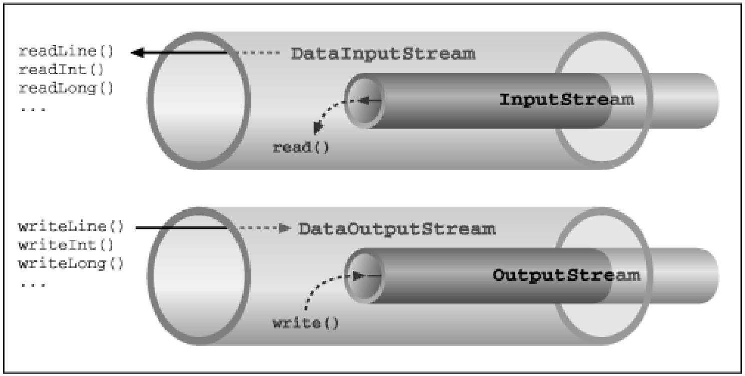
- The **DataInputStream** and **DataOutputStream** Classes are filtered streams that read or write strings and primitive data types that comprise more than a byte.
- The **readUTF()** and **writeUTF()** methods of DataInputStream and DataOutputStream read and write a Java String of Unicode characters using the UTF-8 (encoding of Unicode characters commonly used for the transmission and storage of Unicode text).
- Use the **flush()** method to data out the contents.

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Java I/O (Input and Output)



Java I/O (Input and Output)



香 港 中 文 大 學 (深 圳)
The Chinese University of Hong Kong, Shenzhen

Java I/O (Input and Output)

Character Streams

- The **InputStreamReader** class bridge the gap between the world of character streams and the world of byte streams. These are character streams that are wrapped around an underlying byte stream.
- When we wrap an InputStreamReader around System.in, we object converts the incoming bytes of System.in to characters using the default encoding scheme.

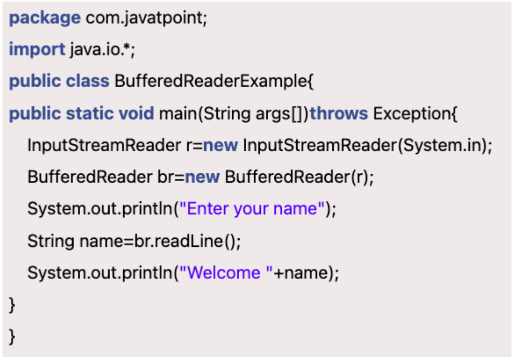
Java I/O (Input and Output)

Buffered streams

- The BufferedReader class add a data buffer of a specified size to the stream path.
- A buffer can increase efficiency by reducing the number of physical read or write operations that correspond to read() or write() method calls.
- You can wrap another stream around a buffered stream. BufferedReader gives us the readLine() method, which we can use to retrieve a full line of text into a String.

Java I/O (Input and Output)

Read data from the console.



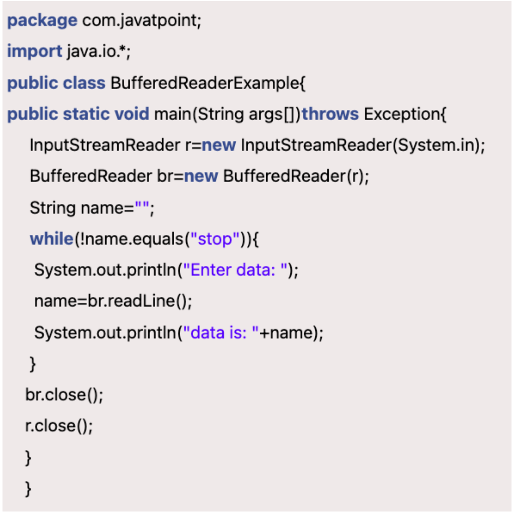
```
package com.javatpoint;
import java.io.*;
public class BufferedReaderExample{
public static void main(String args[])throws Exception{
    InputStreamReader r=new InputStreamReader(System.in);
    BufferedReader br=new BufferedReader(r);
    System.out.println("Enter your name");
    String name=br.readLine();
    System.out.println("Welcome "+name);
}
}
```

```
Enter your
Nakul Jain
Welcome Nakul Jain
```

香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

Java I/O (Input and Output)

Read data from console until the user writes stop



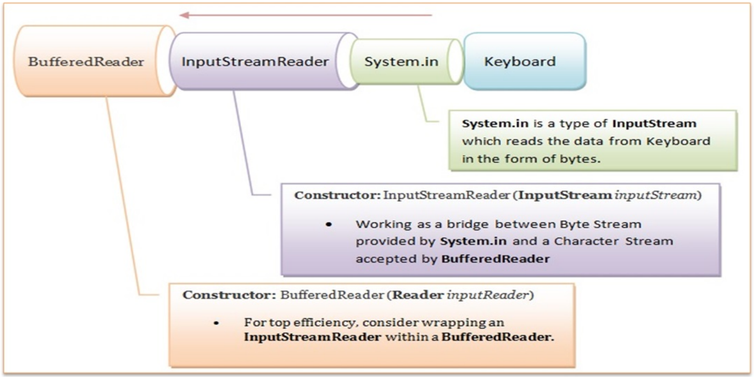
```
package com.javatpoint;
import java.io.*;
public class BufferedReaderExample{
public static void main(String args[])throws Exception{
   InputStreamReader r=new InputStreamReader(System.in);
   BufferedReader br=new BufferedReader(r);
   String name="";
   while(!name.equals("stop")){
    System.out.println("Enter data: ");
    name=br.readLine();
    System.out.println("data is: "+name);
   }
  br.close();
  r.close();
  }
  }
```

```
Enter data: Nakul
data is: Nakul
Enter data: 12
data is: 12
Enter data: stop
data is: stop
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Java I/O (Input and Output)

Java InputStreamReader – Read from Keyboard



System.in is a type of **InputStream** which reads the data from Keyboard in the form of bytes.

Constructor: InputStreamReader (**InputStream** *inputStream*)

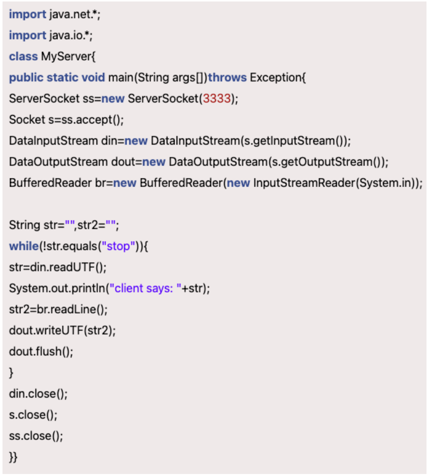
- Working as a bridge between Byte Stream provided by **System.in** and a Character Stream accepted by **BufferedReader**

Constructor: BufferedReader (**Reader** *inputReader*)

- For top efficiency, consider wrapping an **InputStreamReader** within a **BufferedReader.**

Example of Java Socket Programming and Data Streaming

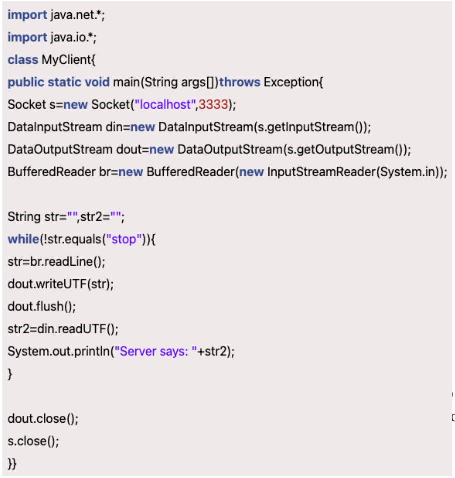
- MyServer.java



```
import java.net.*;
import java.io.*;
class MyServer{
public static void main(String args[])throws Exception{
ServerSocket ss=new ServerSocket(3333);
Socket s=ss.accept();
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

String str="",str2="";
while(!str.equals("stop")){
str=din.readUTF();
System.out.println("client says: "+str);
str2=br.readLine();
dout.writeUTF(str2);
dout.flush();
}
din.close();
s.close();
ss.close();
}}
```

- MyClient.java



```
import java.net.*;
import java.io.*;
class MyClient{
public static void main(String args[])throws Exception{
Socket s=new Socket("localhost",3333);
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

String str="",str2="";
while(!str.equals("stop")){
str=br.readLine();
dout.writeUTF(str);
dout.flush();
str2=din.readUTF();
System.out.println("Server says: "+str2);
}

dout.close();
s.close();
}}
```

Question and Answering (Q&A)