UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

# COS110 - Program design: Introduction

## Assignment 4 Specifications

Due Date: 26-11-2021 at 23:50

Total Marks: 120

# Contents

# 1 General instructions

- This assignment should be completed individually, no group effort is allowed.
- Be ready to upload your assignment well before the deadline, as **no extension will be granted.**
- If your code does not compile, you will be awarded a mark of 0. The output of your program will be primarily considered for marks, although internal structure may also be tested (eg. the presence of certain functions or classes).
- Read the entire assignment thoroughly before you start coding.
- **To ensure that you did not plagiarize, your code will be inspected with the help of dedicated software.**
- Note that plagiarism is considered a very serious offence. Plagiarism will not be tolerated, and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at http://www.ais.up.ac.za/plagiarism/index.htm.
- Unless otherwise stated, the usage of C++11 or additional libraries outside of those indicated in the assignment, will not be allowed. Some of the appropriate files that you have submit will be overwritten during marking to ensure compliance to these requirements **Please ensure you use C++98**

# 2 Overview

For this assignment you will implement a simulated network based on a simplified version of the ISO OSI model. Upon successful completion of this assignment, you would have implemented the following:

- Data structures
  - Cyclically linked list
  - Doubly linked list
- Design Pattern
  - Layered architectural design pattern
- Programming technique
  - Recursion
- Object Orientated Programming principles
  - Inheritance
  - Polymorphism

You would also have formed a basic understanding of how data traverses the ISO OSI model as well as a basic understanding of how data are routed through a simplistic circular network topography.

# 3 Background

## 3.1 ISO OSI model

### 3.1.1 Concept

The OSI (Open System Interconnected) model is a conceptual model proposed by the ISO (International Organization for Standardization) in 1984 to standardize the communications functions of telecommunication or computing systems. This model uses the Layered architectural design pattern as a bases for the above-mentioned standardization. This is a very formal definition. More informally the ISO OSI model is the reference model stipulating how communication in a network will be accomplished. We use the term reference model as there has never been a network that fully satisfies the model's requirements. The ISO OSI model is comprised of 7 layers and will be discussed below. Due to the architectural design pattern used each layer in the ISO OSI model is only aware of the layer above and below it. Each layer is also logically aware of its peer layer (A peer layer is the equivalent layer on another node in the network).

### 3.1.2 Layers

As stated above the ISO OSI model has 7 layers:

- Layer 7: Application Layer
  This is the layer that interacts with the web application or the application that will utilize the network. This layer is responsible for the forming of a request and the displaying of a response from data sent/received over the network.

- Layer 6: Presentation Layer
  This is the layer that changes the data received from the Application layer into the correct format. This layer is thus responsible for the translation of the data from one character standard to another, encryption, and decryption of the data and lastly the compression of the data.

- Layer 5: Session Layer
  This is the layer that deals with the security of the connection. This layer is thus responsible for establishing a connecting with another node in the network, maintaining this connection/session as well as any authentication. (Please note for this assignment you will not be implementing this layer)

- Layer 4: Transport Layer
  This is the layer that deals with end-to-end delivery of a message passed through the network. This layer also deals with the communication of acceptance or rejection of a message. Please note in this case rejection implies an incomplete message has been received.

- Layer 3: Network Layer
  This is the layer that is responsible with the transmission of data from origin node to the destination node and back. This layer is also responsible for routing messages not intended for this node to the next node in the network. This layer has the added responsibility of calculating the logical IP address of the origin and the destination nodes.

- Layer 2: Data-link Layer
  This is the layer that ensures that data is transferred to the next node in the network over the physical interface in an error free method. This layer usually adds a character set to the start and end of a message to indicate the start and end point of a message.

- Layer 1: Physical Layer
  This is the layer that is physically connected to the network. This layer is responsible for transmitting the message over the network in binary representation.

*Please note that the above mentioned is a simplified description of the layers of the ISO OSI model. You will learn more about this model in COS332*

#### 3.1.3 Additional Resources

For more information please see the following videos by Prof Martin Olivier:
https://www.youtube.com/watch?v=J9Cn-B6rt-I
https://www.youtube.com/watch?v=FYPhTfF4zlQ

### 3.2 Layered architectural design pattern

#### 3.2.1 Concept

The layered architectural design pattern is used when the system can be divided into separate layers. These layers pass data to the next layer and receives data from the previous layer. Layers are thus seen as separate and only know of the next and previous layers.

### 3.3 Data structures

#### 3.3.1 Doubly Linked List

From theory you should know what a doubly linked list is but in short: A doubly linked list is the same as a normal linked list where a node in the link list points to the next and previous node.

#### 3.3.2 Circular Linked List

From theory you should know what a circular linked list is but in short: A circular linked list is a link list where the final node in the list next point to the first node in the list effectively forming a circle.

# 4 Your Tasks

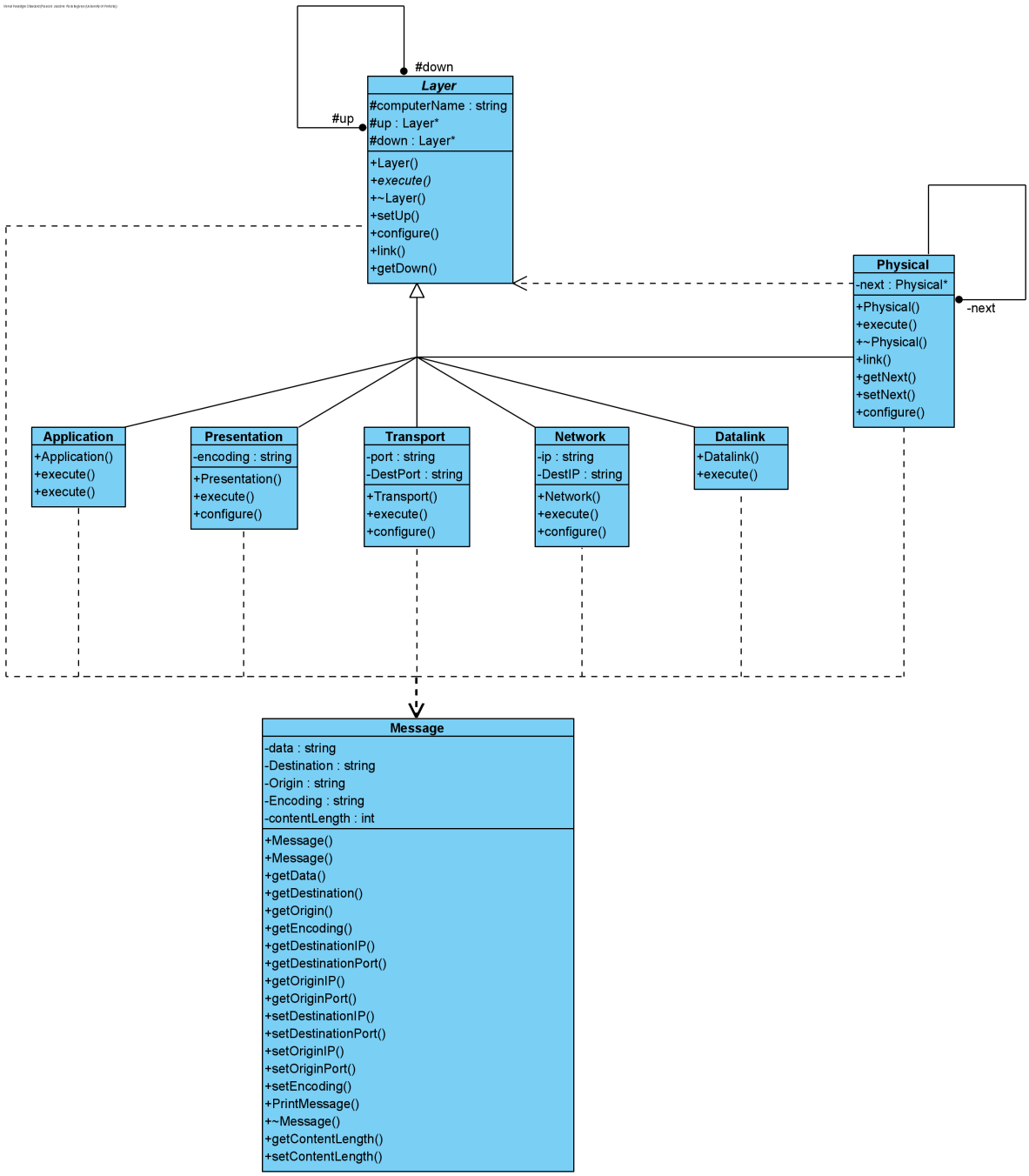


Figure 1: Class diagram without config struct

Please note that the term **network stack** refers to the **doubly linked list**.

## 4.1 Task 1: 20 Marks

### 4.1.1 Intro

For this task you are required to implement the message class. This is the class that will contain all the information sent between computers in the network.

### 4.1.2 Message.h

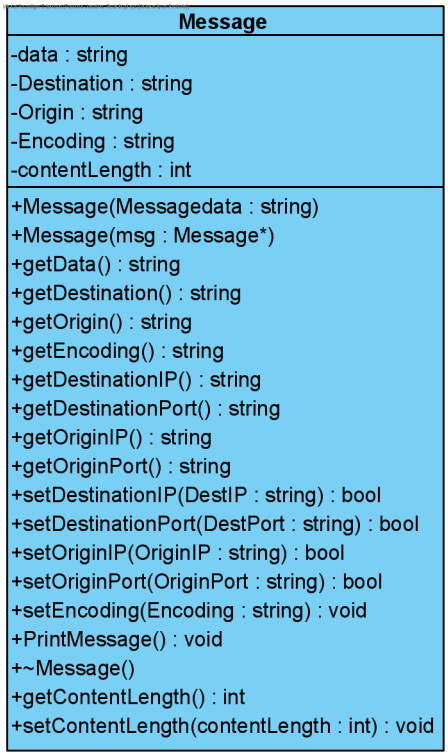


Figure 2: Message.h

**Please see below for exact function signatures**

- Private variables/functions:
  - data: string
    - This holds the data the message will be transporting
  - Destination: string
    - This holds the destination the message will be sent to in an IPv4:Port format.
    - The format of the destination is as follows:
      A.B.C.D:P where
      - A-D are numbers in between 0 and 255 (inclusive)
      - P is the port number between 0 and 65535 (inclusive)
    - Example of a valid Destination: 127.0.0.1:8800
  - Origin: string
    - This holds the origin the message was sent from in an IPv4:Port format
    - The format of the origin is as follows:
      A.B.C.D:P where
      - A-D are numbers in between 0 and 255 (inclusive)
      - P is the port number between 0 and 65535 (inclusive)
    - Example of a valid Origin: 127.0.0.1:8800

  - Encoding: string
    - This is the encoding the message uses
  - contentLength: int
    - This specifies the length of the data. Please note that contentLength and the actual length of the data can be different.

- Public variables/functions:
  - Message(Messagedata: string)
    - This is the constructor for Message. It takes in a string parameter which contains the data of the message.
    - The content length property should be initialized to the length of the passed in string.
    - Encoding should be initialized to an empty string.
    - Student may choose how they would like to initialize other members (note **do not** initialize destination/origin with valid data) .
  - Message(msg: Message*)
    - This is the copy constructor for the message class and should create a copy of the passed in message object. Only the **message** of the passed in message should be copied.
  - getData(): string
    - This returns the data of the message.
  - getDestination(): string
    - This function returns the destination address in its complete form (IPv4:Port).
    - If the destination has not been set or is incomplete the function should return the following:
      **ERROR: INCOMPLETE ADDRESS**
    - Example of a complete address that the function should return:
      127.0.0.1:5555
  - getOrigin(): string
    - This function returns the origin address in its complete form (IPv4:Port).
    - If the origin has not been set or is incomplete the function should return the following:
      **ERROR: INCOMPLETE ADDRESS**
    - Example of a complete address that the function should return:
      127.0.0.1:5555
  - getEncoding(): string
    - Returns the encoding of the message.
    - If the encoding has not been set a empty string should be returned.

- getDestinationIP(): string
  - This function returns the **IP** portion of the destination
  - If the IP portion has not been set then the following should be returned:
    **ERROR: IPV4 NOT SET**
  - Example of a possible return value:
    127.0.0.1
- getDestinationPort(): string
  - This function returns the **port** portion of the destination
  - If the port portion has not been set then the following should be returned:
    **ERROR: PORT NOT SET**
  - Example of a possible return value:
    55555
- getOriginIP(): string
  - This function returns the **IP** portion of the origin
  - If the IP portion has not been set then the following should be returned:
    **ERROR: IPV4 NOT SET**
  - Example of a possible return value:
    127.0.0.1
- getOriginPort(): string
  - This function returns the **port** portion of the origin
  - If the port portion has not been set then the following should be returned:
    **ERROR: PORT NOT SET**
  - Example of a possible return value:
    55555
- setDestinationIP(DestIP: string): bool
  - This function sets the **IP** portion of the destination.
  - It is up to the student to choose which function will add the ":" at the appropriate place.
  - This function should also verify that the IP address is in the correct format:
    - Let A,B,C,D be a integer number in the range [0,255] (inclusive)
    - Correct format is thus:
      A.B.C.D
    - Example of a valid IP: 127.0.0.255
  - If an invalid IP was passed as parameter to the function, the function should **not** set the IP and return false
  - If the destination IP is **already** set the function should **not** set the new IP and return false
  - If the destination IP has **not** been set and the passed in IP is **valid** then the function should set the destination IP and return true

- setDestinationPort(DestPort: string): bool
  - This function sets the **port** portion of the destination.
  - It is up to the student to choose which function will add the ":" at the appropriate place.
  - This function should also verify that the port is valid.
    - A valid port is a number in the range [0,65535] (inclusive)
  - If the port is invalid the function should **not** set the destination port and return false
  - If the destination port has been already set the function should **not** set the destination port and return false
  - If the port is **valid** and the destination port has **not** been set then the function should set the port and return true
- setOriginIP(OriginIP: string): bool
  - This function sets the **IP** portion of the origin.
  - It is up to the student to choose which function will add the ":" at the appropriate place.
  - This function should also verify that the IP address is in the correct format:
    - Let A,B,C,D be an integer number in the range [0,255] (inclusive)
    - Correct format is thus:
      A.B.C.D
    - Example of a valid IP: 127.0.0.255
  - If an invalid IP was passed as parameter to the function, the function should **not** set the IP and return false
  - If the origin IP is **already** set the function should **not** set the new IP and return false
  - If the origin IP has **not** been set and the passed in IP is **valid** then the function should set the origin IP and return true
- setOriginPort(OriginPort: string): bool
  - This function sets the **port** portion of the origin.
  - It is up to the student to choose which function will add the ":" at the appropriate place.
  - This function should also verify that the port is valid.
    - A valid port is a number in the range [0,65535] (inclusive)
  - If the port is invalid the function should **not** set the origin port and return false
  - If the origin port has been already set the function should **not** set the origin port and return false
  - If the port is **valid** and the origin port has **not** been set then the function should set the port and return true
- setEncoding(Encoding: string): void
  - This function sets the encoding of a message
  - The encoding **can** be overwritten..

- PrintMessage(): void
  - This function should print out the information of the message as follows on a new line:
    - Data: data of message
    - Destination: contains the same information that getDestination() returns
    - Destination IPV4: contains the same information that getDestinationIP() returns
    - Destination Port: contains the same information that getDestinationPort() returns
    - Origin: contains the same information that getOrigin() returns
    - Origin IPV4: contains the same information that getOriginIP() returns
    - Origin Port: contains the same information that getOriginPort() returns
    - Encoding: encoding of the message (If their is no encoding it should still print Encoding: followed by a **single** space)
  - Example 1:
    Data: Example 1
    Destination: 127.0.0.2:500
    Destination IPV4: 127.0.0.2
    Destination Port: 500
    Origin: 192.168.0.2:400
    Origin IPV4: 192.168.0.2
    Origin Port: 400
    Encoding: UTF-8

  - Example 2:
    Data: Example 2
    Destination: ERROR: INCOMPLETE ADDRESS
    Destination IPV4: ERROR: IPV4 NOT SET
    Destination Port: ERROR: PORT NOT SET
    Origin: ERROR: INCOMPLETE ADDRESS
    Origin IPV4: ERROR: IPV4 NOT SET
    Origin Port: ERROR: PORT NOT SET
    Encoding:

- ~Message()
  - This is the destructor for the message class
  - This function should print out the following ending with a newline:
    Deleted message containing: $X$
    Where $X$ is the data of the message
  - Example:
    Deleted message containing: Data for message 1
- getContentLength(): int
  - This function returns the contentLength property
- setContentLength(contentLength: int): void
  - This function should update the contentLength of the message.
  - This function should **overwrite** the contentLength property

## 4.2 Task 2: 100 Marks

### 4.2.1 Intro

For this task you will be implementing the bulk of the assignment. In this task you will implement the network stack as a doubly linked list. You will also implement the network as a cyclical linked list. This task will also require inheritance, polymorphism, and virtualization.

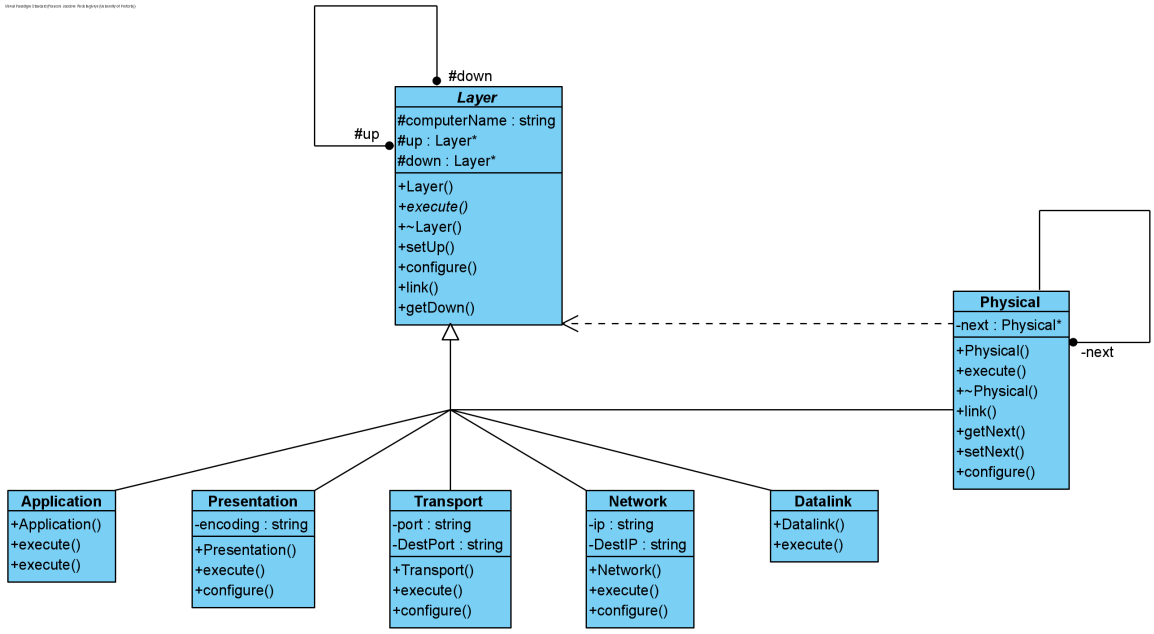


Figure 3: Task 2

**Please note the UML diagram only shows the basic structure. See the description below each diagram for the exact function signatures**

### 4.2.2 Layer.h

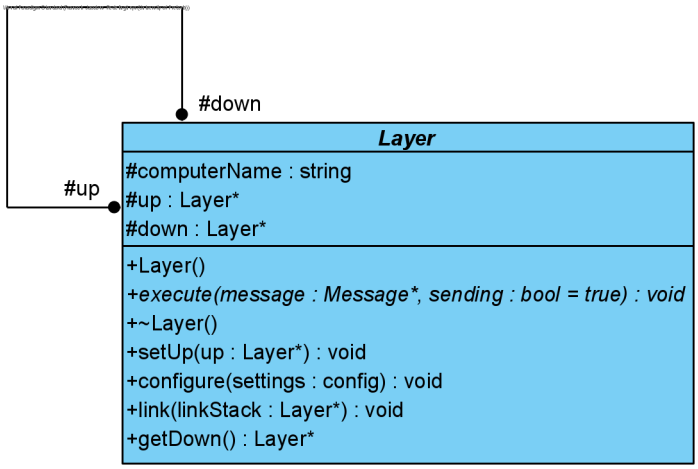


Figure 4: Layer.h

- Protected variables/functions:
  - computerName: string
    - This is the name of the computer
  - up: Layer*
    - This is a pointer to the layer "above" the current layer
  - down: Layer*
    - This is a pointer to the layer "below" the current layer
- Public variables/functions:
  - Layer()
    - This is the constructor for the layer class
    - The constructor should initialize the up and down variables to NULL
  - pure virtual execute(message: Message*, sending = true : Bool): void
    - This is a pure virtual function.
    - The function takes in a pointer to a Message object.
    - The sending parameter should have a default value of true
  - virtual ~Layer()
    - This is a virtual destructor for the layer class
    - The destructor should also deallocate the down member.
    - Remember to set the layer below to NULL after deleting it
  - setUp(up: Layer*): void
    - This function sets the up property to the passed in parameter.
    - **Do not** make a deep copy of the passed in parameter

- virtual configure(settings: config): void
  - This is a virtual function which takes in a struct of type config.
  - This function will be used to configure the stack recursively.
  - This function sets the computerName property with the computerName property of the passed struct
  - This function should also call the configure function of the down property and pass settings as parameter
- virtual link(linkStack: Layer*): void
  - This is a virtual function used to insert stacks into the network.(More information on this in the Physical layer)
  - All this function should do in the Layer.cpp is call the link function of the down property and pass in the linkStack parameter
- getDown(): Layer*
  - This returns the down property

### 4.2.3 Config struct

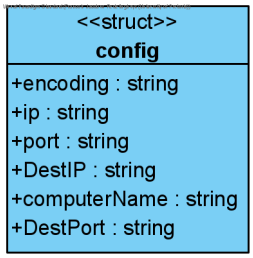


Figure 5: Config struct

Please declare the config struct in the Layer.h file This struct will be used to configure the network stack

### 4.2.4 Application.h

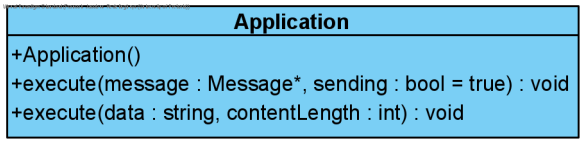


Figure 6: Application.h

- The application class inherits publicly from Layer class
- Public variables/functions
  - Application()
    - This constructor initializes the down property with a Presentation class object
    - Also sets the down's up property to the current object
  - execute(message: Message*, sending: bool): void
    - This function is used to pass messages up and down the doubly linked list.
    - If sending is true down's execute should be called. The passed in message and sending should be passed to the function.
    - Else your program should output the following ending with a new line:
      Computer [computerName] Received message: [message data]
    - Example:
      **Computer Comp 3 Received message: This is message's data**
  - execute(data: String, contentLength: int): void
    - This function should create a new message object and pass in the data parameter.
    - This function should also set the message's content length to the passed in parameter.
    - This function should then call down's execute function passing the newly created message and true as parameters.

### 4.2.5 Presentation.h

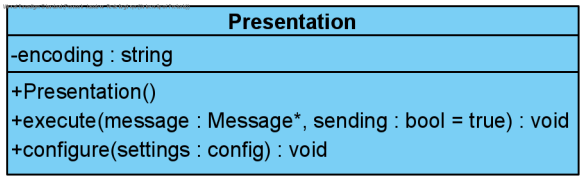


Figure 7: Presentation.h

- The presentation class inherits publicly from Layer class
- Private variables/functions
  - encoding: String
    - This is the encoding that the network stack will use
- Public variables/functions
  - Presentation()
    - This is the constructor for this class
    - The constructor should initialize the down property with a new Transport object.
    - The constructor should also set the down's up to the current object
  - execute(message: Message*, sending=true: bool): void
    - Please note the sending has a default value of true
    - If sending is true this function should set the passed in message's encoding with the encoding property and call down's execute function passing in sending and message as parameters.
    - If sending is not true the function should do the following:
      - If the encoding of the passed message is equal to the encoding property, then print out the following ending with a new line, and call up's execute and pass the message and false as parameters:
        **Message at Presentation layer on computer: [computer name]**
      - Example:
        Message at Presentation layer on computer: Desktop1
      - If the encoding of the passed message is not equal to the encoding property, then print out the following and call the PrintMessage function of the message:
        **Encoding type mismatch on computer: [computer name] Expected: [encoding property]**

· Example:
Encoding type mismatch on computer: Comp 1 Expected: UTF-8
Data: Wrong data encoding
Destination: 127.0.0.1:300
Destination IPV4: 127.0.0.1
Destination Port: 300
Origin: 192.168.0.1:400
Origin IPV4: 192.168.0.1
Origin Port: 400
Encoding: ACSII

- configure(settings: config): void
  - ∗ This function will set the encoding property to the settings' encoding property.
  - ∗ It will then call Layer's configure function.

### 4.2.6 Transport.h

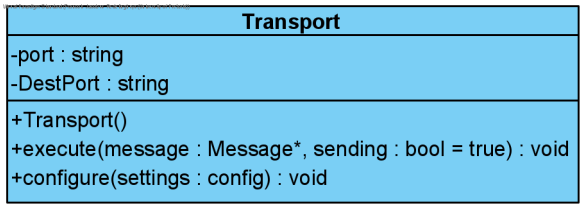


Figure 8: Transport.h

- Transport class publicly inherits from Layer class
- Private variables/functions:
  - port: String
    - ∗ This is the port of the current network stack
  - DestPort: String
    - ∗ This is the port of the destination of all messages sent using current stack configuration.
- Public variables/functions:
  - Transport()
    - ∗ This is the constructor for the Transport class
    - ∗ The constructor should initialize the down property with a Network object
    - ∗ The constructor should also set the up property of the new Network object to itself

- execute(message: Message*, sending=true: bool): void
  - If sending is true then the function should do the following:
    - The function should set the passed in message's origin port by calling the appropriate message function with the appropriate member variable
    - If the above function returns false the function should print out the following with a new line at the end and exit the function:
      **ERROR OCCURRED WITH ORIGIN PORT**
    - The function should then set the passed message's destination port by calling the appropriate message function with the appropriate member variables
    - If the above function returns false the function should print out the following with a new line at the end and exit the function:
      **ERROR OCCURRED WITH DESTINATION PORT**
    - The function should then call down's execute and pass in message
  - Else if sending is false then the function should do the following:
    - If the message's destination port is not equal to the network stack's port then print out the following followed by message's PrintMessage function:
      **Message sent to wrong port**
    - Example:
      Message sent to wrong port
      Data: Wrong Port Message
      Destination: 127.0.0.2:500
      Destination IPV4: 127.0.0.2
      Destination Port: 500
      Origin: 192.168.0.2:400
      Origin IPV4: 192.168.0.2
      Origin Port: 400
      Encoding: UTF-8

    - If the message's actual data length is not equal to the message's content length then print out the following followed by the message's PrintMessage function:
      **Content Length error**
    - Example:
      Content Length error
      Data: Wrong content length
      Destination: 192.168.0.1:400
      Destination IPV4: 192.168.0.1
      Destination Port: 400
      Origin: 127.0.0.1:300
      Origin IPV4: 127.0.0.1
      Origin Port: 300
      Encoding: UTF-8

· If the message's destination port is equal to the current network stack's port property and the content length of the message is equal to the length of the message's data, then the function should print out the following and call up's execute function. It should pass the message and false as parameters:
**Message at Transport layer**

* configure(settings: config): void
  · This function should set the port and DestPort properties with setting's port and DestPort respectively.
  · This function should then call the Layer's configure function passing in settings

### 4.2.7 Network.h

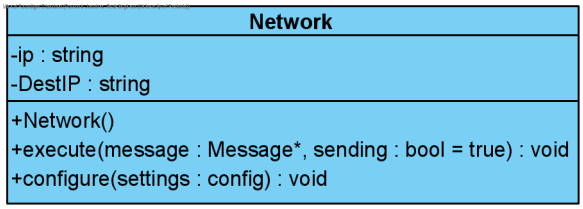


Figure 9: Network.h

- Network class publicly inherits from Layer class
- Private variables/functions
  - ip: string
    * This stores the ip address of the current network stack
  - DestIP: string
    * This stores the destination ip for all messages sent using current network stack configuration
- Public variables/functions
  - Network()
    * This is the constructor for the Network class
    * The constructor should initialize the down property with a Datalink object
    * The constructor should also set the up property of the new Datalink object to itself

- execute(message: Message*, sending=true: bool): void
  - If sending is true then the function should do the following:
    - The function should set the passed in message's origin ip with the ip property by calling the appropriate message function .
    - If the function returns false it should print out the following follwed by a new line and exit the function:
      **ERROR OCCURRED WITH ORIGIN IP**
    - The function should then set the passed in message's destination ip with the DestIP property by calling the appropriate message function.
    - If the function returns false it should print out the following followed by a new line and exit the function:
      **ERROR OCCURRED WITH DESTINATION IP**
    - The function should then call down's execute function passing in message and true
  - If sending is false then the function should do the following:
    - If the message's destination IP is equal to the ip property, then the function should print out the following and call up's execute function passing message and false as parameters:
      **Message at Network layer**
    - If the message's origin IP is equal to the ip property then the function should print out the following, followed by a newline and then the message's PrintMessage function:
      **Message has been returned to sender without finding the destination**
    - Example:
      Message has been returned to sender without finding the destination
      Data: Unknown Destination
      Destination: 224.212.45.01:300
      Destination IPV4: 224.212.45.01
      Destination Port: 300
      Origin: 10.4.23.55:300
      Origin IPV4: 10.4.23.55
      Origin Port: 300
      Encoding: UTF-8

    - Else the function should print out the following ending on a new line and call down's execute function passing message and true as parameters:
      **Message at Network layer**

- configure(settings: config): void
  - This function should set the ip and DestIP with the settings' ip and DestIP respectively
  - The function should then call layer's configure function passing in settings

### 4.2.8 Datalink.h

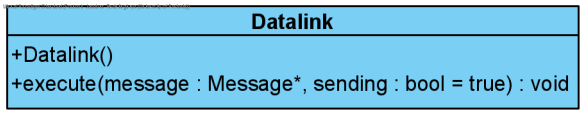


Figure 10: Datalink.h

- Datalink class publicly inherits from Layer class
- Public variables/functions:
  - Datalink()
    - This is the constructor for the Datalink class
    - The constructor should initialize the down property with a Physical object
    - The constructor should also set the up property of the new Physical object to itself
  - execute(message: Message*, sending=true: bool): void
    - If sending is true then the function should call down's execute function and pass message and true as parameters
    - Else the function should print out the following follwed by a newline and call up's execute function and pass message and false as arguments:
      **Message at Datalink layer**

### 4.2.9 Physical.h

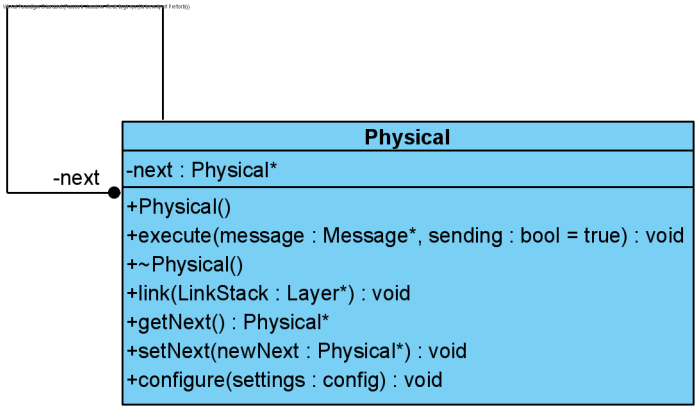


Figure 11: Physical.h

- Physical class publicly inherits from Layer class
- This layer will be form part of both the doubly linked list and the cyclically linked list.

- Private variables/functions:
  - next: Physical*
    - This stores the next node in the network (cyclically linked list)
- Public variables/functions:
  - Physical()
    - This is the constructor for the Physical class
    - The constructor should initialize the next property to null
  - execute(message: Message*, sending=true: bool): void
    - If sending is true the function should do the following:
      - If next is null the function should print out the following ending with a new line followed by the message's PrintMessage() function:
        **NOT CONNECTED TO A NETWORK**
      - If next is not null, the function should call next's execute and pass message and false as parameters.
    - If sending is false the function should print out the following ending with a new line followed by calling up's execute function passing message and false as parameters:
      **Message at Physical layer of computer: *computerName***
      where computerName is the name of the computer
  - ~Physical()
    - This is the destructor of the Physical class
    - The destructor should decouple the current physical object from the network (cyclically linked list).
  - link(LinkStack: Layer*): void
    - This function will insert a new node into the network (cyclically linked list) by setting the current's next to the passed LinkStack.
    - Note next is of type Physical so you need to find the bottom of LinkStack and cast it to a Physical object.
    - Remember the network is a cyclically linked list.
  - getNext(): Physical*
    - This function should return the next property
  - setNext(newNext: Physical*): void
    - This sets next to the passed in parameter
  - configure(settings: config): void
    - This function sets the computer name property to the settings' computer name property.
    - This function **does not call** the Layer configure function

# 5 Visual representation of a message traversal

For added clarification to send a message from C1 to C4 the following will have to occur

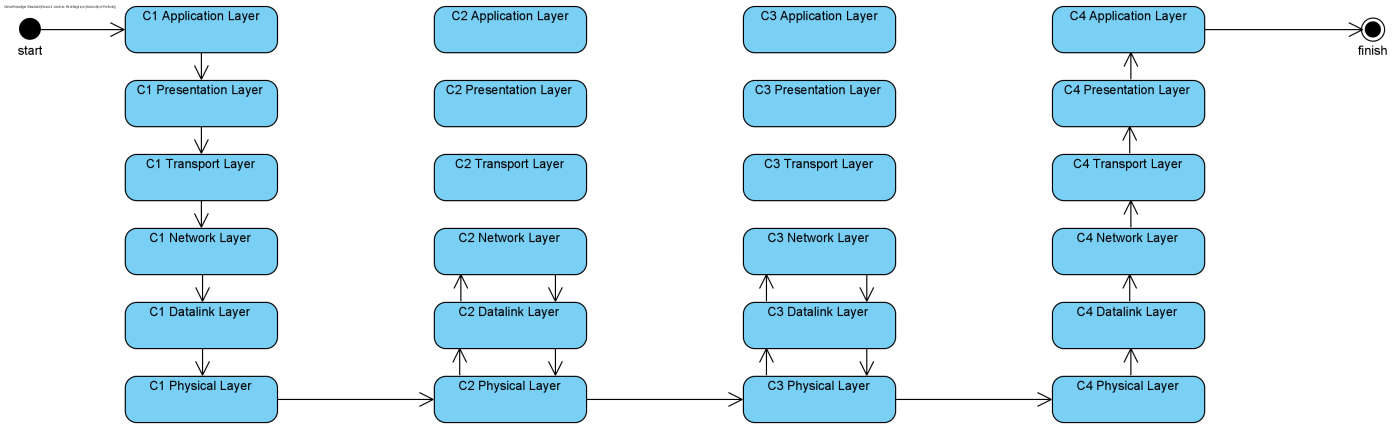


Figure 12: A visual representation of a message traversal

Please note C1 is linked with C2. C2 is linked with C3. C3 is lined with C4 and C4 is linked with C1 to form a cyclically linked list.

# 6 Implementation Details

- You must implement the functions in the header files exactly as stipulated in this specification. Failure to do so will result in compilation errors on fitch fork.
- You may only utilize the specified libraries. Failure to do so will result in compilation errors on fitch fork.
- Do not include using namespace std in the h files
- You may only use the following libraries:
  - String
  - StringStream (sstream)
  - Iostream
- Hint for this assignment:
  - Use pen and paper sketch transitions between nodes in the link list
  - Test your code thoroughly before submitting. **Do not use fitch fork as a debugger!**

# 7 Upload checklist

- Task 1:
  - Message.cpp
  - Message.h
- Task 2:
  - Message.cpp
  - Message.h
  - Layer.cpp
  - Layer.h
    - Including the config struct
  - Application.h
  - Application.cpp
  - Presentation.h
  - Presentation.cpp
  - Transport.h
  - Transport.cpp
  - Network.h
  - Network.cpp
  - Datalink.h
  - Datalink.cpp
  - Physical.h
  - Physical.cpp

Good luck!