# Notes for: Programming 3rd. Semester Datamatiker

**Pensum 2025 Java and C#:**

**Java:**

Netværksprogrammering > Det grundlæggende <http://www.docjava.dk/network/basic/basic.htm>

Netværksprogrammering > Flertrådede servere

<http://www.docjava.dk/network/multi_threaded_server/multi_threaded_server.htm>

**C# and .NET:**

XML > Sekventiel parsing af XML <http://www.fkj.dk/doccs/xml/reader_writer/reader_writer.htm>

"Implementing Security for Applications with Microsoft Visual Basic.Net and Microsoft Visual C#.Net", Tony Northrup, Microsoft Press, 2005.

— Lesson 2-4, s.22-48. <https://mega.nz/file/QEoAXTYS#fjwKeNS7PKKH2ePJ4fbEl91lxxtK9dLoISfo8V2e8cY>

"Pro ASP.NET Core 6, 9th ed.", Adam Freeman, Apress 2022”

— kapitel: 1-4, 6-11 (s.280n-287 er kursorisk), 38-39 (s.1225-1241 er ikke pensum)

<https://mega.nz/file/sFIEjLhK#bq9J1eHWLSBGVIyZEC_EygJPgX61UGjMn3oXcG5f-kk>

Stub-Skeleton Pattern: <https://en.wikipedia.org/wiki/Distributed_object_communication>

"Netværk": <http://www.fkj.dk/datamatiker/network/network.html>

(de mange videoer der refereres til er ikke pensum)

(MVC og Razor Pages er pensum. Blazor er ikke pensum)

"Bootstrap Beginner Crash Course", Traversy Media, YouTube. (kursorisk)  
<https://www.youtube.com/watch?v=5GcQtLDGXy8>

Større program-eksempler (se fkj.dk under Public Files): <https://eamvdk-my.sharepoint.com/personal/fkj_eamv_dk/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Ffkj%5Feamv%5Fdk%2FDocuments%2FPublic%20Files%2F3%2Esem&ga=1>

2024-08-22 - MultiThreaded Server (C#).zip  
2024-08-29 - Public Key Encryptor (solved).zip  
2024-08-30 - Hash (solved).zip  
2024-09-03 - Stub-Skeleton Pattern (opgave A).zip  
2024-09-06 - Stub-Skeleton Pattern (opgave B).zip  
2024-09-09 - Stub-Skeleton Pattern (opgave C+D).zip

Andet (se ligeledes fkj.dk under Public Files):  
  
Netværkskode (C#).txt

Contents

[Notes for: Programming 3rd. Semester Datamatiker 1](#_Toc190700388)

[Notes: 4](#_Toc190700389)

[Netværksprogrammering > Det grundlæggende 4](#_Toc190700390)

[Connection-Based (TCP - Transmission Control Protocol) 4](#_Toc190700391)

[Connectionless (UDP - User Datagram Protocol) 4](#_Toc190700392)

[What is an IP Address? 4](#_Toc190700393)

[Domain Name System (DNS) 4](#_Toc190700394)

[InputStream vs OutputStream: 5](#_Toc190700395)

[UDP – User Datagram Protocol 5](#_Toc190700396)

[Netværksprogrammering > Flertrådede servere 5](#_Toc190700397)

[Multi-Threaded Server Architecture 5](#_Toc190700398)

[Ports 6](#_Toc190700399)

[How does routers work? 7](#_Toc190700400)

[NAT (Network Address Translation) 7](#_Toc190700401)

[How does NAT work? 8](#_Toc190700402)

[What is a Network Packet? 9](#_Toc190700403)

[How is Port Forwarding Different from NAT? 10](#_Toc190700404)

[XML > Sekventiel parsing af XML 10](#_Toc190700405)

[Implementing Security for Applications 10](#_Toc190700406)

[Pro ASP.NET Core 12](#_Toc190700407)

[Stub-Skeleton Pattern 12](#_Toc190700408)

[Netværk 12](#_Toc190700409)

[MVC og Razor Pages 12](#_Toc190700410)

[Bootstrap Beginner Crash Course 12](#_Toc190700411)

[Større program-eksempler 12](#_Toc190700412)

[Netværkskode (C#) 12](#_Toc190700413)

[Andet 12](#_Toc190700414)

## Notes:

## Netværksprogrammering > Det grundlæggende

There are **two primary types of network communication**:

### Connection-Based (TCP - Transmission Control Protocol)

* A **continuous stream** of data is exchanged between two applications.
* A **virtual connection** is maintained for reliable data transfer.
* Examples: **Web browsing, email, file transfers.**
* **Pros**: Reliable, ordered delivery.
* **Cons**: More overhead, slower.

### Connectionless (UDP - User Datagram Protocol)

* Data is sent in **discrete packets** like a postal system.
* No persistent connection is maintained.
* Examples: **Online gaming, video streaming, VoIP.**
* **Pros**: Faster, lower overhead.
* **Cons**: Unreliable (packets may be lost or arrive out of order).

### What is an IP Address?

* An IP address (Internet Protocol address) uniquely identifies a device on a network.
* **IPv4 (Internet Protocol version 4)** uses **32-bit addresses** (e.g., 192.168.1.1).
* **IPv6** uses **128-bit addresses** to support more devices (2001:db8::ff00:42:8329).
* Used to route data across networks.

### Domain Name System (DNS)

* DNS **translates human-readable domain names** (e.g., www.google.com) **into IP addresses** (172.217.11.174).
* Instead of remembering IPs, you can use domain names.

DNS Resolution Process:

* User enters [www.example.com](http://www.example.com)
* Browser asks a DNS server for the **IP address**.
* DNS server returns 93.184.216.34.
* Browser connects to the server using that IP.

#### Client-Server Interaction:

* **Client:** Initiates a request (e.g., a browser requesting a webpage).
* **Server**: Listens on a port and **responds** to requests (e.g., a web server returning HTML content).

### InputStream vs OutputStream:

* **InputStream**: Used to **read data** (e.g., reading from a socket, file, or keyboard input).
* **OutputStream**: Used to **write data** (e.g., sending data to a server, saving to a file).

**If a Client Sends a Message, is it InputStream or OutputStream?**

* **On the Client Side:** Sending a message = **OutputStream** (writing data).
* **On the Server Side**: Receiving a message = **InputStream** (reading data).

### UDP – User Datagram Protocol

Unlike TCP, UDP is **connectionless** and **does not guarantee delivery**.

**How UDP works:**

* **No handshake** – packets are just sent.
* If packets **arrive out of order or get lost**, they **aren't automatically resent**.
* **Used in real-time applications** (e.g., gaming, video streaming, DNS lookups).

**Java has built-in networking tools** for handling both TCP and UDP.

## Netværksprogrammering > Flertrådede servere

A **multi-threaded server** allows handling **multiple clients** simultaneously by creating a **new thread** for each client connection. This enables the server to **process requests independently** without blocking new incoming connections.

### Multi-Threaded Server Architecture

ClientManager (Main Server Thread)

* Creates a ServerSocket and listens for connections.
* Accepts new client connections.
* Delegates each client to a new **ClientWorker** (separate thread).
* Maintains a **list of active client threads**.
* ClientManager starts and binds to a PORT (6010 or other)

ClientWorker (Per-Client Thread)

* Handles communication for a single client.
* Reads and writes data from/to the client.
* Terminates when the client disconnects.

SimpleClient

* Connects to the server.
* Receives messages from the server.

### Ports

**Definition:**   
Logical communication endpoint, used to identify processes or services on a device.  
It routes network traffic correctly so multiple applications can use same network connection.

What is a port?

* Every device on a network is identified by IP address.
* A device can run multiple network services, web server, mail server, game server.
* Ports direct incoming and outgoing traffic.
* Ip address is like a building. And Ports are like apartment numbers.

How do ports work?

* A client chooses a port to SEND data from.
* The server listens on a port to RECIEVE data on.
* Your operating system directs the traffic based on the chosen port.

Types of Ports:

* 0 to 1023 **Well-Known Ports** (Standard services) Example: Http (80) HTTPS (443)
* 1024 to 49151 **Registered Ports** (Used by specific applications) MySql (3306)
* 49152 to 65535 **Dynamic/Ephemeral Ports** (Used temporary by clients)

**More about Ports:**

* Ports are used in both TCP and UDP
* You can check open ports with CMD “netstat -an”
* Your firewall can block or allow traffic based on port number.
* Port Scanning is used by hackers to find open ports with tools like (Nmap)
* Closed ports help prevent attacks

**Port Forwarding:**

By default, devices inside a local network (**LAN**) are **not directly accessible** from the internet. **Port forwarding allows you to make certain services reachable from outside.**

**How Port Forwarding works:**

* A request arrives at your **public IP address** on a **specific port**.
* Your **router intercepts** the request.
* The router **forwards** the request to a specific local device (pc/server etc.)

**Types of Port Forwarding:**

* **Static** (Redirects specific external port to an internal port) Example: Forward external port 8080 to internal port 80.
* **Dynamic** (Often used with VPNs, allows a remote machine to securely tunnel into a network.
* **UPnP** Universal Plug and Play (Automatic port forwarding for apps torrent/gaming)
* **DMZ** Demilitarized Zone (forwards all traffic to a specific device)

### How does routers work?

**Definition:**  
A **router** is the **traffic manager** of a network, directing data between your **private (LAN)** and the **public internet (WAN)**

**How does a router Intercept Incoming Traffic?**

**WAN** Interface (Wide Area Network)

* Connected to the internet.
* Has a **public IP address** assigned by your ISP (Internet Service Provider)
* Receives all incoming network requests from outside.

**LAN** interface (Local Area Network)

* Connected to home devices (pc, servers etc.)
* Uses **private IP addresses** (192.168.1.x)
* Distributes traffic **WITHIN** the home network.

### NAT (Network Address Translation)

**Definition:**   
NAT is a technique used by routers to **modify network packets** so that multiple devices can **share a single public IP address** while communicating with the internet.

* NAT translates between public and private IPs
* When port forwarding is configured NAT keeps a Port Forwarding Table.
* NAT ensures that **multiple devices share one public IP** while still being accessible.

**What is NAT and Why Do We Need It?**

* **IPv4 addresses are limited** (only about 4.3 billion unique addresses).
* Most ISPs **assign only one public IP** to a household.
* It works by **modifying the source/destination IPs in network packets**.
* **IPv6 solves** the IP shortage, so NAT isn’t necessary. But many networks still use NAT for compatibility with IPv4.

### How does NAT work?

A device inside a LAN wants to access the internet.

**Packet is sent from a local device** (Laptop 192.168.1.10, wants to access google.com)

* The device sends a packet: 

**Router modifies the Packet (NAT Table)**

* Since 192.168.1.10 is private IP, it cant be used on the internet.
* The router replaces the source IP with public IP (45.67.89.100)
* The Modified Packer is sent:   
  

**Google sends a response**

* The response from Google is addressed to the public IP:



**Router Uses NAT table to route the Packet**

* The router checks the NAT table and sees that the request came from 192.168.1.10
* The Router rewrites the destination IP back to 192.168.1.10 and sends it.
* To google it looks like all devices in your house came from the same public IP (45.67.89.100) But the router keeps track of who made each request.

**If you got 3 ppl using the internet, the NAT Table will look like this:  
A screenshot of a black screen

AI-generated content may be incorrect.  
Notice the same WAN.**

**Types of NAT:**

* **Static** Maps one private IP to one public IP. (Hosting a public server.)
* **Dynamic** Maps multiple private IPs to a pool of public IPs. (Large enterprise networks.)
* **PAT (Port Address Translation)** Maps multiple private IPs to a single public IP, using different ports. (Most common in home routers.)

Example of NAT rule:   
How a NAT table looks like with Port Forward:

A screenshot of a graph

AI-generated content may be incorrect.

### What is a Network Packet?

**Definition:**  
A **network packet** is a **small chunk of data** that is sent across the internet.  
Every time you send an email, load a webpage, or play a game, **your data is broken into packets**.

**Structure:**

Header (Contains instructions)

* Source IP (Where its from)
* Destination IP (Where its going)
* Protocol (TCP/UDP)
* Port Number
* Other metadata (Checksum, TTL etc.)

Payload (Actual data)

* Contents of email, part of webpage, game command.

**Firewall:**

* **By default, routers block incoming connections** for security.
* The firewall **only allows traffic on ports you specify**.

**DHCP Server:**

* Assigns Ips to devices

### XML > Sekventiel parsing af XML

XML (**Extensible Markup Language**) is a **structured format** for storing and organizing data.  
It **looks similar to HTML** but has **different purposes and rules**.

XML is used in .NET and C# a lot.   
  
Example of XML-based config files:

* App.config (Windows apps)
* Web.config (ASP.NET apps)
* .csproj (C# Project files)

**Sequential Parsing:**

* **Sequential parsing** reads XML **from start to end** rather than loading it as a full tree (DOM parsing).
* This method is **faster**, **uses less memory**, and is **natural for reading streams** of XML data.
* **Drawback:** No **random access** to XML elements, making it harder to modify data.
* In XML, **some characters have special meanings** and must be **escaped** to be valid.
* **Unescaped content** refers to **text that does not follow normal XML rules** and contains special characters like < and &.

|  |  |  |
| --- | --- | --- |
| Concept | Explanation | Used for |
| Sequential XML Parsing | Reads XML from top to bottom. Faster but lacks random access. |  |
| XmlReader | is an **iterator** that reads XML **sequentially**. **Cannot modify XML**, only reads | Large XML files, stream-based reading. |
| XmlWriter | Writes XML elements in a structured way. | Generating XML efficiently. |
| Attributes | Can be accessed using GetAttribute() or MoveToAttribute(). | Store metadata inside elements (<book language="English"/>). |
| XmlDocument | Loads the entire XML into memory (DOM-based). | When modifications are needed. |
| Namespaces | Used to avoid element name conflicts. | xmlns:bdd="http://books.docjava.dk |
| CDATA (Character Data) | Stores unescaped content inside XML. |  |
| XDocument | LINQ-based XML parsing, modern approach. |  |

A screenshot of a computer program

AI-generated content may be incorrect.

### Implementing Security for Applications

Protecting files, configuring ACLs, isolated storage and developing secure applications

**Lesson 2: Protecting Files Using Isolated Storage**

**Definition:**Isolated storage is a feature in .NET that provides a sandboxed file storage area for applications.  
**Why is Isolated Storage safer than normal file access?**

* Applications **cannot access each other's data**.
* Storage is **restricted** to the user/application context.
* It **avoids unnecessary full file system permissions**, reducing security risks.
* **Isolated Storage:** is a .NET feature that allows applications to securely store data without requiring access to the full file system.
* Works well for **storing user preferences, temporary cache, and sensitive data**.
* **Classes for working with isolated storage**: IsolatedStorageFile, IsolatedStorageFileStream.

**How to implement:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**Explanation of Code:**

* **GetUserStoreForAssembly():** Retrieves isolated storage **for the specific assembly**.
* **IsolatedStorageFileStream:** Creates a **secure file** inside isolated storage.
* Data can only be accessed by **the application that created it**, preventing unauthorized access.

**Lesson 3: Configuring Your Development Environment**

* **Least Privilege Development:** Always develop applications with the minimum privileges required to function.
* **Configuring accounts**: Ensure developers **don't work as administrators** unless necessary.
* **Debugging with least privileges**: Use techniques like **Run As** to test applications in standard user mode.

**Why is it important?**

* Ensures malicious code cannot gain High-level access.
* Helps developers test applications as a standard user.

**Best practices:**

* Enable UAC (User Account Control)
* Run Visual Studio in standard user mode to simulate real-world security constraints.

**Lesson 4: Taking Advantage of Platform Security**

**What is an Access Control List (ACLs)?**

**Definition:**An ACL is a **list of permissions** attached to an object (file, folder, etc.) that determines who can access it and what actions they can perform.

**Types of ACLs:**

* **Discretionary ACL (DACL)**: Determines **who** has access to an object.  
  Used to allow or deny user action.
* **System Access Control List (SACL):** Used for auditing access attempts.

**Configuring ACLs**:   
Can be done **manually** (GUI) or programmatically (code).

**How to implement:**

**A screen shot of a computer code

AI-generated content may be incorrect.**

**Explanation of Code:**

* Creates a file named testfile.txt.
* Retrieves the file’s **current security settings**.
* Adds an **Access Control Rule** that **denies modification** to the "Everyone" group.
* Applies the updated security rule using File.SetAccessControl().

### Pro ASP.NET Core

**Chapter 1: Putting ASP.NET Core in Context**

Understanding ASP.NET Core: ASP.NET Core is a cross-platform, high-performance framework for building modern web apps.

Frameworks within ASP.NET Core:

* **MVC (Model-View-Controller):** Used for creating structured web applications with clear separation of concerns.
* **Razor Pages:** A lightweight alternative to MVC for simple web apps.
* **Blazor:** Allows building interactive UIs using C# and WebAssembly.
* **Hosting and Deployment:** Applications run on Kestrel, IIS, or Nginx, and can be containerized using Docker.

**Chapter 4: Configuring ASP.NET Core Applications**

### Middleware

* Middleware is software that is executed on every request and response.
* **Request Pipeline:** Requests flow through middleware components before reaching the final response.
* **Adding Middleware:** Done via UseMiddleware<T>() or inline with app.Use().
* **Built-in Middleware:** Authentication, Authorization, Routing, Exception Handling, Logging.

Example:

A screen shot of a computer code

AI-generated content may be incorrect.

### Dependency Injection

* is a **design pattern** used in ASP.NET Core to manage the dependencies between objects.
* Instead of **creating instances manually** inside a class, the **framework provides the dependencies automatically**.

Example Without:

A screenshot of a computer

AI-generated content may be incorrect.

* OrderService is **tightly coupled** to PaymentService
* If PaymentService changes, OrderService **must also be modified**.
* **Difficult to test** (cannot replace PaymentService with a mock).

Example With DI: A screenshot of a computer program

AI-generated content may be incorrect.

* Now OrderService **depends on an interface** (IPaymentService) instead of a concrete class.
* The **implementation is injected via the constructor**.
* **Easier to swap implementations** (e.g., use a fake/mock service in tests).

### Registering Dependencies in ASP.NET Core

Dependencies are registered inside **Program.cs**.

A computer screen with white text

AI-generated content may be incorrect.

**Service Lifetimes:**

* Transient (New instance every request)
* Scoped (One instance per HTTP request. Used in Database contexts (DbContext))
* Singleton (Single instance for the lifetime of the app)

### Injecting Services into Controllers:

The framework automatically injects IOrderService into OrdersController.

A screen shot of a computer code

AI-generated content may be incorrect.

**Why use DI?**

A screenshot of a computer program

AI-generated content may be incorrect.

* **Tightly Coupled:** OrderService **directly creates** PaymentService and EmailService.
* **Difficult to Test:** You cannot replace PaymentService with a mock for testing.
* **Difficult to Change:** If you want to switch to a different payment provider, you **must modify OrderService**.

A screenshot of a computer program

AI-generated content may be incorrect.

**Why?** Now OrderService doesn't care about the implementation, only the interface.

A screenshot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

Now **OrderService** is loosely coupled!

You can swap out **PaymentService** and **EmailService** without modifying **OrderService**.

A screenshot of a computer program

AI-generated content may be incorrect.

Now ASP.NET Core provides the correct instances when needed.

A screenshot of a computer program

AI-generated content may be incorrect.

**What Would Happen If We Didn't Use Dependency Injection?**

* **Harder to Test –** We can't mock **PaymentService or EmailService.**
* **Harder to Change** – Switching to a new payment provider would require modifying multiple classes.
* **Code Duplication** – If OrderService and CartService both need PaymentService, we’d create multiple instances manually.
* **Performance Issues** – Creating a new instance every time increases memory usage.
* **Tightly Coupled Code** – Services depend on **specific implementations**, making it harder to maintain.

### Stub-Skeleton Pattern

### Netværk

### MVC og Razor Pages

### Bootstrap Beginner Crash Course

### Større program-eksempler

### Netværkskode (C#)

### Andet