NP

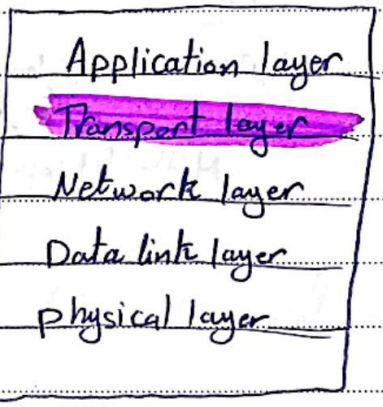
SARAH KASASBEH

POWERUNIT

Lecture 1

Introduction

- TCP → Reliable, in-order delivery (reliable transport between sending and receiving processes. / No timing (concerned with the accuracy of data). ex:- file transfer, e-mail.
- UDP → Unreliable ("best-effort"), unordered data transfer between sending and receiving processes. (Concerned with the delay to transfer data. ex:- streaming multimedia.
- TCP, UDP are transport layer protocols.



TCP/IP

Chapter 1

- Digital signal:- n-discrete levels of electrical signals.
- analog signal:- ∞ number of levels (continuous).
- Host (end system)


client server

- servers → powerful hardware
- clients → moderate hardware

- Data centers → a large group of networked computer servers typically used by organizations for processing data. ex:- google colab

Lecture 2

- almost every network adapter has an address that uniquely identifies it (identifies the NIC "Network Interface card")

- We have two types of addresses not only one to avoid layers overlapping (ex:- Network layer addressing is different from physical layer addressing).

- Port number addressing in transport layer for protocols types.

addresses
- physical (6 bytes) → MAC address
- logical (4 bytes) → IPV4 / IP V6 → network layer

Computer networks
- Packet-switched: less expensive
- Circuit-switched: better quality → ex:- telephone networks

- in packet-switched data is broken into packets and each packet is handled separately, to save time and resources in case an error occured.

- HTTP → application layer protocol
- TCP / UDP → Transport layer protocol
- IP → network layer protocol
- IEEE 802.3 (Ethernet) → Data link layer protocol.

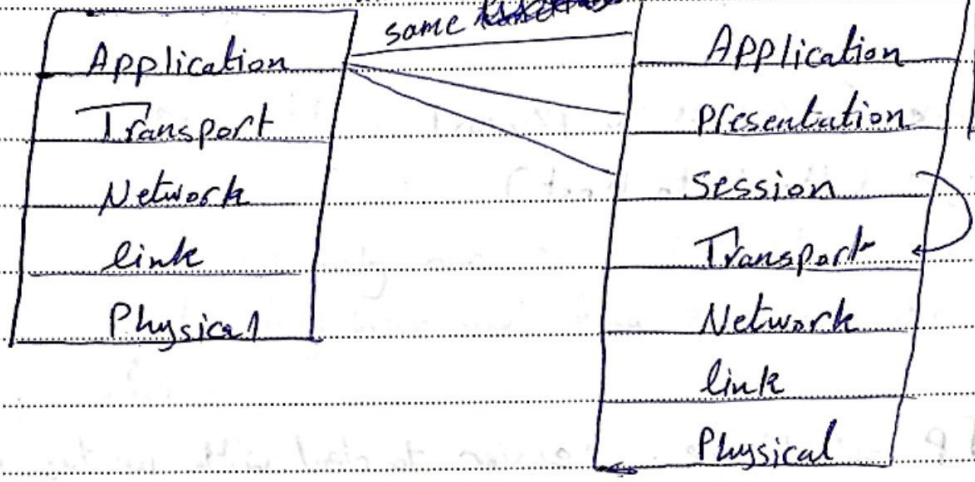
- each layer is isolated from the other layers

Network models

better ↩

TCP/IP (5 layers)

OSI (7 layers)



- RARP is not used these days for security reasons
- ARP → match the IP address with the suitable MAC address

- protocols code (ex:- TCP) are on the OS (kernel), and can be modified.

- Router is layer 3 device (functions), but it can deal with Application layer (sending or receiving services or)

Protocol Unit consists of

- Header
- Data (Protocol data Unit (PDU))

- header contains information about the protocol and services (Figure 2.4 . (DG = H7 + D7))

PDU in Application layer → message
" in Transport layer → segment
" in Network layer → packet or datagram
" in Data link layer → frame.

- example 2.3

Trailer → error detection
Header → source and destination

Dst.	Interface

- In transport layer (process to process)
- In network layer (Host to Host)

- DNS

IP ⇒ Name "easier to deal with unique name for each website instead of dealing with IP addresses".

- We may have more than one IP address with the same name (used in ex:- backup server)

- Why not centralize DNs? (why not save on the same server?)
  - single point of failure
  - traffic volume
  - maintenance
  - distant centralized database

- Converting (Name → IP address) in 2 ways:-

1- iterated query → using local DNS server "on the same LAN"
↳ send packet to the local DNS server

2- recursive query ↳ root DNs → TLD DNs → authoritative DNs
root DNs: .com .edu .io
TLD DNs: .edu
authoritative DNs: gives the IP

Lecture 4

* Logical Addr. (IP addr.)
* Mask
* local DNS

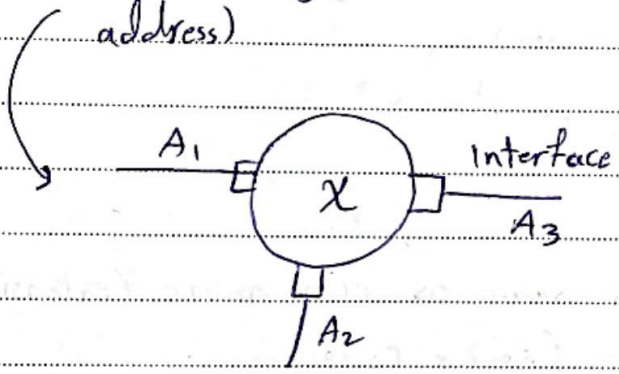
→ so each device can communicate with other devices on a network

* DHCP → in application layer.

Note:- Host is something offers service to Nodes.

Note:- Node is a device which participate in the networking connection for forwarding packets

- Review of IPV4 addresses
  - Host has only one NIC (1 address)
  - Nodes may have more than ~~one~~ one NIC (more than on address)



- Class A → 1 byte → network id
- Class B → 2 bytes → network id
- Class C → 3 bytes → network id

network Adr. ↑ ↳ Broadcast Adr.

↳ host id (0 - 255)

ex:- 193.188.6.0 → class C

↳ network id

- host id (0, 255) → reserved.
- classful Ip addresses → waste in IP addresses.

Lecture 5

- NAT → Network address translation :- It's a way to map multible local private addresses to a public one before transferring the information

NAT table for G.	
Private Addr.	Dest. Addr.

- PAT → Port address translation :- private IP addresses are translated into the public IP address via port numbers allowing multiple hosts to share a single IP address while using different port numbers.

- java intro. Chapter 2

  - File name should be the same as class name (extension → .java)
  - main code in source file (source code)
  - Compile (javac File name.java) → byte code
  - Run Bytecode
  - Result (console)
    - on

- Compilation in windows :- "CMD"

  javac file.java

- Run :-

  java file

/ /

lecture 6

- Compiling java source code → bytecode which can run on any computer with JVM (Java virtual machine)
  - ↳ software that interprets java byte code.

- Anatomy of a java program:-

1) class name → starts with an uppercase letter.
2) Main method → in order to run the class, the program starts here. (___)
3) statements → actions {___}
4) statements terminator → ; end of the statement.
5) Reserved words → have a specific purposes, and cannot be used for other purposes.
6) Comments → //___ , /* */
7) Blocks

- Programming errors :-

1) Syntax errors → detected at compile time
2) Runtime errors → " " run time
3) logical errors → ex:- ~~unexpected output~~ unexpected output.

"import" statement → tells the compiler that we want to use a class that is defined under a package.

Lecture 7

- Exception :- unwanted or unexpected event, which occurs during the execution of a program "at run time" that disrupts the normal flow of the program's instruction.

- Java exception handling :- mechanism to handle runtime errors.

- A package in Java is used to group related classes.

- Try - Catch
  - The try statement allows you to define a block of code to be tested for errors while it is being executed. The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

- Type of exceptions in Java
  - NullPointerException
  - Arithmeticexception
  - Ioexception
  - Arrayindexoutofboundsexception
  - Classnotfoundexception
  - Runtimeexception
  - Filenotfoundexception
  - :
  - etc

Catch (exceptiontype name)

- you can declare your own exceptions too.

- The finally block in Java is used to put important codes, finally block executes whether exception rise or not and whether exception handled or not.

Chapter 3

Internet addressing with Java

- java.net.InetAddress → represents hostname and ip addresses.
  it contains 2 public static methods return InetAddress

- static methods in InetAddress :-
  1) getByName → takes the hostname and returns the ip address and the hostname.
  2) getAllByName → takes the host name as an argument, and returns array of InetAddress.

- each InetAddress is associated with a NIC (Network Interface Card)

- There are other methods, but only 2 return the InetAddress.
  - return string
  - Total → 6 methods

Lecture 8

- "ifconfig" → linux command } → To display all interfaces.
- "ipconfig" → windows command
- each interface has either ipv4 or ipv6 addresses.
- import java.net.*; → to use the InetAddress
- args.length → # of arguments
- System.exit(0) → termination

/ /

* A Constructor is a block of code that initializes a newly created object. " it doesn't return a value, it can be used to initialize an object".
* A method is a collection of statements which returns a value upon its execution.

* The toString method is used to return a string representation of an object.

* ~~isReachable~~ isReachable method → in Application layer. test whether that address is reachable, it takes timeout as a parameter "the time, in milliseconds, before the call aborts", and it returns a boolean indicating if the address is reachable.

* A constructor is a block of code that initializes a newly created object. "it doesn't return a value, it can be used to initialize an object"
* A method is a collection of statements which returns a value upon its execution.

* The toString method is used to return a string representation of an object.

- ~~isReachable~~ isReachable method → test whether that address is reachable, it takes timeout as a parameter "the time, in milliseconds, before the call aborts", and it returns a boolean indicating if the address is reachable. ← in Application layer.

Chapter - 4 Lecture 9

- Data streaming :- is the process of transmitting a continuous flow of data. Data stream :- Consists of a series of data elements.

- transmission of data → serially

- How streams work? → at byte level (raw data)
  - ↳ super class for input stream → java.io.InputStream
  - ↳ Super class for writing "output stream" → java.io.outputStre

- int read() → returns an integer (the next byte of data from the stream)

- System.err.println → error console

Creating input stream → InputStream fileInput = new FileInputStream(args[0]);
(InputStream: super class; FileInputStream: child class)

. methods from child class.

- Read the first byte → int data = fileInput.read();
- System.out.write(data) → display the read data on the screen.
- fileInput.close(); → to close the file

(1/11/2022) lecture 10

- Inheritance :- mechanism by which one class is allowed to inherit the features (methods) of another class. "creating new classes based on existing ones".

- A class that inherits from another class can reuse the methods (override) and fields of that class.

- inputStream1 = new FileInputStream(args[0]);
  - inputStream1 → declared type
  - FileInputStream → actual type

Note :- in inheritance only the object of the subclass is created.

* methods are from the actual type

Buffering :- block of bytes in memory used to cache data, to reduce the number of calls to the OS.

Abstract class :- it may or may not include abstract methods. it cannot be instantiated, but they can be subclassed, the subclass provides implementations for all of the abstract methods in its parent class.

Lecture 11

- in any file → each letter is considered a byte; (ex: welcome → 7 bytes)
- space → is considered a byte.
- reading and writing to file → 2 args [source, destination]

- Byte level communication is inefficient ~~read byte by byte~~
  L: read byte by byte / send byte by byte.
  ↓
  instead → buffering (once the buffer is full, send group of bytes / read)

Filter streams

. Input stream and output stream allow to read and write bytes, either singly or in groups.

. Filters are connected to streams by their constructor

ex: FileInputStream fin = new FileInputStream("data.txt");
BufferedInputStream bin = new BufferedInputStream(fin);

⇓

1) fileInputStream object is created (fin)
2) BufferedInputstream object (bin) is created by passing (fin) as an argument

⇓

intermixing calls to different streams connected to the same source may violate several implicit contracts of the filter streams → so use only the last filter to do reading and writing →

/ /

ex:- InputStream in = new FileInputStream("data.txt"); } → Polymorphism
in = new BufferedInputStream(in);

⇓

there's no longer any way to access the file input stream so you can't accidently read from it and corrupt the buffer

- Polymorphism → the ability of a class to provide different implementations of a method, depending on the type of object that is passed to the method. (uses Is-A test). Polymorphism occurs when there is inheritance

- As we can say in the following example, each shape implements the area method in different ways according to the type of the object, and the passed arguments.

```
class Shapes {
  public void area() {
    System.out.println("The formula
for area of ");
  }
}
class Triangle extends Shapes {
  public void area() {
    System.out.println("Triangle is
½ * base * height ");
  }
}
class Circle extends Shapes {
  public void area() {
    System.out.println("Circle is
3.14 * radius * radius ");
  }
}
class Main {
  public static void main(String[]
args) {
    Shapes myShape = new Shapes();
// Create a Shapes object
    Shapes myTriangle = new
Triangle();  // Create a Triangle
object
    Shapes myCircle = new Circle();
// Create a Circle object
    myShape.area();
    myTriangle.area();
    myShape.area();
    myCircle.area();
  }
}
```

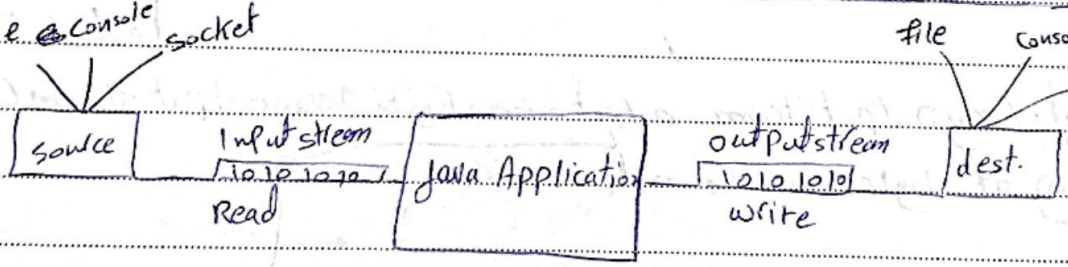
ex: InputStream in = new FileInputStream ("data.txt"); } → Polymorphism
in = new BufferedInputStream (in);

⇓

there's no longer any way to access the file input stream so you can't accidently read from it and corrupt the buffer

- Polymorphism → the ability of a class to provide different implementations of a method, depending on the type of object that is passed to the method. (uses Is-A test). Polymorphism occurs when there is inheritance

- As we can say in the following example, each shape implements the area method in different ways according to the type of the object, and the passed arguments.



lecture 12

Inputstream in = new InputStream ("data.txt"); X error
becuse inputstream is abstract class.

InputStream in = new FileInputStream (f); → correct.
in = new BufferedInputStream;

(in) is now instance from BufferedInputstream only.

- PrintStream (class, adds functionality to another output stream)
- read and write operations must take place on the new filter stream not on the (FileOutputStream / FileInputStream).

```
FileOutputStream fout = new FileOutputStream ('data.txt');
PrintStream pout = new PrintStream (fout);
pout.println ("hello world");
```

↳ create file called (data.txt) and write (hello world) on it.

```
> type data.txt
```

↳ to show context of data file (windows)

data.txt
hello world

Lecture 13

- ByteArrayOutputStream output = new ByteArrayoutputstrem();
  ↳ Array of bytes on the output stream
- Buffer's default size = 512 bytes
- to force bytes to be written out to output use (output.flush();)
  ↳ bytes will be written on the Array.

if you don't invoke flush();

Array will stay empty because the buffer isn't full.

- once the buffer is full, it'll write its content on the Array directly → by default.

- What Causes "FileNotFoundException"?
  1) If a file with the specified pathname doesn't exist.
  2) If a file with the specified pathname is inaccessible
     ↳ for example: if the file is read-only and is attempted to be opened for writing or vice versa. "العكس صحيح"

- An interface in Java is a blueprint of a class. It has static constants and abstract methods. There can be only abstract methods in the Java interface, not method body.

lecture 14

```
FileOutputStream file = new FileOutputStream("file.txt");
DataOutputStream data = new DataOutputStream (file);
data.writeInt(65);
data.flush();
data.close();
```

- Int in Java is 4 bytes

output → A
↳ because the text editor translates the integers into the ASCII code.

- Creating new file using cmd
  - in linux → 'touch' command
  - in windows → copy nul command.

Notes about Abstract Classes and Interfaces in Java.

** Abstract Class:

* declared using "**abstract**" keyword.

* subclasses "extends" abstract class.

* abstract class can have **implemented methods** and 0 or more **abstract methods**.

* we can extend **only one** abstract class.

* an **abstract method** cannot be contained in a **non-abstract class**.

* if a **subclass** of an **abstract superclass** does not implement **ALL** the abstract methods, the subclass must be defined as **abstract**.

* in a **non-abstract subclass** extended from an **abstract class**, **ALL** the abstract methods **MUST** be implemented.

* abstract methods are non-static.

* an **abstract class** cannot be instantiated using the "**new**" operator, but you can still define its constructors, which are invoked in the constructors of its **subclasses**.

* a class that contains **abstract methods MUST** be **abstract**. However, it is possible to define an **abstract class** that **does not** contain any **abstract methods**. In this case, you **cannot** create instances of the class using the "**new**" operator. This class is used as a base class for defining subclasses.

* a **subclass** can be abstract even if its **superclass** is concrete.

* a **subclass** can override a method from its **superclass** to define it as **abstract**. This is very unusual, but it is useful when the implementation of the method in the **superclass** becomes invalid in the **subclass**. In this case, the **subclass MUST** be defined as abstract.

* you **cannot** create an instance from an **abstract class** using the "**new**" operator, but an **abstract class** can be used as a **data type**. For example, the following statement, which creates an array whose elements are of the myClass (abstract class) type, is correct.

```
myClass[] objects = new myClass[10];
```

** Interface:

* declared using **"interface"** keyword.

* subclasses **"implements"** interfaces.

* in Java 8 onwards, interfaces can have **default** and **static methods**.

* we can **implement** multiple interfaces.

* interfaces **cannot** be used to create objects.

* interface methods **does not** have a body - the body is provided by the **subclass**.

* on implementation of an interface, you **MUST override ALL** of its methods.

* interface methods are by default **abstract** and **public**.

* interface attributes are by default **public**, **static** and **final**.

* an interface **cannot** contain a **constructor** (as it cannot be used to create objects).

** Why and when to use Interfaces?

To achieve security - hide certain details and only show the important details of an object (interface).

Java does not support **"multiple inheritance"** (a class can only **inherit** from **one superclass**). However, it can be achieved with **interfaces**, because the class can **implement multiple interfaces**.

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have	Members of a Java interface are

class members like private, protected, etc.	public by default.

9)**Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

Abstract class achieves partial abstraction (0 to 100%) whereas **interface** achieves full abstraction (100%).

/ /

- PrintStream class → extends the abstract class OutputStream
  - converts the primitive data into the text format instead of bytes
  - auto flush → flush when if the println() method is invoked or if an array of bytes is written in to the printstream.
  - it doesn't throw any input/output exception

* Note : The close() method is used to close the file and releases all resources associated with the stream (reduce garbage data)

- BufferedInputStream class → The purpose of I/O buffering is to improve performance (Reading larger chunckes of bytes and buffering them can speed up I/O operations. Rather than read one byte at a time.)
  - when you are finished reading data from the buffer you must close it, by calling the close() method which is inherited from the InputStream.

Lecture 15

- primitive data types => size = 8-bits (1-byte)
  - (byte / short / int / long / float / double / boolean / char)

- readers and writers deal with data with size greater than 1 byte (not the primitive data types only) "larger data sets"
  - ⇓
  - support 16-bits instead of 8-bits / Unicode characters
  - texts in other languages.

- java.io.Reader Class → no public constructor / for reading character streams
  - no instantiation.
  - CharArrayReader Class → used to read character Array as a stream, it inherits Reader class.

- Reader reader = new InputStream (new FileInputStream (fileName), "UTF-8");
  - methods from Reader class

Lecture 16

- FileReader Class → used to read data from the file, it returns data in byte format like (FileInputStream class).
  - extends InputStreamReader
  - Constructors take diff. types of objects (String / File).

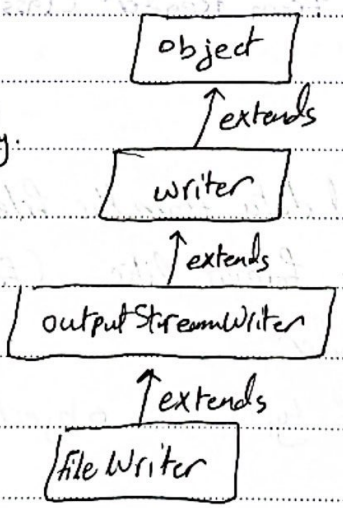
- StringReader Class → takes an input String and changes it into character stream. It inherits Reader class
  - Closing this class is not necessary.

- InputStreamReader Class → is a bridge from byte streams to character streams, it reads bytes and decodes them into charest using a specified charest.
  ↓
  بدعم اللغة العربية

"Writers"

- Writer Class → equivalent (outputStream).
  - ↳ abstract class, it is not useful by itself. However, it's subclasses can be used to write dat.
    - ↳ "no public ~~method~~ Constructor".
  - ↳ used to write Character Streams.

- FileWriter Class → used to write characters oriented data to a file
  - ↳ you don't need to convert string into byte array because it provides method to write string directly. (.write)
  - ↳ it creates the output file if it is not present already.



- StringWriter Class
  - ↳ inherits the writer class / (Sockets and files are not used so closing the StringWriter isn't necessary)
  - ↳ Character stream that collects output from string buffer which can be used to construct a string.

- OutputStreamWriter class → used to convert character stream to byte stream
  - the characters are encoded into bytes using a specified charset. (write() method)

---

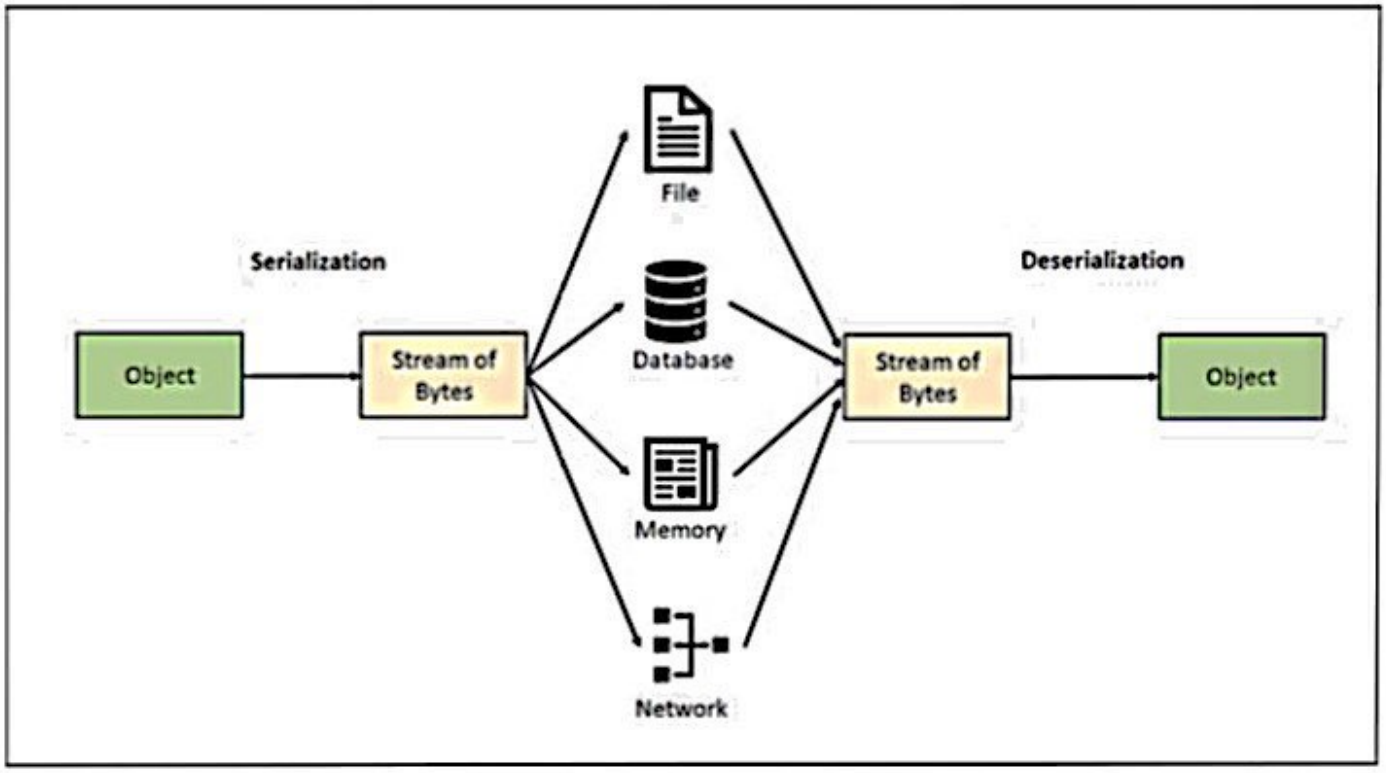
lecture 17

- Object persistence
  - denotes the lifetime of an object. (they can be saved to a data store and be recreated at a later point in time.
  - a persistent object continus to exist beyond the duration of the process that creates it.

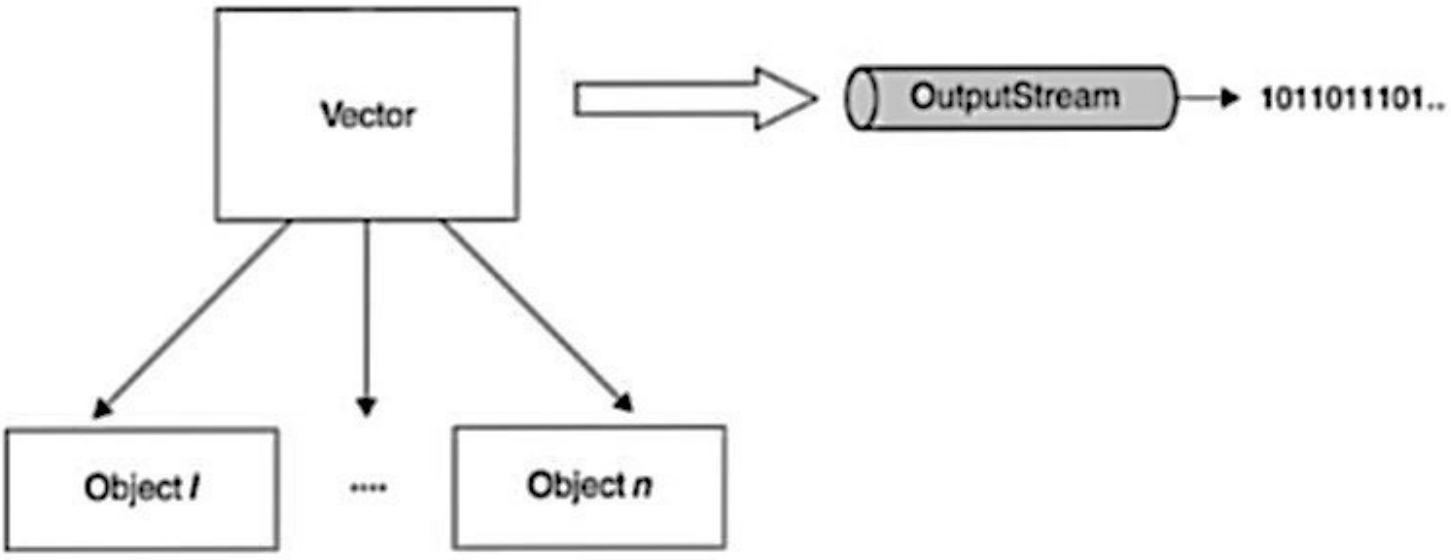
- Object serialization
  - used to save the state of an object in order to be able to recreate it when needed
  - → is the process of converting an object into a stream of bytes to store the object or transmit it to memory.
  - " technique by which object persistence is realised.

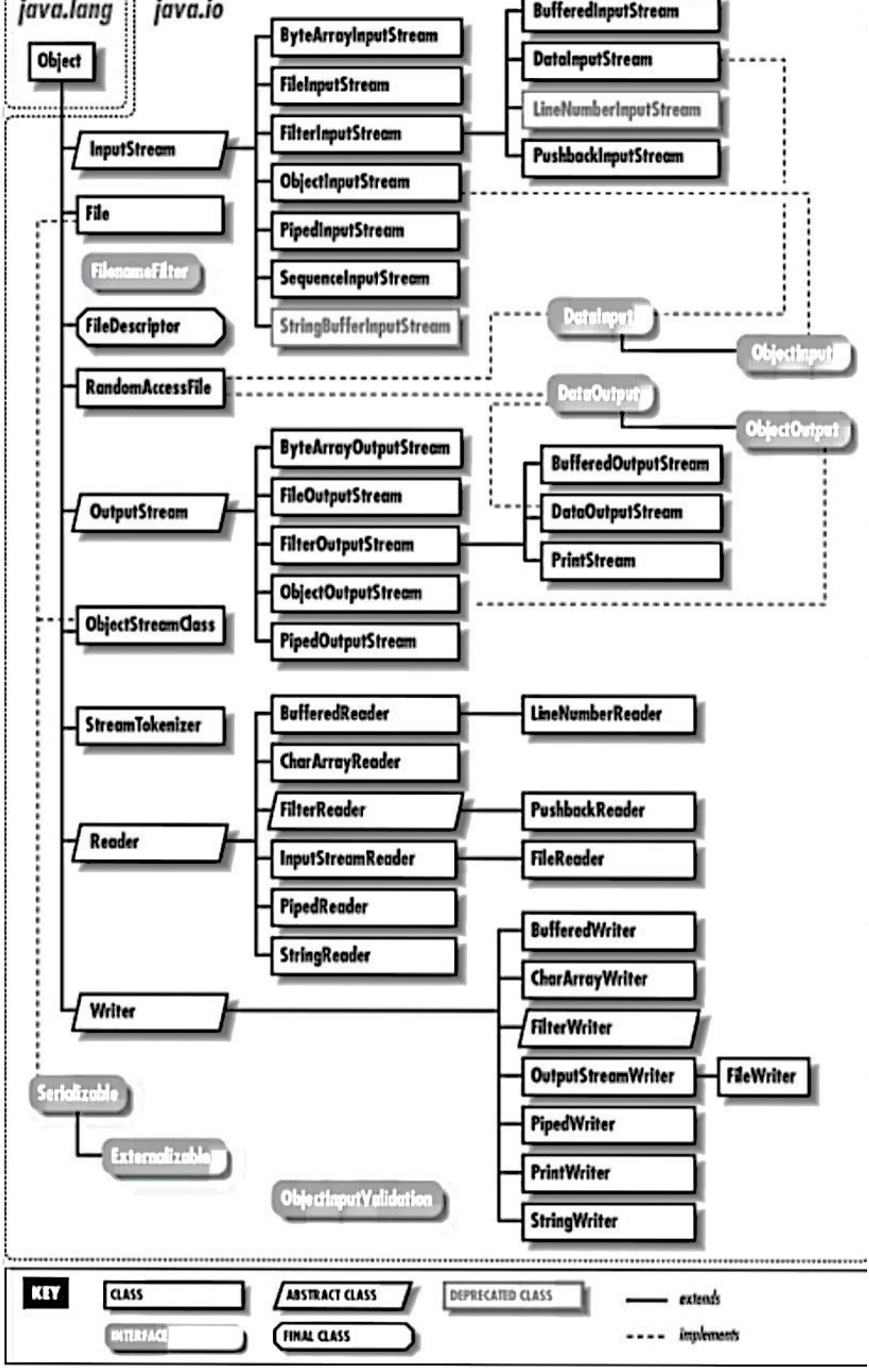
- any object that implements (java.io.Serializable) interface may be serialized
  - marker interface ∴ used as a tag that inform java compiler as a message so that it can add some special behavior to the class implementing it.

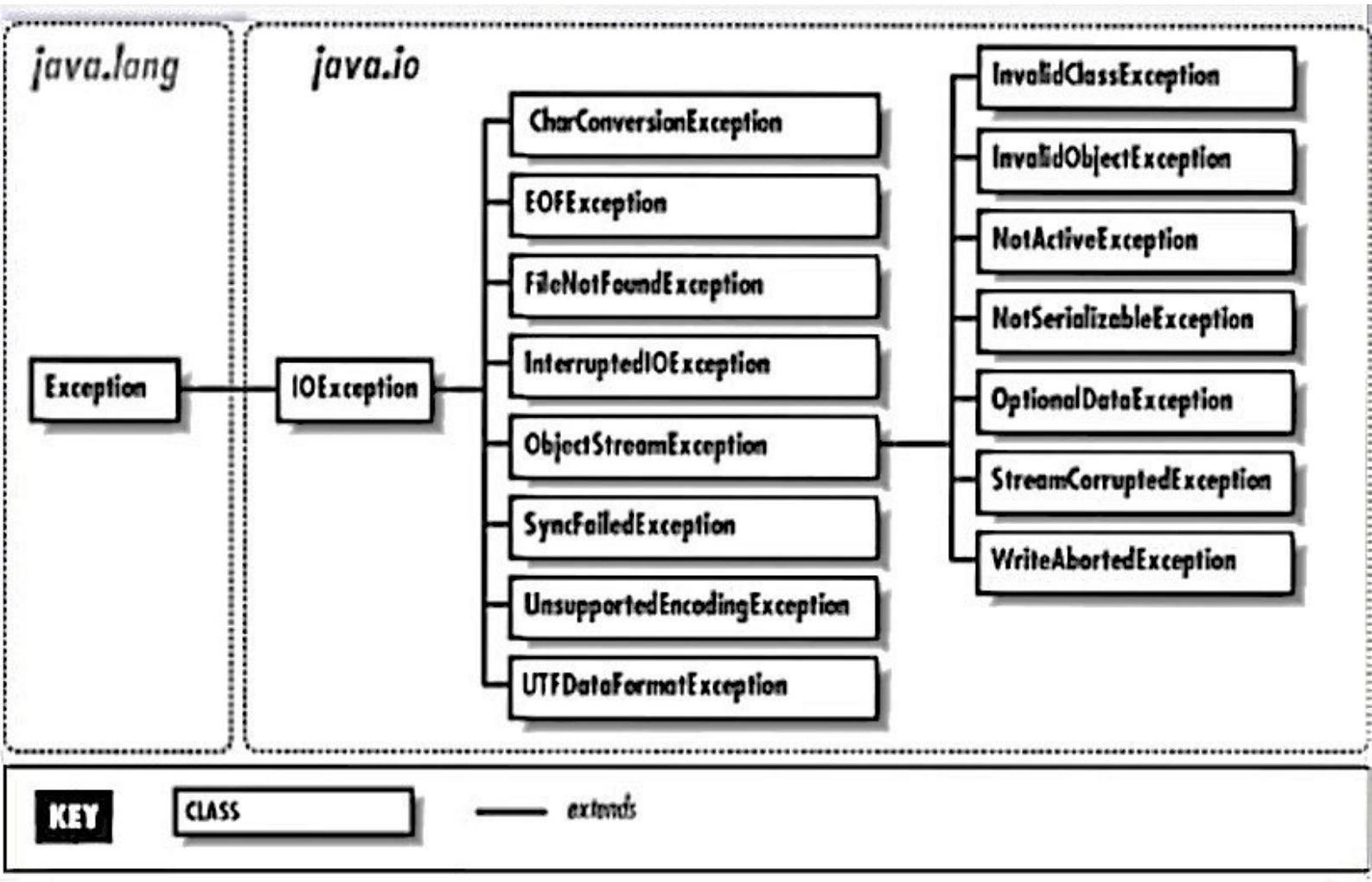
What is Object Serialisation?



What is Object Serialisation? (cont.)







Lecture 18

- Type Casting → feature using which the form or type of a variable or object is cast into some other kind of object.

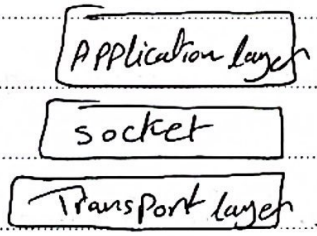
- Chapter 5 "Up-socket_programming"

  - two types of transport layer service via socket API:-
    1) unreliable datagram (related to the UDP)
    2) reliable, byte stream-oriented (related to the TCP)

- UDP (unreliable) → it provides no guarantees to the upper layer Protocol for message delivery.

- TCP takes longer time because it is reliable. (Delay ↑)

- socket → a door between application process and end-end-transport Protocol (UDP or TCP).



→ controlled by operating system.

- a socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.
  - it is a combination of ip address and Port number (we should create the socket first).
    ↳ "claim resources."

- Client/server socket interaction

server:-
- Create socket → in order to connect to another machine
- Bind port → determines where and how a message will be sent or received.
- listen / accept → Waiting for connections
- send / receive
- close socket

Client:-
- creat socket
- Connect to the server
- send / receive
- Close socket

- header size in UDP = 8 bytes
- header size in TCP = ~~5~~ 20 bytes

- Multiplexing :- gathering data from multible sockets of the sender, enveloping that data with a header, and sending them as a whole to the intended receiver.

- Demultiplexing :- Delivering received segments at the receiver side to the correct socket.

- Connectionless Demultiplexing :- requires the IP Address and destination port#. it requires only that two-tuple to identify where to send the datagram. (~~another~~ whether it be TCP or UDP).

- Connection-Oriented Demultiplexing :- is only TCP viable and requires four-tuple (Source IP / Source port# / dest. IP / dest. port#)

Some notes:-

lecture 20

- operating system generates random port numbers.
- TCP takes longer time to transfer data than UDP due to connections which take around 3 round trip time, and the header overhead (20-60) bytes.
- Two-Army problem (Two-Generals' problem) → never any way for computer "A" and computer "B" to guarantee that the data they exchage is received and acknowledged.
- Thread → is a light weight process (we need to only create a copy of a certain thread in order to perform a specific part from the whole process).

- Java support for user datagram Protocol (UDP)
  - two classes are used in Java to support the UDP:-
    - ↳ java.net.DatagramPacket
    - ↳ java.net.DatagramSocket

- Remember:- packets are sequence of bytes, include (header) IP address and port #.
- java.net.DatagramPacket → used to create a packet and control its contents and its size.
  - ↳ represents the packets intended for transmission.
- Why do we create a datagram packet?
  - sending and receiving data using UDP → same class for sending and receiving but different constructors.

- DatagramSocket class → Provides access to a UDP socket, which allows UDP packets to be sent and received.
  - ↳ same DatagramSocket for sending and receiving
  - ↳ represents a socket for sending and receiving packets
  - ↳ mechanism used for transmitting datagram packets over network.

Lecture 21

- creating a client DatagramSocket

  DatagramSocket()

- creating a server DatagramSocket → Binding

  - DatagramSocket(int port) — throws java.net.SocketException

- DatagramSocket(int port, InetAddress addr) → source ip address.
  - ↳ used when a machine is known by several IP addresses you have to specify the IP address, to which a UDP service should be bound.

- listening for a UDP packets

packet —binding→ DatagramSocket —translation→ DatagramPacket → UDP application

- DatagramSocket.receive() → copies a UDP packet into the specified DatagramPacket. → then the content will be processed (reading through DataInputStream class.)

- Sending UDP packets

1) create DatagramPacket
2) set the address and port information. (destination).
3) write the data for transmission into byte Array.
↳ packet buffer.
4) use (socket.send (Packet).

- Lack of Guaranteed packet sequencing → causes missing packets

- Controlling the order in which packets are processed (more complex)
- used by applications that require sequential access to data.
- If a packet arrives out of order, it can be buffered until the earlier packets are received

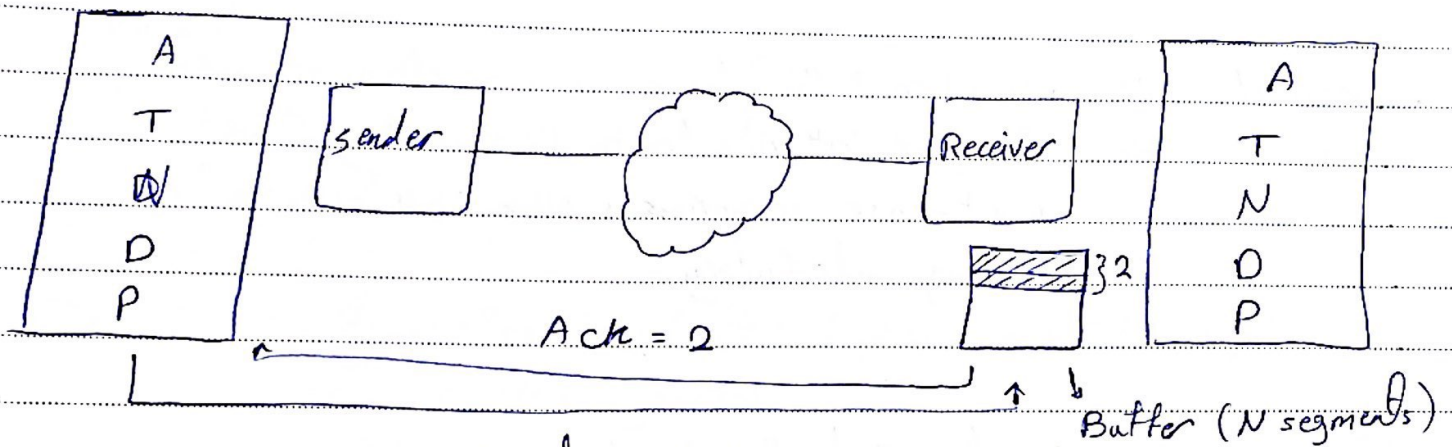
---

- Lack of Flow Control
↳ . To avoid flooding system with more data than it can handle
. helps to prevent systems from becoming overloaded throug :-
1) limiting the packets rate
2) limiting the number of packets that have not yet been acknowledged

Chapter 6

11/12

- TCP → • point-to-point connection (connection-oriented)
  - reliable Protocol "guarantees the delivery of data"
  - slower than UDP.
  - error checking "provides flow control"
  - Retransmission of lost packet is possible in TCP.



rwnd = N - 2

↳ buffer can accept N-2 segments at this point. this process called "flow control".

- in case the buffer is full → rwnd = 0

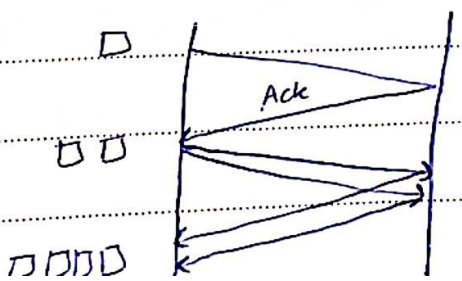
- Flow control → reduce drop on receiver.

- Congestion Control → TCP uses a congestion window in the sender side to congestion avoidance. This window indicates the maximum amount of data that can be sent out on a connection without being acknowledged.

↓

determined by ~~congestion window~~ packet loss and round trip time

rwnd = min (rwnd, cwnd)



- Nagle's algorithm is a means of improving the efficiency of TCP/IP networks by reducing the number of packets that need to be sent over the network.
  - java.net.setTcpNoDelay();
    - ↳ used to enable/disable TCP_NoDelay which disable/enable Nagle's algorithm.

- Deadlock state → is when both sides on a connection are waiting for the other side to do something. The worst-case scenario is when both sides end up waiting indefinitely.

- TCP is a stream-based method of network communication.

- A client creates a socket at its end of transmission, and connect the socket to the server. when a connection is established, the server creates a socket at its end and, the client and server can now ready communicate through writing and reading methods.
  - ↳ from chapter 4.

- no datagram or Packets in TCP, only byte streams represented as an InputStream or an outputStream.

- TCP provides guaranteed delivery of bytes of data

Note:-
UDP → no friends protocol
IP → Best effort

- header size = 20 ~~→~~ 60 bytes

- if all Parameters are equal, TCP execution time is longer than UDP execution time.

- TCP is connection oriented → an exclusive connection must first be established between the client and the server for communication to take place.

- Advantages of TCP over UDP :-

1) Automatic error control → (checksum / acknowledgment / time-out).
2) Reliability → (ensures that data is not damaged, lost, duplicated, out of order)
3) Ease of use →

- TCP_NODELAY → allow the network to bypass Nagle Delays by disabling Nagle's algorithm.

Data sent between two machines participating in a TCP connection is transmitted by IP datagrams
↳ network layer.

Communication in TCP sockets is treated in the same way as file input and output.

- we use the concept of communications port number.

- TCP connection requires pieces of information in order to connect:
  1) IP address of the remote machine
  2) Port number of the remote machine
  3) IP address of the local machine
  4) Port number of the local machine

  → must be defined.

/ /

- 2 socket classes

java.net.socket → client sockets

java.net.serversocket → server

- java.net.Socket → client

→ communication channel between two TCP communications ports belonging to one or two machines

- Constructors :-

1) Socket (InetAddress addr, int port) / Here the address of the local machine and the port number are from the OS.
   - addr → could be used as a String
   - of the server (dest.)

2) Socket (InetAddress, int port, InetAddress, int port).
   - InetAddress, int port → could be used as a string → server (dest.)
   - InetAddress, int port → local machine (source)

- if the remote machine is not responding, the constructor may block for an indefinite amount of time. O.W → connection is established.

- Void close() → closes the socket connection and free resources allocated to the socket. you should flush any output streams before closing a socket connection to make sure that data is sent.

Note:- setReceiveBufferSize → Buffer size of TCP (of the operating System)
↳ it is not preferable to use it because some OS. don't allow adjust it.

- Port scanning → method of enumerating the open or active ports of a target machine, to list the open ports in order to know the applications and services that are currently running.

- Socket s = new Socket ("", "");
↳ once this statment is done (executed) successfully without any exceptions, a connection will be established. (three-way handshaking)

- Note:- By convention, the first 1024 ports are reserved for standard services such as:-

Port 20 → FTP
" 23 → Telnet
" 25 → SMTP
" 80 → HTTP

Thread.sleep(); → put the current thread in a wait state. the actual time that the current thread sleeps depends on the thread scheduler that is part of the OS.
→ the thread can be interrupted and such in case (InterruptedException) is thrown.

- Thread.sleep(long);
↳ Duration of sleep

- getInputStream() → returns an input stream for the given socket.
  - if you close the returned input stream then it'll close the linked socket.
  - returns an input stream for reading bytes from this socket.

```
(socket.getInputStream());
```

- getOutputStream() → returns the output stream for the given socket.
  - if you close the returned output stream then it'll close the linked socket.
  - returns an output stream for writing bytes to this socket.

```
(socket.getOutputStream());
```

- We should set the socket option using socket.setSoTimeout(time) to specify the time the sender waits for a response from the receiver, because in case server stalls it'll wait indefinitely.

Note: in linux → "netstat" command prints network connections (for all ports)
- (netstat -ant) →
- (netstat -ant | grep 80) → check the number of connections on port 80 only.

in windows → (netstat -afb | findstr 80)

• day time server → Port 13 (original service).

• TCP states :-

• closed → There is no connection

• Listen → The local end-point is waiting for a connection request from a remote end-point

• Established → the third step of the three-way handshaking was performed. (The connection is open)

• Fin-Wait-1 → The first step of an active close was performed. The local end-point has sent a connection termination request to the remote-end point

• close-wait → The local end-point has received a connection termination request and acknowledged it (Passive close)

• Fin-wait-2 → The remote end point has sent an acknowledgement for the previously sent connection termination request. the local end-point waits for an active connection termination request from the remote end-point.

• Last-Ack → The local end-point is waiting for an acknowledgment for a connection termination request before going to the time-wait state.

• Tim-wait → The local-end-point waits for twice the maximum segment lifetime (MSL) to pass before going to closed to be sure that the remote end-point received the acknowledgement

• Server Socket class

- Constructor (ServerSocket (int Port))
  - ↓ called socket factory
  - ↳ if port = 0 → it'll take random port number from the operating system, so the client won't know which port it is binding to.
- The (accept()) → function shall extract the first connection on the queue of pending connections (used by server to accept a connection request from a client).
  - ↳ Returns an object of type client socket.
  - ↳ if the listen queue is empty of connection requests, accept() shall block until a connection is present (blocking state)
- Creating server socket doesn't mean that a connection has been established. It only creates an object of type server socket, it'll wait until a client socket is created, then a connection will be established.

Note:- getInetAddress() → returns an InetAddress object.

- writing on a socket :-

```
OutputStream out = Socket.getOutputStream();   // Socket = socketname
PrintStream Pout = new PrintStream (out);
Pout.Print (" ");
out.flush → flush unsent bytes (to the output stream before closing the socket).
```

Note → setSoTimeout() → enables or disable the SO_TIMEOut option with the give timeout value in milliseconds. The timeout value should be greater than zero o.w, it'll throw an error.

Chapter 7

Threads review

- thread is a light weight process
- The idea is to achieve parallelism by dividing a process into multiple threads
- used to increase Performance of the applications
- Threads share code and data/file within a process
- each thread has its own stack, PC, registers.

- Concurrency refers to trying to do multiple things at once while Parallelism refers to doing many things at once.
- On a single core → concurrency is possible by switching among the threads. On a certain moment, one thread is executed.
- In Parallel processing, program instructions (Threads) are divided among multiple processors (cores), with the goal of executing the same program in less time.

- java.lang.Runnable, Interface
- java.lang.Thread

} → used to write multi-threaded Programs
↳ multitasking

- The problem with multi-threaded program is that there is more than one process that can modify the same variables.
- If there is a variable that we dont want to modify except from one program, then it is declared private or protected in that program.

- Process switching is a type of context switching where we switch one process with another. It involves switching of all process resources with those needed by a new process.

# of connections / size of the queue in single thread → 50 in linux by default, even if we change it.

. Thread class provide constructors and methods to create and perform operations on a thread. it extends object and implements Runnable interface.

. Client takes the ip address and port number of the server

. The try with resources statement :- allows us to declare resources to be used in try block with the assurance that the resources will be closed after the execution of that block.
  . the resource has to be both declared and initialized inside the try.

. Web server benchmarking is process of estimating a web server performance in order to find if the server can serve sufficiently high workload.

The performance is usually measured in terms of :-
- Number of requests that can be served per second
- Latency response time in milliseconds for each new connection.
- Throughput in bytes per second.

↑ requests ↑ CPU utilization
in multithreaded servers ↑ CPU utilization (thread for each request) ↓ time for each request

Good Luck!