COMP3358: Tutorial 2
Java Networking

Java Connection

- Supported by the package `java.net.*`
- Classes in **java.net** handles all low-level networking details
- We only need to care about:
  - Setting up server and client connection
    - Through **ServerSocket** and **Socket** objects
  - Sending and receiving of data
    - Using I/O streams

Client-server connection

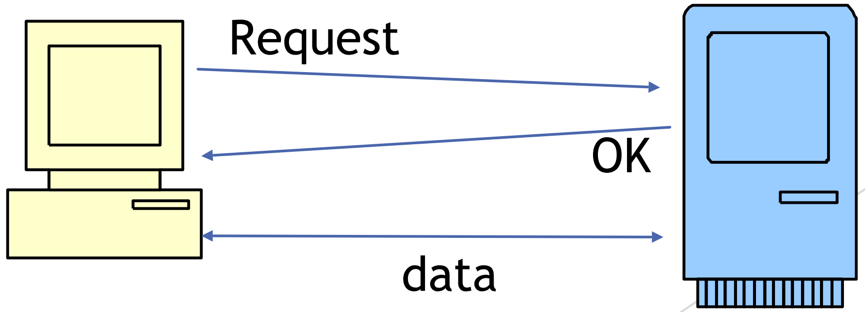
- Server listen to a port
- Client initiate connection to the server at the dedicated port
- Server accepts the connection
- Server and client collect input and output streams for communication

Server

```
ServerSocket ss = new ServerSocket(10000);
Socket s = ss.accept();
InputStream in = s.getInputStream();
OutputStream out = s.getOutputStream();
```

Client

```
Socket s = new Socket('localhost', 10000);
InputStream in = s.getInputStream();
OutputStream out = s.getOutputStream();
```



I/O streams

- `getInputStream()` and `getOutputStream()` will only return low level I/O streams
- Buffered Streams are usually used instead
  - One example will be to use **BufferedReader** for input and **PrintWriter** for output

Server

```
ServerSocket ss = new ServerSocket(10000);
Socket s = ss.accept();
BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));
PrintWriter out = new PrintWriter(s.getOutputStream());
```

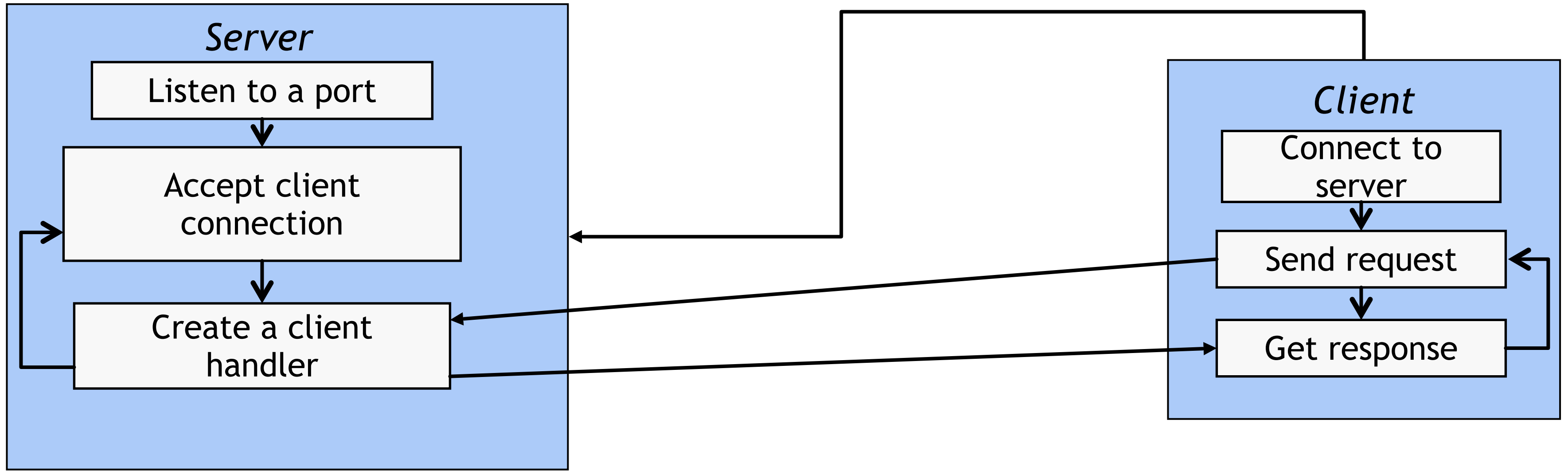
Client

```
Socket s = new Socket('localhost', 10000);
BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));
PrintWriter out = new PrintWriter(s.getOutputStream());
```

Demo: Echo server

- `EchoServer.java` and `EchoClient.java`
- The server can only handle one client at a time (Why?)

Client handling



- How to handle multiple clients at the same time?
  - We need multithreading!

Java Thread

- Two ways to create a thread in Java

Create an object of a subclass of **Thread**

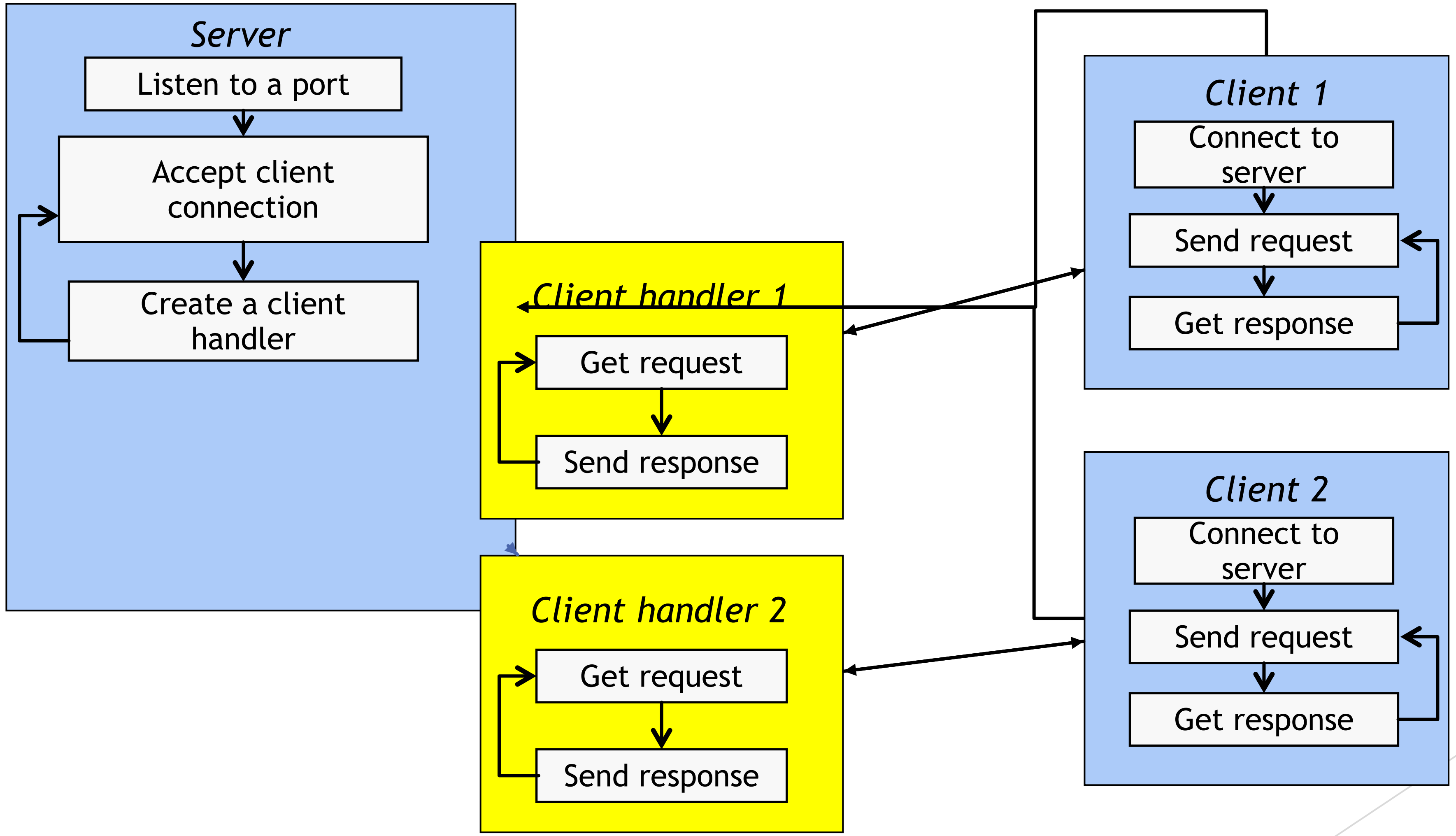
```
class myThread extends Thread {
    public void run() {
        /* code to be executed in a thread */
    }
}
```

```
new myThread().start();
```

Create a **Thread** object using a **Runnable** object

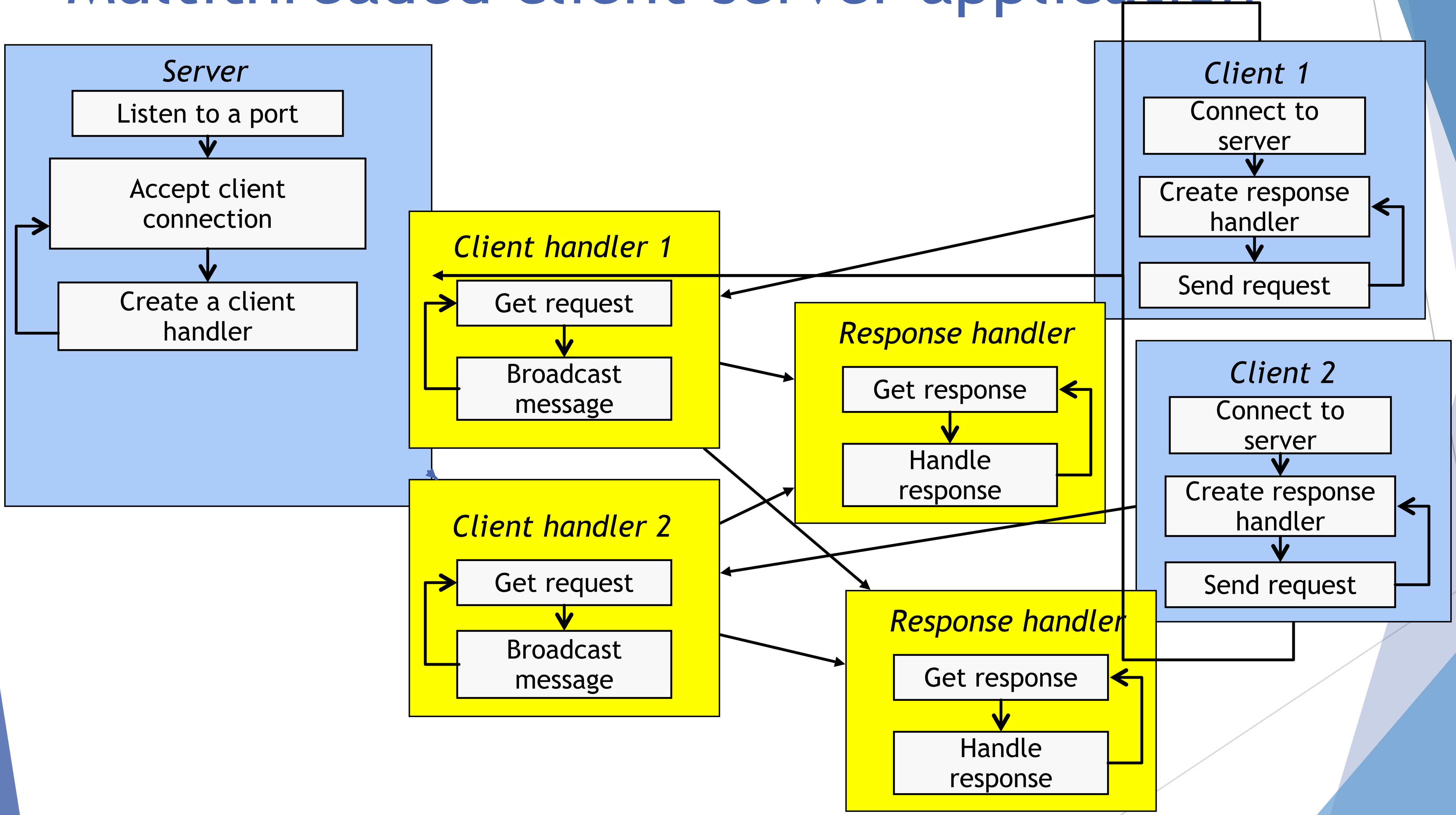
```
Runnable job = new Runnable() {
    public void run() {
        /* code to be executed in a thread */
    }
}
new Thread(job).start();
```

Multithreaded clients handling



Assume every request will get one response only

Multithreaded client-server application



Demo: Chat server

- An **ArrayList** object is used to maintain a list of clients
- The list will be updated when:
  - A **new client** is connected
  - A client **terminates**

- What happen if a number of clients are connecting to the server/terminating at the same time?
  - Try to run `ChatClientTester.java` with `ChatServer.java`
  - We need to avoid concurrent update to the object

Synchronization

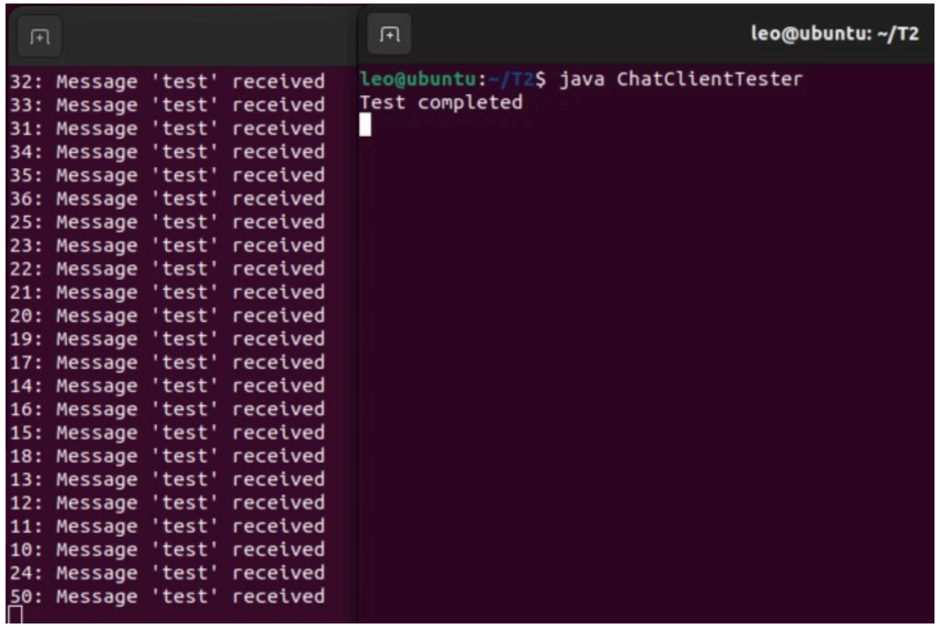
- The synchronized keyword can be added to a method to indicate that the method should only be executed by one thread at a time
- In our example, the **ArrayList** object need to be synchronized, so all method that access such object must be synchronized

```
public synchronized void broadcast(String message) {
        for(ClientHandler client: clients) {
                client.send(message);
        }
}
```

Exercise

- Think about: What other part of the code must be synchronized?
- Modify `ChatServer.java` so that running `ChatClientTest.java` will not cause exception to be thrown in `ChatServer.java`
- **Please do the programming assignments on your virtual machine. Submit the code you have modified and a document (better in pdf format) to Moodle. The doc should contain the highlight of the code you modified (part I) and the screen shots of the java server and client you executed (part II). Example of part II:**

ChatServer



```
32: Message 'test' received
33: Message 'test' received
31: Message 'test' received
34: Message 'test' received
35: Message 'test' received
36: Message 'test' received
25: Message 'test' received
23: Message 'test' received
22: Message 'test' received
21: Message 'test' received
20: Message 'test' received
19: Message 'test' received
17: Message 'test' received
14: Message 'test' received
16: Message 'test' received
15: Message 'test' received
18: Message 'test' received
13: Message 'test' received
12: Message 'test' received
11: Message 'test' received
10: Message 'test' received
24: Message 'test' received
50: Message 'test' received
```

```
leo@ubuntu:~/T2$ java ChatClientTester
Test completed
```

Tester