SERVER-SIDE

MICROSERVICES

THE ARCHITECTURE
TCP PROTOCOL
TCP SERVICES

VERSION

v1.2	23 September 2021	Split presentation in 2. Part II, here.
v1.1	21 September 2021	Revision to TCP Services Section
v1.0	14 September 2021	Initial Release

ACKNOWLEDGMENTS

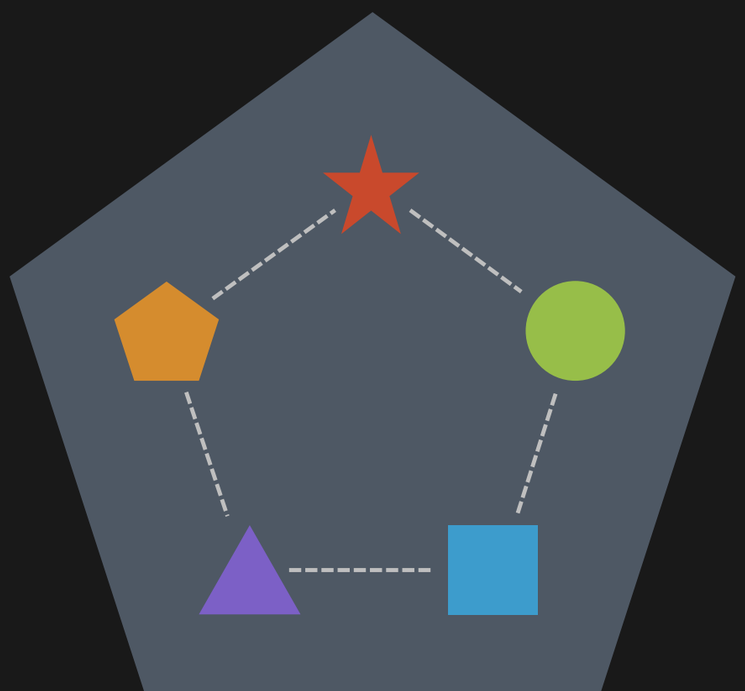
THANKS TO:

- Hamzeh Roumani, who has shaped EECS-4413 into a leading hands-on CS course at EECS and who generously shared all of his course materials and, more importantly, his teaching philosophy with me;
- Parke Godfrey, my long-suffering Master's supervisor and mentor; and
- Suprakash Datta for giving me this opportunity to teach this course.

PRINTABLE VERSION OF THE TALK

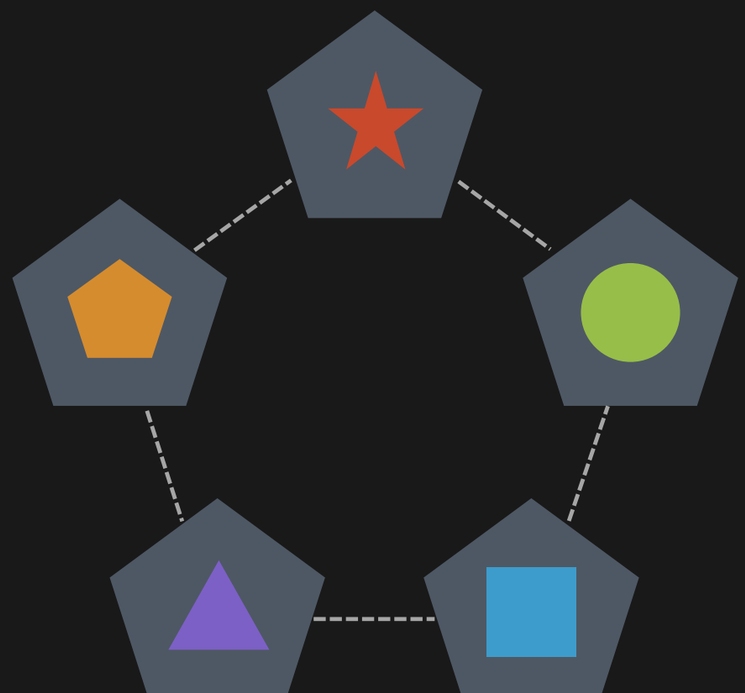
Download PDF

MONOLITHIC



VS

MICROSERVICES



Source: Microservices vs Monolith: which architecture is the best choice for your business? by Romana Gnatyk, 03 October 2018.

MONOLITHIC ARCHITECTURE

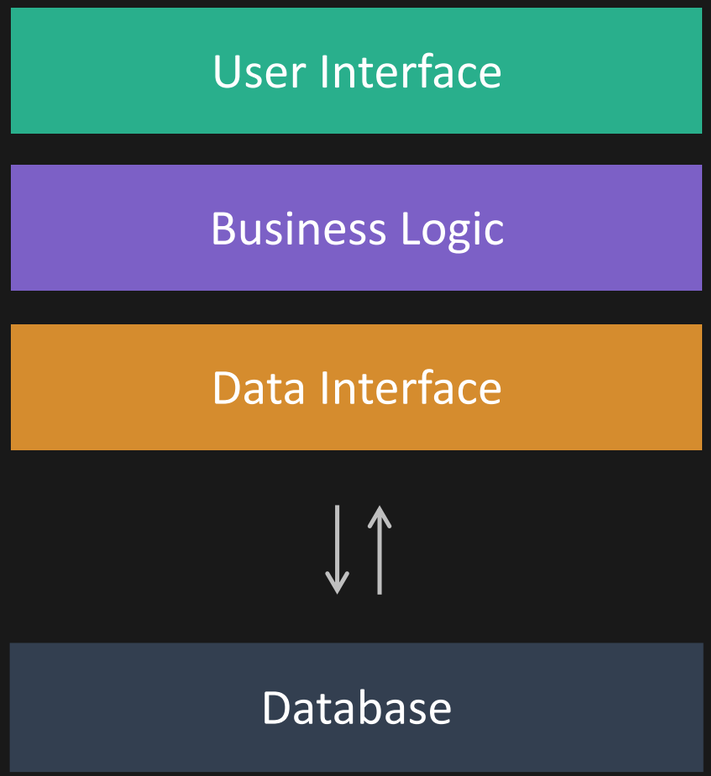
Monolithic architecture is considered to be a traditional way of building applications. A monolithic application is built as a single and indivisible unit. Usually, such a solution comprises a client-side user interface, a server side-application, and a database. It is unified and all the functions are managed and served in one place.

STRENGTHS

- Less cross-cutting concerns:
  - Logging,
  - Handling,
  - Caching,
  - Performance monitoring, and
  - Security.
- Easier debugging and testing
- Simple to deploy
- Simple to develop

WEAKNESSES

- Understanding and complexity
- Making changes
- Scalability
- New technology barriers

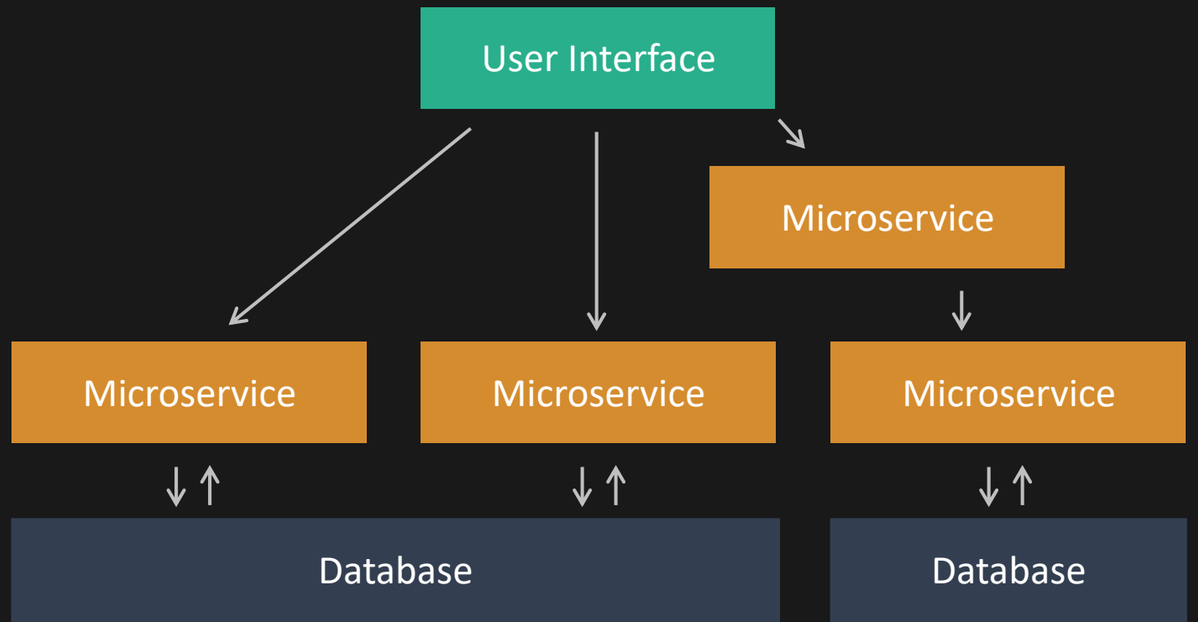


Source: Microservices vs Monolith: which architecture is the best choice for your business? by Romana Gnatyk, 03 October 2018.

MICROSERVICE ARCHITECTURE

A microservices architecture breaks the application down into a collection of smaller independent units. These units carry out every application process as a separate service. So all the services have their own logic and access the database separately to perform specific functions. They are multithreaded, pooled, containerized, and each on separate nodes.

Functionality is divided into independently deployable modules that communicate with each other through APIs over technology-agnostic protocols such as HTTP. Each service has its own scope and can be updated, deployed, and scaled independently.



Source: Microservices vs Monolith: which architecture is the best choice for your business? by Romana Gnatyk, 03 October 2018.

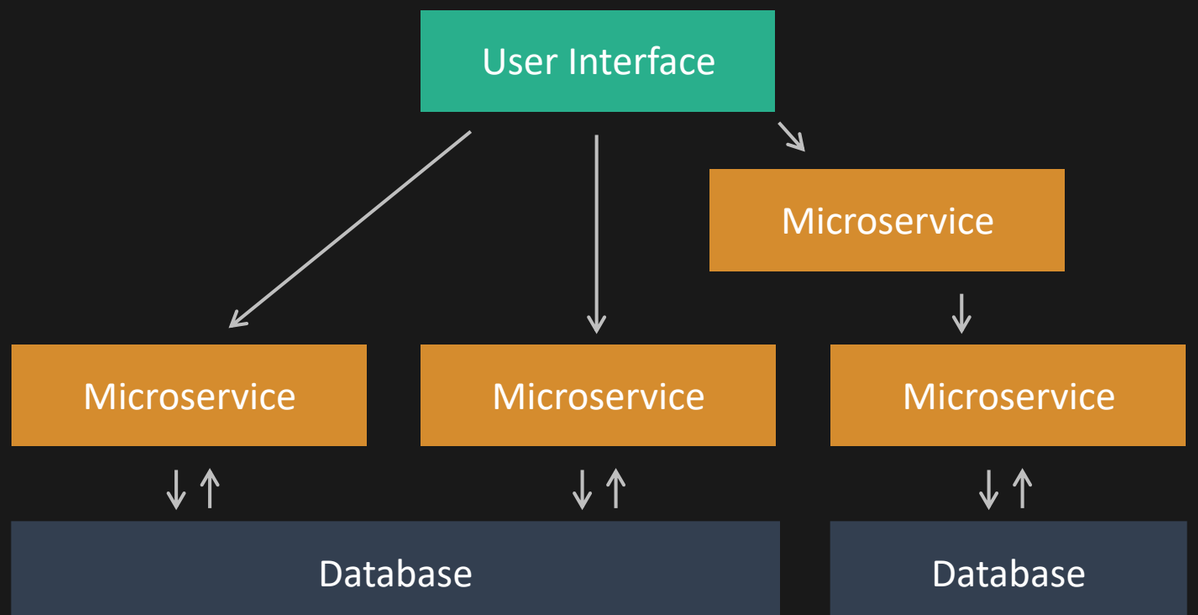
MICROSERVICE ARCHITECTURE

STRENGTHS

- Independent components
- Easier understanding
- Better scalability
- Load-balancing
- Flexibility with technology
- The higher level of agility

WEAKNESSES

- Extra complexity
- System distribution
- Testing
- Cross-cutting concerns:
  - Configurations,
  - Logging,
  - Metrics,
  - Health checks,
  - Security, and
  - Others.



Source: Microservices vs Monolith: which architecture is the best choice for your business? by Romana Gnatyk, 03 October 2018.

TCP

THE PROTOCOL

WHAT IS TCP/IP?

TCP/IP stands for **Transmission Control Protocol / Internet Protocol**.

- TCP/IP is a set of standardized rules that allow computers to communicate on a network such as the Internet.
- TCP and IP are two separate network protocols. **IP** is the part that obtains the address to which data is sent. **TCP** is responsible for data delivery once that **IP address** has been found.
- TCP is a text- and binary-based protocol, and custom or standard protocol built on top of it, such as HTTP, SMTP, FTP, SSH, etc.
- IP is machine-to-machine. TCP is process-to-process. Each process listens to a **socket**. Each socket has an assigned **Port number**.
- TCP provides reliable, ordered, and error-checked delivery of a stream of bytes between applications running on hosts communicating via an IP network.

WHAT IS TCP/IP?

TCP/IP stands for **Transmission Control Protocol / Internet Protocol**.

- The **loopback** (127.0.0.1) or **localhost** is a hostname that refers to the current computer used to access it. It is used to access the network services that are running on the host via the loopback network interface.
- The **Domain Name System (DNS)** is a hierarchical and decentralized naming system for computers, services, or other resources connected to the Internet or a private network. It maps numeric IP addresses to human-readable domain names or hostnames of specific computers.
- There are private and public IPs. Private IPs are only accessible within the network and public IPs are accessible to everyone on the Internet.
- A router maps public IPs to private IPs via **Network address translation (NAT)**, a method of mapping an IP address space into another by modifying network address information in the IP header of packets while they are in transit across a traffic routing device.

PRIVATE IPs

Reserved private IPv4 network ranges:

Name	CIDR Block	Address Range	Number of Addresses
24-bit block	10.0.0.0/8	10.0.0.0 – 10.255.255.255	16,777,216
20-bit block	172.16.0.0/12	172.16.0.0 – 172.31.255.255	1,048,576
16-bit block	192.168.0.0/16	192.168.0.0 – 192.168.255.255	65,536

TELNET

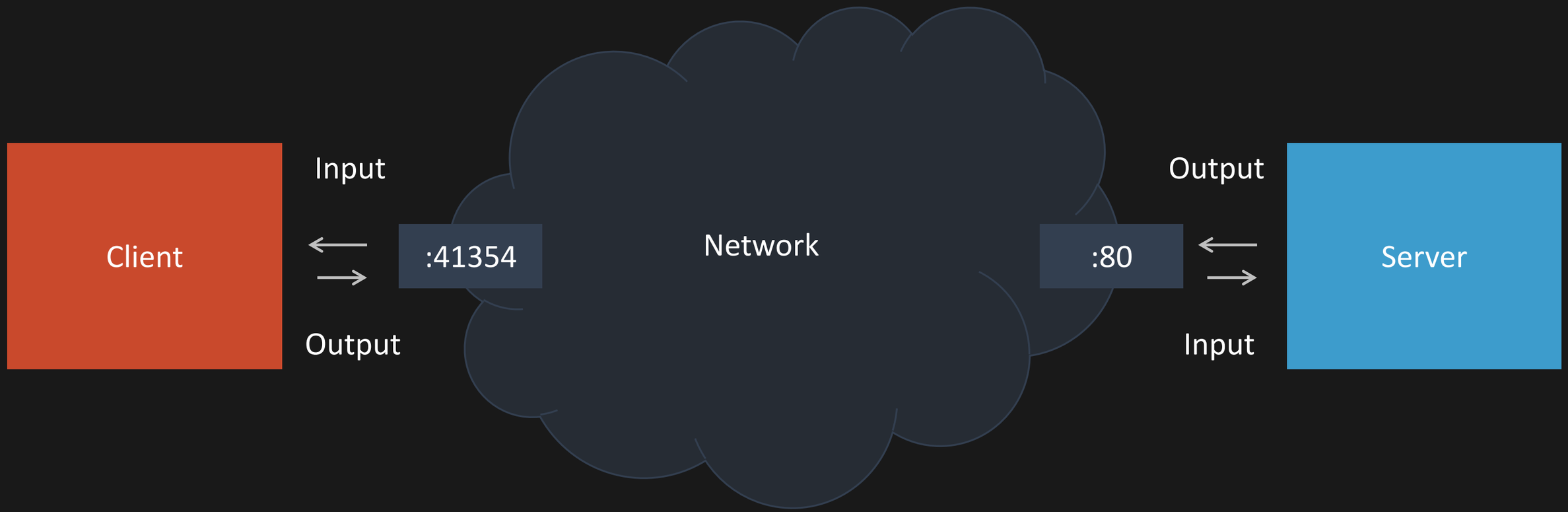
```
$ telnet hostname port
```

- Telnet is a client-server protocol based on text-oriented data exchange over TCP connections.
- Telnet enables remote communication with a TCP server via text-based inputs and outputs.

TCP

SERVICES

SOCKETS



TCP CLIENT

```java
/** Usage: java TCPClient <host> <port> */
public class TCPClient {
  private static PrintStream Log = System.out;
  private static void validateArgs(String[] args) { ... }
  public static void main(String[] args) throws Exception {
    validateArgs(args);

    Socket client = null;
    try {
      client          = new Socket(args[0], Integer.parseInt(args[1])); // Socket(host, port)
      PrintStream req = new PrintStream(client.getOutputStream(), true);
      Scanner res     = new Scanner(client.getInputStream());
      Scanner in      = new Scanner(System.in);

      Log.printf("Connected to server %s:%d\n", client.getInetAddress(), client.getPort());
      Log.print("Enter your request, then press <Enter>: ");
      String request = in.nextLine();
      req.println(request);

      String response = res.nextLine();
      Log.print("The response is: ");
      Log.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      try {
        client.close();
      } catch (Exception e) {
        Log.println("Failed to close client connection: " + e.getMessage());
      }
      Log.println("Client connection closed.");
    }
  }
}
```

TCP CLIENT

- Specify the server's host IP address and TCP socket port number that it is listening to.

```java
/** Usage: java TCPClient <host> <port> */
public class TCPClient {
  private static PrintStream Log = System.out;
  private static void validateArgs(String[] args) { ... }
  public static void main(String[] args) throws Exception {
    validateArgs(args);

    Socket client = null;
    try {
      client          = new Socket(args[0], Integer.parseInt(args[1])); // Socket(host, port)
      PrintStream req = new PrintStream(client.getOutputStream(), true);
      Scanner res     = new Scanner(client.getInputStream());
      Scanner in      = new Scanner(System.in);

      Log.printf("Connected to server %s:%d\n", client.getInetAddress(), client.getPort());
      Log.print("Enter your request, then press <Enter>: ");
      String request = in.nextLine();
      req.println(request);

      String response = res.nextLine();
      Log.print("The response is: ");
      Log.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      try {
        client.close();
      } catch (Exception e) {
        Log.println("Failed to close client connection: " + e.getMessage());
      }
      Log.println("Client connection closed.");
    }
  }
}
```

TCP CLIENT

- Specify the server's host IP address and TCP socket port number that it is listening to.
- Get an output stream to issue the request to the server.

```java
/** Usage: java TCPClient <host> <port> */
public class TCPClient {
  private static PrintStream Log = System.out;
  private static void validateArgs(String[] args) { ... }
  public static void main(String[] args) throws Exception {
    validateArgs(args);

    Socket client = null;
    try {
      client          = new Socket(args[0], Integer.parseInt(args[1])); // Socket(host, port)
      PrintStream req = new PrintStream(client.getOutputStream(), true);
      Scanner res     = new Scanner(client.getInputStream());
      Scanner in      = new Scanner(System.in);

      Log.printf("Connected to server %s:%d\n", client.getInetAddress(), client.getPort());
      Log.print("Enter your request, then press <Enter>: ");
      String request = in.nextLine();
      req.println(request);

      String response = res.nextLine();
      Log.print("The response is: ");
      Log.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      try {
        client.close();
      } catch (Exception e) {
        Log.println("Failed to close client connection: " + e.getMessage());
      }
      Log.println("Client connection closed.");
    }
  }
}
```

TCP CLIENT

- Specify the server's host IP address and TCP socket port number that it is listening to.
- Get an output stream to issue the request to the server.
- Get an input stream to read the response from the server.

```java
/** Usage: java TCPClient <host> <port> */
public class TCPClient {
  private static PrintStream Log = System.out;
  private static void validateArgs(String[] args) { ... }
  public static void main(String[] args) throws Exception {
    validateArgs(args);

    Socket client = null;
    try {
      client          = new Socket(args[0], Integer.parseInt(args[1])); // Socket(host, port)
      PrintStream req = new PrintStream(client.getOutputStream(), true);
      Scanner res     = new Scanner(client.getInputStream());
      Scanner in      = new Scanner(System.in);

      Log.printf("Connected to server %s:%d\n", client.getInetAddress(), client.getPort());
      Log.print("Enter your request, then press <Enter>: ");
      String request = in.nextLine();
      req.println(request);

      String response = res.nextLine();
      Log.print("The response is: ");
      Log.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      try {
        client.close();
      } catch (Exception e) {
        Log.println("Failed to close client connection: " + e.getMessage());
      }
      Log.println("Client connection closed.");
    }
  }
}
```

TCP CLIENT

- Specify the server's host IP address and TCP socket port number that it is listening to.
- Get an output stream to issue the request to the server.
- Get an input stream to read the response from the server.
- Close the connection with the server.

```java
/** Usage: java TCPClient <host> <port> */
public class TCPClient {
  private static PrintStream Log = System.out;
  private static void validateArgs(String[] args) { ... }
  public static void main(String[] args) throws Exception {
    validateArgs(args);

    Socket client = null;
    try {
      client          = new Socket(args[0], Integer.parseInt(args[1])); // Socket(host, port)
      PrintStream req = new PrintStream(client.getOutputStream(), true);
      Scanner res     = new Scanner(client.getInputStream());
      Scanner in      = new Scanner(System.in);

      Log.printf("Connected to server %s:%d\n", client.getInetAddress(), client.getPort());
      Log.print("Enter your request, then press <Enter>: ");
      String request = in.nextLine();
      req.println(request);

      String response = res.nextLine();
      Log.print("The response is: ");
      Log.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      try {
        client.close();
      } catch (Exception e) {
        Log.println("Failed to close client connection: " + e.getMessage());
      }
      Log.println("Client connection closed.");
    }
  }
}
```

TCP CLIENT

- Specify the server's host IP address and TCP socket port number that it is listening to.
- Get an output stream to issue the request to the server.
- Get an input stream to read the response from the server.
- Close the connection with the server.
- Or use a try-with-resources block to automatically close the connection and the input and output streams.

```java
/** Usage: java TCPClient <host> <port> */
public class TCPClient {
  private static PrintStream Log = System.out;
  private static void validateArgs(String[] args) { ... }
  public static void main(String[] args) throws Exception {
    validateArgs(args);

    try (
      Socket client   = new Socket(args[0], Integer.parseInt(args[1])); // Socket(host, port)
      PrintStream req = new PrintStream(client.getOutputStream(), true);
      Scanner res     = new Scanner(client.getInputStream());
      Scanner in      = new Scanner(System.in);
    ) {
      Log.printf("Connected to server %s:%d\n", client.getInetAddress(), client.getPort());
      Log.print("Enter your request, then press <Enter>: ");
      String request = in.nextLine();
      req.println(request);

      String response = res.nextLine();
      Log.print("The response is: ");
      Log.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      Log.println("Client connection closed.");
    }
  }
}
```

TCP SERVER

```java
/** Usage: java TCPServer */
public class TCPServer extends Thread {
  private static PrintStream Log = System.out;
  private Socket client;
  private TCPServer(Socket client) {
    this.client = client;
  }
  public static void main(String[] args) throws Exception {
    int port = 0;
    InetAddress host = InetAddress.getLocalHost(); // .getLoopbackAddress();
    try (ServerSocket server = new ServerSocket(port, 0, host)) {
      Log.printf("Server listening on %s:%d\n", server.getInetAddress(), server.getLocalPort(
      while (true) {
        Socket client = server.accept();
        (new TCPServer(client)).start();
      }
    }
  }
  public void run() {
    // ...
  }
}
```

TCP SERVER

- A server must be able to accept multiple clients at the same time. To support concurrency, we extend the `Thread` class. There are two ways to create a new thread of execution. This is the first.

```java
/** Usage: java TCPServer */
public class TCPServer extends Thread {
  private static PrintStream Log = System.out;
  private Socket client;
  private TCPServer(Socket client) {
    this.client = client;
  }
  public static void main(String[] args) throws Exception {
    int port = 0;
    InetAddress host = InetAddress.getLocalHost(); // .getLoopbackAddress();
    try (ServerSocket server = new ServerSocket(port, 0, host)) {
      Log.printf("Server listening on %s:%d\n", server.getInetAddress(), server.getLocalPort(
      while (true) {
        Socket client = server.accept();
        (new TCPServer(client)).start();
      }
    }
  }
  public void run() {
    // ...
  }
}
```

TCP SERVER

- A server must be able to accept multiple clients at the same time. To support concurrency, we extend the `Thread` class. There are two ways to create a new thread of execution. This is the first.
- To start the thread running, we call the `start` method which invokes the `run` method. We override the `run` method with our own implementation. More later.

```java
/** Usage: java TCPServer */
public class TCPServer extends Thread {
  private static PrintStream Log = System.out;
  private Socket client;
  private TCPServer(Socket client) {
    this.client = client;
  }
  public static void main(String[] args) throws Exception {
    int port = 0;
    InetAddress host = InetAddress.getLocalHost(); // .getLoopbackAddress();
    try (ServerSocket server = new ServerSocket(port, 0, host)) {
      Log.printf("Server listening on %s:%d\n", server.getInetAddress(), server.getLocalPort(
      while (true) {
        Socket client = server.accept();
        (new TCPServer(client)).start();
      }
    }
  }
  public void run() {
    // ...
  }
}
```

TCP SERVER

- A server must be able to accept multiple clients at the same time. To support concurrency, we extend the `Thread` class. There are two ways to create a new thread of execution. This is the first.
- To start the thread running, we call the `start` method which invokes the `run` method. We override the `run` method with our own implementation. More later.
- Create a server instance on a host and port. We will use the local host's own IP address. Alternatively, we could use the loopback address. However, this would restrict connections to the server to clients on the same host. See next slide for detail about `ServerSocket` constructor.

```
1  /** Usage: java TCPServer */
2  public class TCPServer extends Thread {
3    private static PrintStream Log = System.out;
4    private Socket client;
5    private TCPServer(Socket client) {
6      this.client = client;
7    }
8    public static void main(String[] args) throws Exception {
9      int port = 0;
10     InetAddress host = InetAddress.getLocalHost(); // .getLoopbackAddress();
11     try (ServerSocket server = new ServerSocket(port, 0, host)) {
12       Log.printf("Server listening on %s:%d\n", server.getInetAddress(), server.getLocalPort(
13       while (true) {
14         Socket client = server.accept();
15         (new TCPServer(client)).start();
16       }
17     }
18   }
19   public void run() {
20     // ...
21   }
22 }
```

TCP SERVER

- A server must be able to accept multiple clients at the same time. To support concurrency, we extend the `Thread` class. There are two ways to create a new thread of execution. This is the first.
- To start the thread running, we call the `start` method which invokes the `run` method. We override the `run` method with our own implementation. More later.
- Create a server instance on a host and port. We will use the local host's own IP address. Alternatively, we could use the loopback address. However, this would restrict connections to the server to clients on the same host. See next slide for detail about `ServerSocket` constructor.
- Query the server's host and port for logging.

```java
/** Usage: java TCPServer */
public class TCPServer extends Thread {
  private static PrintStream Log = System.out;
  private Socket client;
  private TCPServer(Socket client) {
    this.client = client;
  }
  public static void main(String[] args) throws Exception {
    int port = 0;
    InetAddress host = InetAddress.getLocalHost(); // .getLoopbackAddress();
    try (ServerSocket server = new ServerSocket(port, 0, host)) {
      Log.printf("Server listening on %s:%d\n", server.getInetAddress(), server.getLocalPort(
      while (true) {
        Socket client = server.accept();
        (new TCPServer(client)).start();
      }
    }
  }
  public void run() {
    // ...
  }
}
```

TCP SERVER

- A server must be able to accept multiple clients at the same time. To support concurrency, we extend the `Thread` class. There are two ways to create a new thread of execution. This is the first.
- To start the thread running, we call the `start` method which invokes the `run` method. We override the `run` method with our own implementation. More later.
- Create a server instance on a host and port. We will use the local host's own IP address. Alternatively, we could use the loopback address. However, this would restrict connections to the server to clients on the same host. See next slide for detail about `ServerSocket` constructor.
- Query the server's host and port for logging.
- Accept the next client connection. Blocks until a request arrives. Then, create a new thread and starts it running. Loops until the server is shut down.

```java
/** Usage: java TCPServer */
public class TCPServer extends Thread {
  private static PrintStream Log = System.out;
  private Socket client;
  private TCPServer(Socket client) {
    this.client = client;
  }
  public static void main(String[] args) throws Exception {
    int port = 0;
    InetAddress host = InetAddress.getLocalHost(); // .getLoopbackAddress();
    try (ServerSocket server = new ServerSocket(port, 0, host)) {
      Log.printf("Server listening on %s:%d\n", server.getInetAddress(), server.getLocalPort(
      while (true) {
        Socket client = server.accept();
        (new TCPServer(client)).start();
      }
    }
  }
  public void run() {
    // ...
  }
}
```

SERVERSOCKET

Create a server with the specified port, listen backlog, and local IP address to bind to.

```
public ServerSocket(int port, int backlog, InetAddress bindAddr) throws IOException;
```

port Must be between `0` and `65535` (16-bits), inclusive. A port number of `0` means that the port number is automatically allocated.

backlog Is the requested maximum number of pending connections on the socket. The value provided should be greater than `0`. If it is less than or equal to `0`, then an implementation-specific default will be used.

bindAddr Can be used on a multi-homed host for a `ServerSocket` that will only accept connect requests to one of its addresses. If `bindAddr` is null, it will default accepting connections on any/all local addresses.

TCP SERVER `run`

In microservices, `run` would typically: compute, use APIs, lookup a database, use HTTP, etc.

```java
/** Usage: java TCPServer */
public class TCPServer extends Thread {
  // ...
  public static void main(String[] args) throws Exception {
    int port = 0;
    InetAddress host = InetAddress.getLocalHost(); // .getLoopbackAddress();
    try (ServerSocket server = new ServerSocket(port, 0, host)) {
      Log.printf("Server listening on %s:%d\n", server.getInetAddress(), server.getLocalPort());
      while (true) {
        (new TCPServer(server.accept())).start();
      }
    }
  }
  public void run() {
    Log.printf("Connected to %s:%d\n", client.getInetAddress(), client.getPort());

    try (
      Socket client   = this.client; // Ensures client connection will be closed by try-statement.
      Scanner req     = new Scanner(client.getInputStream());
      PrintStream res = new PrintStream(client.getOutputStream(), true);
    ) {
      String response;
      String request = req.nextLine();

      if (validateRequest(request)) {
        response = handleRequest(request);
      } else {
        response = "Don't understand: " + request;
      }
      res.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      Log.printf("Disconnected from %s:%d\n", client.getInetAddress(), client.getPort());
    }
  }
}
```

TCP SERVER `run`

In microservices, `run` would typically: compute, use APIs, lookup a database, use HTTP, etc.

- Query the client socket:
  - `client.getPort`
  - `client.getInetAddress`
  - `client.getInputStream`
  - `client.getOutputStream`

```java
/** Usage: java TCPServer */
public class TCPServer extends Thread {
  // ...
  public static void main(String[] args) throws Exception {
    int port = 0;
    InetAddress host = InetAddress.getLocalHost(); // .getLoopbackAddress();
    try (ServerSocket server = new ServerSocket(port, 0, host)) {
      Log.printf("Server listening on %s:%d\n", server.getInetAddress(), server.getLocalPort());
      while (true) {
        (new TCPServer(server.accept())).start();
      }
    }
  }
  public void run() {
    Log.printf("Connected to %s:%d\n", client.getInetAddress(), client.getPort());

    try (
      Socket client   = this.client; // Ensures client connection will be closed by try-statement.
      Scanner req     = new Scanner(client.getInputStream());
      PrintStream res = new PrintStream(client.getOutputStream(), true);
    ) {
      String response;
      String request = req.nextLine();

      if (validateRequest(request)) {
        response = handleRequest(request);
      } else {
        response = "Don't understand: " + request;
      }
      res.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      Log.printf("Disconnected from %s:%d\n", client.getInetAddress(), client.getPort());
    }
  }
}
```

TCP SERVER `run`

In microservices, `run` would typically: compute, use APIs, lookup a database, use HTTP, etc.

- Query the client socket:
  - `client.getPort`
  - `client.getInetAddress`
  - `client.getInputStream`
  - `client.getOutputStream`
- Get an input stream to read the request from the client.

```
1  /** Usage: java TCPServer */
2  public class TCPServer extends Thread {
3    // ...
4    public static void main(String[] args) throws Exception {
5      int port = 0;
6      InetAddress host = InetAddress.getLocalHost(); // .getLoopbackAddress();
7      try (ServerSocket server = new ServerSocket(port, 0, host)) {
8        Log.printf("Server listening on %s:%d\n", server.getInetAddress(), server.getLocalPort());
9        while (true) {
10         (new TCPServer(server.accept())).start();
11       }
12     }
13   }
14   public void run() {
15     Log.printf("Connected to %s:%d\n", client.getInetAddress(), client.getPort());
16
17     try (
18       Socket client   = this.client; // Ensures client connection will be closed by try-statement.
19       Scanner req     = new Scanner(client.getInputStream());
20       PrintStream res = new PrintStream(client.getOutputStream(), true);
21     ) {
22       String response;
23       String request = req.nextLine();
24
25       if (validateRequest(request)) {
26         response = handleRequest(request);
27       } else {
28         response = "Don't understand: " + request;
29       }
30       res.println(response);
31     } catch (Exception e) {
32       Log.println(e);
33     } finally {
34       Log.printf("Disconnected from %s:%d\n", client.getInetAddress(), client.getPort());
35     }
36   }
37 }
```

TCP SERVER `run`

In microservices, `run` would typically: compute, use APIs, lookup a database, use HTTP, etc.

- Query the client socket:
  - `client.getPort`
  - `client.getInetAddress`
  - `client.getInputStream`
  - `client.getOutputStream`
- Get an input stream to read the request from the client.
- Validate the request, process it and form a respond.

```java
/** Usage: java TCPServer */
public class TCPServer extends Thread {
  // ...
  public static void main(String[] args) throws Exception {
    int port = 0;
    InetAddress host = InetAddress.getLocalHost(); // .getLoopbackAddress();
    try (ServerSocket server = new ServerSocket(port, 0, host)) {
      Log.printf("Server listening on %s:%d\n", server.getInetAddress(), server.getLocalPort());
      while (true) {
        (new TCPServer(server.accept())).start();
      }
    }
  }
  public void run() {
    Log.printf("Connected to %s:%d\n", client.getInetAddress(), client.getPort());

    try (
      Socket client   = this.client; // Ensures client connection will be closed by try-statement.
      Scanner req     = new Scanner(client.getInputStream());
      PrintStream res = new PrintStream(client.getOutputStream(), true);
    ) {
      String response;
      String request = req.nextLine();

      if (validateRequest(request)) {
        response = handleRequest(request);
      } else {
        response = "Don't understand: " + request;
      }
      res.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      Log.printf("Disconnected from %s:%d\n", client.getInetAddress(), client.getPort());
    }
  }
}
```

TCP SERVER `run`

In microservices, `run` would typically: compute, use APIs, lookup a database, use HTTP, etc.

- Query the client socket:
  - `client.getPort`
  - `client.getInetAddress`
  - `client.getInputStream`
  - `client.getOutputStream`
- Get an input stream to read the request from the client.
- Validate the request, process it and form a respond.
- Send the respond via an output stream to the client.

```java
/** Usage: java TCPServer */
public class TCPServer extends Thread {
  // ...
  public static void main(String[] args) throws Exception {
    int port = 0;
    InetAddress host = InetAddress.getLocalHost(); // .getLoopbackAddress();
    try (ServerSocket server = new ServerSocket(port, 0, host)) {
      Log.printf("Server listening on %s:%d\n", server.getInetAddress(), server.getLocalPort());
      while (true) {
        (new TCPServer(server.accept())).start();
      }
    }
  }
  public void run() {
    Log.printf("Connected to %s:%d\n", client.getInetAddress(), client.getPort());

    try (
      Socket client   = this.client; // Ensures client connection will be closed by try-statement.
      Scanner req     = new Scanner(client.getInputStream());
      PrintStream res = new PrintStream(client.getOutputStream(), true);
    ) {
      String response;
      String request = req.nextLine();

      if (validateRequest(request)) {
        response = handleRequest(request);
      } else {
        response = "Don't understand: " + request;
      }
      res.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      Log.printf("Disconnected from %s:%d\n", client.getInetAddress(), client.getPort());
    }
  }
}
```

TCP SERVER `run`

In microservices, `run` would typically: compute, use APIs, lookup a database, use HTTP, etc.

- Query the client socket:
  - `client.getPort`
  - `client.getInetAddress`
  - `client.getInputStream`
  - `client.getOutputStream`
- Get an input stream to read the request from the client.
- Validate the request, process it and form a respond.
- Send the respond via an output stream to the client.
- Close the connection with the client. or let Java automatically close it along the input and output streams at the end of the try-with-statement.

```java
/** Usage: java TCPServer */
public class TCPServer extends Thread {
  // ...
  public static void main(String[] args) throws Exception {
    int port = 0;
    InetAddress host = InetAddress.getLocalHost(); // .getLoopbackAddress();
    try (ServerSocket server = new ServerSocket(port, 0, host)) {
      Log.printf("Server listening on %s:%d\n", server.getInetAddress(), server.getLocalPort());
      while (true) {
        (new TCPServer(server.accept())).start();
      }
    }
  }
  public void run() {
    Log.printf("Connected to %s:%d\n", client.getInetAddress(), client.getPort());

    try (
      Socket client   = this.client; // Ensures client connection will be closed by try-statement.
      Scanner req     = new Scanner(client.getInputStream());
      PrintStream res = new PrintStream(client.getOutputStream(), true);
    ) {
      String response;
      String request = req.nextLine();

      if (validateRequest(request)) {
        response = handleRequest(request);
      } else {
        response = "Don't understand: " + request;
      }
      res.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      Log.printf("Disconnected from %s:%d\n", client.getInetAddress(), client.getPort());
    }
  }
}
```

TCP CLIENT & SERVER

```java
/** Usage: java TCPClient <host> <port> */
public class TCPClient {
  private static PrintStream Log = System.out;
  private static void validateArgs(String[] args) { ... }
  public static void main(String[] args) throws Exception {
    validateArgs(args);

    try (
      Socket client   = new Socket(args[0], Integer.parseInt(args[1])); // Socket(host, por
      PrintStream req = new PrintStream(client.getOutputStream(), true);
      Scanner res     = new Scanner(client.getInputStream());
      Scanner in      = new Scanner(System.in);
    ) {
      Log.printf("Connected to server %s:%d\n", client.getInetAddress(), client.getPort());
      Log.print("Enter your request, then press <Enter>: ");
      String request = in.nextLine();
      req.println(request);

      String response = res.nextLine();
      Log.print("The response is: ");
      Log.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      Log.println("Client connection closed.");
    }
  }
}
```

```java
/** Usage: java TCPServer */
public class TCPServer extends Thread {
  // ...
  public static void main(String[] args) throws Exception {
    int port = 0;
    InetAddress host = InetAddress.getLocalHost(); // .getLoopbackAddress();
    try (ServerSocket server = new ServerSocket(port, 0, host)) {
      Log.printf("Server listening on %s:%d\n", server.getInetAddress(), server.getLocalPor
      while (true) {
        (new TCPServer(server.accept())).start();
      }
    }
  }
  public void run() {
    Log.printf("Connected to %s:%d\n", client.getInetAddress(), client.getPort());

    try (
      Socket client   = this.client; // Ensures client connection will be closed by try-sta
      Scanner req     = new Scanner(client.getInputStream());
      PrintStream res = new PrintStream(client.getOutputStream(), true);
    ) {
      String response;
      String request = req.nextLine();

      if (validateRequest(request)) {
        response = handleRequest(request);
      } else {
        response = "Don't understand: " + request;
      }
      res.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      Log.printf("Disconnected from %s:%d\n", client.getInetAddress(), client.getPort());
    }
  }
}
```

TCP CLIENT & SERVER

```java
/** Usage: java TCPClient <host> <port> */
public class TCPClient {
  private static PrintStream Log = System.out;
  private static void validateArgs(String[] args) { ... }
  public static void main(String[] args) throws Exception {
    validateArgs(args);

    try (
      Socket client   = new Socket(args[0], Integer.parseInt(args[1])); // Socket(host, por
      PrintStream req = new PrintStream(client.getOutputStream(), true);
      Scanner res     = new Scanner(client.getInputStream());
      Scanner in      = new Scanner(System.in);
    ) {
      Log.printf("Connected to server %s:%d\n", client.getInetAddress(), client.getPort());
      Log.print("Enter your request, then press <Enter>: ");
      String request = in.nextLine();
      req.println(request);

      String response = res.nextLine();
      Log.print("The response is: ");
      Log.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      Log.println("Client connection closed.");
    }
  }
}
```

Server starts and listens for new connections.

```java
/** Usage: java TCPServer */
public class TCPServer extends Thread {
  // ...
  public static void main(String[] args) throws Exception {
    int port = 0;
    InetAddress host = InetAddress.getLocalHost(); // .getLoopbackAddress();
    try (ServerSocket server = new ServerSocket(port, 0, host)) {
      Log.printf("Server listening on %s:%d\n", server.getInetAddress(), server.getLocalPor
      while (true) {
        (new TCPServer(server.accept())).start();
      }
    }
  }
  public void run() {
    Log.printf("Connected to %s:%d\n", client.getInetAddress(), client.getPort());

    try (
      Socket client   = this.client; // Ensures client connection will be closed by try-sta
      Scanner req     = new Scanner(client.getInputStream());
      PrintStream res = new PrintStream(client.getOutputStream(), true);
    ) {
      String response;
      String request = req.nextLine();

      if (validateRequest(request)) {
        response = handleRequest(request);
      } else {
        response = "Don't understand: " + request;
      }
      res.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      Log.printf("Disconnected from %s:%d\n", client.getInetAddress(), client.getPort());
    }
  }
}
```

TCP CLIENT & SERVER

Client start connection to the server. » Server accepts the connection from the client.

```java
/** Usage: java TCPClient <host> <port> */
public class TCPClient {
  private static PrintStream Log = System.out;
  private static void validateArgs(String[] args) { ... }
  public static void main(String[] args) throws Exception {
    validateArgs(args);

    try (
      Socket client   = new Socket(args[0], Integer.parseInt(args[1])); // Socket(host, por
      PrintStream req = new PrintStream(client.getOutputStream(), true);
      Scanner res     = new Scanner(client.getInputStream());
      Scanner in      = new Scanner(System.in);
    ) {
      Log.printf("Connected to server %s:%d\n", client.getInetAddress(), client.getPort());
      Log.print("Enter your request, then press <Enter>: ");
      String request = in.nextLine();
      req.println(request);

      String response = res.nextLine();
      Log.print("The response is: ");
      Log.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      Log.println("Client connection closed.");
    }
  }
}
```

```java
/** Usage: java TCPServer */
public class TCPServer extends Thread {
  // ...
  public static void main(String[] args) throws Exception {
    int port = 0;
    InetAddress host = InetAddress.getLocalHost(); // .getLoopbackAddress();
    try (ServerSocket server = new ServerSocket(port, 0, host)) {
      Log.printf("Server listening on %s:%d\n", server.getInetAddress(), server.getLocalPor
      while (true) {
        (new TCPServer(server.accept())).start();
      }
    }
  }
  public void run() {
    Log.printf("Connected to %s:%d\n", client.getInetAddress(), client.getPort());

    try (
      Socket client   = this.client; // Ensures client connection will be closed by try-sta
      Scanner req     = new Scanner(client.getInputStream());
      PrintStream res = new PrintStream(client.getOutputStream(), true);
    ) {
      String response;
      String request = req.nextLine();

      if (validateRequest(request)) {
        response = handleRequest(request);
      } else {
        response = "Don't understand: " + request;
      }
      res.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      Log.printf("Disconnected from %s:%d\n", client.getInetAddress(), client.getPort());
    }
  }
}
```

TCP CLIENT & SERVER

Client forms a request and sends it to the server. » Server reads the request from the client.

```java
/** Usage: java TCPClient <host> <port> */
public class TCPClient {
  private static PrintStream Log = System.out;
  private static void validateArgs(String[] args) { ... }
  public static void main(String[] args) throws Exception {
    validateArgs(args);

    try (
      Socket client   = new Socket(args[0], Integer.parseInt(args[1])); // Socket(host, por
      PrintStream req = new PrintStream(client.getOutputStream(), true);
      Scanner res     = new Scanner(client.getInputStream());
      Scanner in      = new Scanner(System.in);
    ) {
      Log.printf("Connected to server %s:%d\n", client.getInetAddress(), client.getPort());
      Log.print("Enter your request, then press <Enter>: ");
      String request = in.nextLine();
      req.println(request);

      String response = res.nextLine();
      Log.print("The response is: ");
      Log.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      Log.println("Client connection closed.");
    }
  }
}
```

```java
/** Usage: java TCPServer */
public class TCPServer extends Thread {
  // ...
  public static void main(String[] args) throws Exception {
    int port = 0;
    InetAddress host = InetAddress.getLocalHost(); // .getLoopbackAddress();
    try (ServerSocket server = new ServerSocket(port, 0, host)) {
      Log.printf("Server listening on %s:%d\n", server.getInetAddress(), server.getLocalPor
      while (true) {
        (new TCPServer(server.accept())).start();
      }
    }
  }
  public void run() {
    Log.printf("Connected to %s:%d\n", client.getInetAddress(), client.getPort());

    try (
      Socket client   = this.client; // Ensures client connection will be closed by try-sta
      Scanner req     = new Scanner(client.getInputStream());
      PrintStream res = new PrintStream(client.getOutputStream(), true);
    ) {
      String response;
      String request = req.nextLine();

      if (validateRequest(request)) {
        response = handleRequest(request);
      } else {
        response = "Don't understand: " + request;
      }
      res.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      Log.printf("Disconnected from %s:%d\n", client.getInetAddress(), client.getPort());
    }
  }
}
```

TCP CLIENT & SERVER

```java
/** Usage: java TCPClient <host> <port> */
public class TCPClient {
  private static PrintStream Log = System.out;
  private static void validateArgs(String[] args) { ... }
  public static void main(String[] args) throws Exception {
    validateArgs(args);

    try (
      Socket client   = new Socket(args[0], Integer.parseInt(args[1])); // Socket(host, por
      PrintStream req = new PrintStream(client.getOutputStream(), true);
      Scanner res     = new Scanner(client.getInputStream());
      Scanner in      = new Scanner(System.in);
    ) {
      Log.printf("Connected to server %s:%d\n", client.getInetAddress(), client.getPort());
      Log.print("Enter your request, then press <Enter>: ");
      String request = in.nextLine();
      req.println(request);

      String response = res.nextLine();
      Log.print("The response is: ");
      Log.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      Log.println("Client connection closed.");
    }
  }
}
```

Server validates and processes request and forms a response.

```java
/** Usage: java TCPServer */
public class TCPServer extends Thread {
  // ...
  public static void main(String[] args) throws Exception {
    int port = 0;
    InetAddress host = InetAddress.getLocalHost(); // .getLoopbackAddress();
    try (ServerSocket server = new ServerSocket(port, 0, host)) {
      Log.printf("Server listening on %s:%d\n", server.getInetAddress(), server.getLocalPor
      while (true) {
        (new TCPServer(server.accept())).start();
      }
    }
  }
  public void run() {
    Log.printf("Connected to %s:%d\n", client.getInetAddress(), client.getPort());

    try (
      Socket client   = this.client; // Ensures client connection will be closed by try-sta
      Scanner req     = new Scanner(client.getInputStream());
      PrintStream res = new PrintStream(client.getOutputStream(), true);
    ) {
      String response;
      String request = req.nextLine();

      if (validateRequest(request)) {
        response = handleRequest(request);
      } else {
        response = "Don't understand: " + request;
      }
      res.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      Log.printf("Disconnected from %s:%d\n", client.getInetAddress(), client.getPort());
    }
  }
}
```

TCP CLIENT & SERVER

Client receives the response from the server. «« Server sends the response to the client.

```java
/** Usage: java TCPClient <host> <port> */
public class TCPClient {
  private static PrintStream Log = System.out;
  private static void validateArgs(String[] args) { ... }
  public static void main(String[] args) throws Exception {
    validateArgs(args);

    try (
      Socket client   = new Socket(args[0], Integer.parseInt(args[1])); // Socket(host, por
      PrintStream req = new PrintStream(client.getOutputStream(), true);
      Scanner res     = new Scanner(client.getInputStream());
      Scanner in      = new Scanner(System.in);
    ) {
      Log.printf("Connected to server %s:%d\n", client.getInetAddress(), client.getPort());
      Log.print("Enter your request, then press <Enter>: ");
      String request = in.nextLine();
      req.println(request);

      String response = res.nextLine();
      Log.print("The response is: ");
      Log.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      Log.println("Client connection closed.");
    }
  }
}
```

```java
/** Usage: java TCPServer */
public class TCPServer extends Thread {
  // ...
  public static void main(String[] args) throws Exception {
    int port = 0;
    InetAddress host = InetAddress.getLocalHost(); // .getLoopbackAddress();
    try (ServerSocket server = new ServerSocket(port, 0, host)) {
      Log.printf("Server listening on %s:%d\n", server.getInetAddress(), server.getLocalPor
      while (true) {
        (new TCPServer(server.accept())).start();
      }
    }
  }
  public void run() {
    Log.printf("Connected to %s:%d\n", client.getInetAddress(), client.getPort());

    try (
      Socket client    = this.client; // Ensures client connection will be closed by try-sta
      Scanner req      = new Scanner(client.getInputStream());
      PrintStream res = new PrintStream(client.getOutputStream(), true);
    ) {
      String response;
      String request = req.nextLine();

      if (validateRequest(request)) {
        response = handleRequest(request);
      } else {
        response = "Don't understand: " + request;
      }
      res.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      Log.printf("Disconnected from %s:%d\n", client.getInetAddress(), client.getPort());
    }
  }
}
```

TCP CLIENT & SERVER

Client closes connection with the server.

```java
/** Usage: java TCPClient <host> <port> */
public class TCPClient {
  private static PrintStream Log = System.out;
  private static void validateArgs(String[] args) { ... }
  public static void main(String[] args) throws Exception {
    validateArgs(args);

    try (
      Socket client   = new Socket(args[0], Integer.parseInt(args[1])); // Socket(host, por
      PrintStream req = new PrintStream(client.getOutputStream(), true);
      Scanner res     = new Scanner(client.getInputStream());
      Scanner in      = new Scanner(System.in);
    ) {
      Log.printf("Connected to server %s:%d\n", client.getInetAddress(), client.getPort());
      Log.print("Enter your request, then press <Enter>: ");
      String request = in.nextLine();
      req.println(request);

      String response = res.nextLine();
      Log.print("The response is: ");
      Log.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      Log.println("Client connection closed.");
    }
  }
}
```

Server closes connection with the client.

```java
/** Usage: java TCPServer */
public class TCPServer extends Thread {
  // ...
  public static void main(String[] args) throws Exception {
    int port = 0;
    InetAddress host = InetAddress.getLocalHost(); // .getLoopbackAddress();
    try (ServerSocket server = new ServerSocket(port, 0, host)) {
      Log.printf("Server listening on %s:%d\n", server.getInetAddress(), server.getLocalPor
      while (true) {
        (new TCPServer(server.accept())).start();
      }
    }
  }
  public void run() {
    Log.printf("Connected to %s:%d\n", client.getInetAddress(), client.getPort());

    try (
      Socket client   = this.client; // Ensures client connection will be closed by try-sta
      Scanner req     = new Scanner(client.getInputStream());
      PrintStream res = new PrintStream(client.getOutputStream(), true);
    ) {
      String response;
      String request = req.nextLine();

      if (validateRequest(request)) {
        response = handleRequest(request);
      } else {
        response = "Don't understand: " + request;
      }
      res.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      Log.printf("Disconnected from %s:%d\n", client.getInetAddress(), client.getPort());
    }
  }
}
```

TCP CLIENT & SERVER

```java
/** Usage: java TCPClient <host> <port> */
public class TCPClient {
  private static PrintStream Log = System.out;
  private static void validateArgs(String[] args) { ... }
  public static void main(String[] args) throws Exception {
    validateArgs(args);

    try (
      Socket client   = new Socket(args[0], Integer.parseInt(args[1])); // Socket(host, por
      PrintStream req = new PrintStream(client.getOutputStream(), true);
      Scanner res     = new Scanner(client.getInputStream());
      Scanner in      = new Scanner(System.in);
    ) {
      Log.printf("Connected to server %s:%d\n", client.getInetAddress(), client.getPort());
      Log.print("Enter your request, then press <Enter>: ");
      String request = in.nextLine();
      req.println(request);

      String response = res.nextLine();
      Log.print("The response is: ");
      Log.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      Log.println("Client connection closed.");
    }
  }
}
```

Server continues accepting new connection from other clients.

```java
/** Usage: java TCPServer */
public class TCPServer extends Thread {
  // ...
  public static void main(String[] args) throws Exception {
    int port = 0;
    InetAddress host = InetAddress.getLocalHost(); // .getLoopbackAddress();
    try (ServerSocket server = new ServerSocket(port, 0, host)) {
      Log.printf("Server listening on %s:%d\n", server.getInetAddress(), server.getLocalPor
      while (true) {
        (new TCPServer(server.accept())).start();
      }
    }
  }
  public void run() {
    Log.printf("Connected to %s:%d\n", client.getInetAddress(), client.getPort());

    try (
      Socket client   = this.client; // Ensures client connection will be closed by try-sta
      Scanner req     = new Scanner(client.getInputStream());
      PrintStream res = new PrintStream(client.getOutputStream(), true);
    ) {
      String response;
      String request = req.nextLine();

      if (validateRequest(request)) {
        response = handleRequest(request);
      } else {
        response = "Don't understand: " + request;
      }
      res.println(response);
    } catch (Exception e) {
      Log.println(e);
    } finally {
      Log.printf("Disconnected from %s:%d\n", client.getInetAddress(), client.getPort());
    }
  }
}
```

This slide is intentionally left blank.

Return to Course Page.