CSCI 3100 Software Engineering

Assignment 4

Jonathan Lam Pui Hong (1155093597)

Question 1

```
module EmailNotification
exports
    procedure sendEmail(subject: in string, body: in string,
        receivers: in array[1..5] of string)
        This procedure sends an email to all `receivers`.
end EmailNotification

module GitHubRepository
exports
    type GitHubRepositoryType: ?
        This is the abstract data type to be exported.
    procedure pullCode(gitHubURL: in string)
        This procedure pulls code from the `gitHubURL`.
    function genTextReport(repository: in GitHubRepositoryType): string
        This procedure generates a text report of a given repository.
end GitHubRepository

module CheckValidTeammate
uses GroupDB.studentInDB
exports
    function checkValidTeammate(studentID: in string): boolean
        This function checks whether the student has been included in other
        teams and returns a boolean of the result.
implementation
    This module checks if the `studentID` is present in `GroupDB` and returns
    a boolean of the result.
end CheckValidTeammate

module GenTeamID
uses GroupDB.teamIDInDB
exports
    function generateID(): teamID;
        This function generates and returns a team ID which is not taken yet.
implementation
    This module generates a team ID and checks if it is present in the
    database. If not, it will be returned; if so, repeat the process.
end GenTeamID
```

module GroupDB
exports
    function getEmails(teamID: in integer): array[1..5] of string
        This function returns emails of the team members in `teamID`.
    function teamIDInDB(teamID: in integer): boolean
        This function checks whether the team with `teamID` already exists in database.
    function studentInDB(studName: in string, studID: in string): boolean
        This function checks whether the student with `studName` and `studID` *is in the database.*
    function submittedURL(teamID: in integer): boolean
        This function checks whether a team has submitted the GitHub URL.
    function submittedDoc(teamID: in integer, docType: in integer): boolean
        This function checks whether a team has submitted a document of `docType`.
    procedure modifyDB(data: in ?)
        This process allows modification (save or overwrites) data in the database.
implementation
    This module accesses the database and provides different functionalities for ease of use.
end GroupDB

module ProjectManagment
uses
    EmailNotification, GitHubRepository
exports
    procedure registerTeam(studentNames: in array[1..5] of string,
        studentIDs: in array[1..5] of string,
        projectName: in string, teamID: out teamID)
    procedure submitGitHubURL(teamID: in integer, gitHubURL: in string)
    procedure submitProjectDoc(teamID: in integer, docType: in integer,
        file: in file of string)
    procedure pullCode(teamID: in integer, phaseType: in integer)
    procedure releaseMark(teamID: in integer, phaseType: in integer,
        marks: in string, comments: in string)
    There are fixed range for three types of input.
    teamID: (integer)
        When initiated by client, this should be a correct format of teamID.
    docType: (integer)
        0: indicating initial design document, or
        1: indicating final report.
    phaseType: (integer)
        0: indicating initial code, or
        1: indicating completed code, or
        2: indicating commented code.

implementation

is composed of GenTeamID, CheckValidTeammate, GroupDB

`registerName` first uses CheckValidTeammate to check if any of the `studentNames` or `studentIDs` are already present in `GroupDB`. If not, it uses `GenTeamID` to generate a unique `teamID`. It then sends an email notification with either team information (success) or reason of failure as the email body.

`submitGitHubURL` accepts the URL from a teammate and saves (or overwrites if previous submission is present) the URL into the `GroupDB`. Upon completion, this module sends emails to students indicating success.

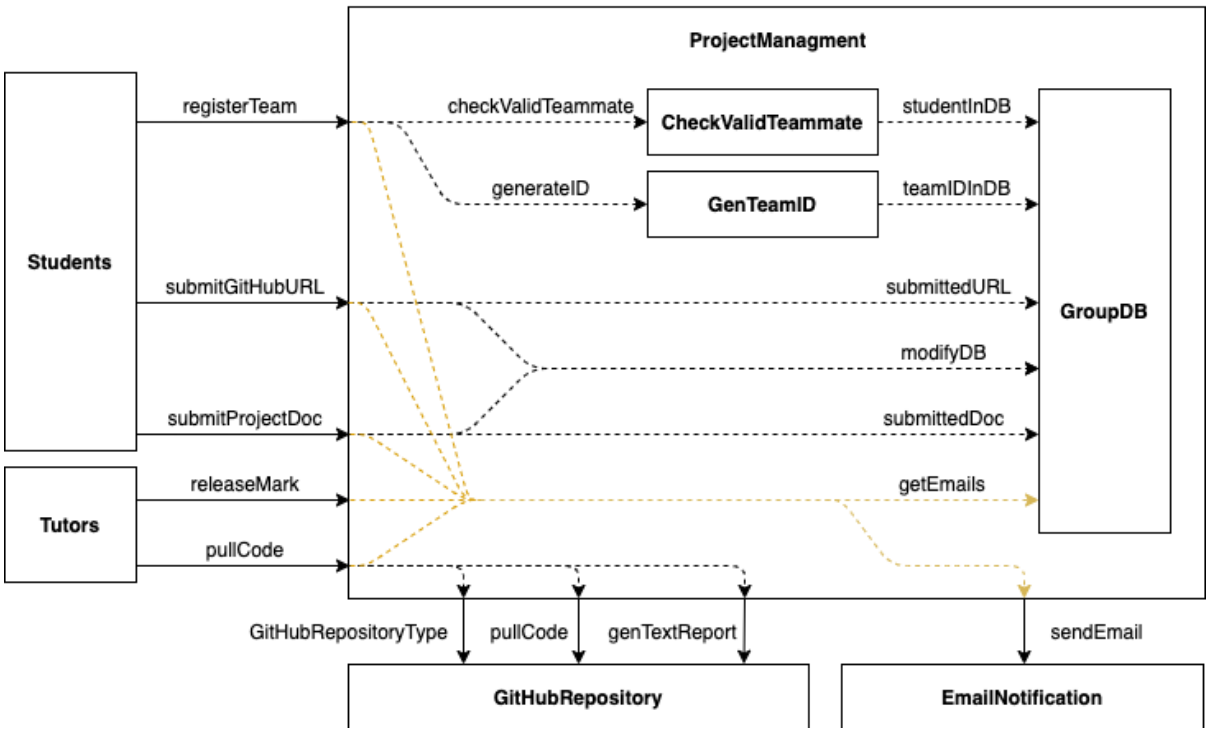
`submitProjectDoc` accepts the file from a teammate and saves (or overwrites if previous submission is present) the file into `GroupDB`. Upon completion, this module sends emails to students indicating success.

`pullCode` fetches the team's respective GitHub URL and members' emails from `GroupDB` using `teamID`, then uses `GitHubRepository` to pull the code of the team, generates a status report of whether the pull is successful, and uses `EmailNotification` to send emails to students with the status report as a body.

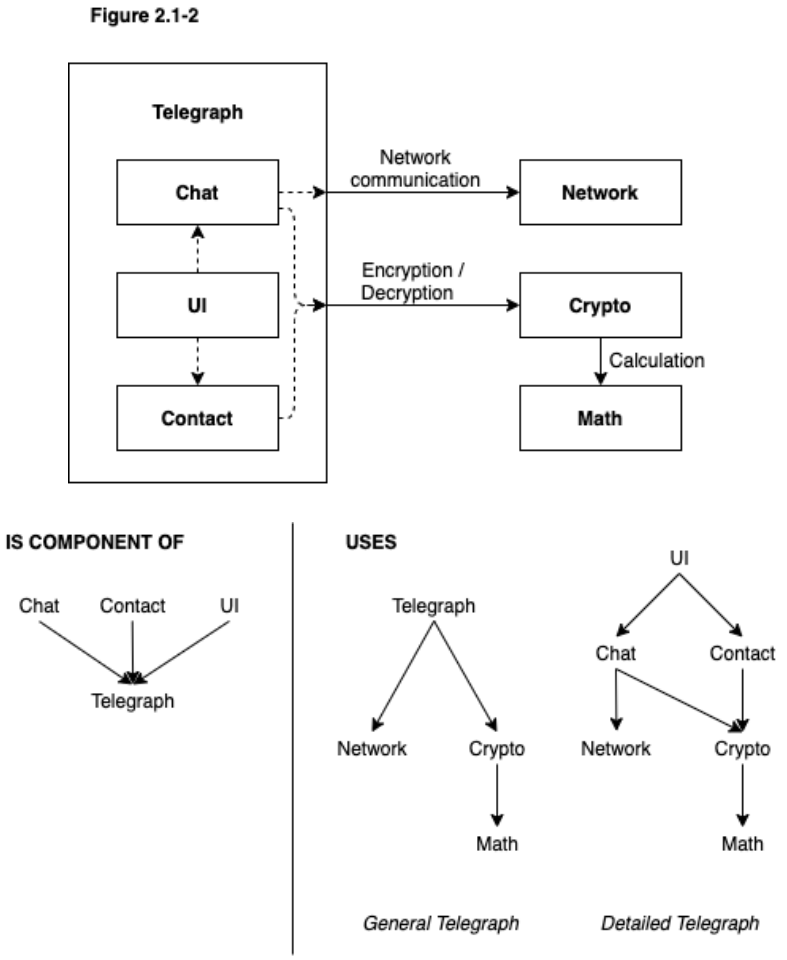
`releaseMark fetches team members' emails from `GroupDB` using `teamID` and then sends emails to students with `phaseType`, `marks`, and `comments` as the email body.

end ProjectManagment

Figure 1.3



Question 2



Module	d-in (direct)	d-in (indirect)	d-out (direct)	d-out (indirect)	Stability
Telegraph	-	-	Network, Crypto	Math	0
Chat	UI	-	Network, Crypto	Math	$\frac{1}{4}$
Contact	UI	-	Crypto	Math	$\frac{1}{3}$
UI	-	-	Chat, Contact	Network, Crypto, Math	0
Network	Chat	UI	-	-	1
Crypto	Chat. Contact	UI	Math	-	$\frac{3}{4}$
Math	Crypto	Chat, Contact, UI	-	-	1

Stability, S, measures the reusability of the module. The more the module is dependent by other modules, the higher the stability index is, and the more reusable it is. Since a module of high stability is used by many modules, it should be stable, well-defined, less error, and able to accommodate different uses. It is then said to be unlikely to change, since small changes will cause the behaviour of its incoming dependencies change as well. On the other hand, modules with low stability are likely to change as its changes will not affect other modules as much.

> QUESTION 3.1

Adjacency list

1	5	6	
2	3		
3	2	4	6
4	2		
5	1	2	4
6	1	4	5

Adjacency matrix

	1	2	3	4	5	6
1	0	0	0	0	1	1
2	0	0	1	0	0	0
3	0	1	0	1	0	1
4	0	1	0	0	0	0
5	1	1	0	1	0	0
6	1	0	0	1	1	0

> QUESTION 3.2

Initial

```
let L be the adjacency list, where L[n] is the nodes to which n connects.

while L is not empty loop
    n = L.pop()
    # Transverse the nodes connecting to n and their subsequent children
    # (i.e. L[n] and L[n]'s children) and checks if they create a loop.
    # If so, haut and return false.
end loop
stop and return true
```

Refinement 1

```
let L  be the adjacency list, where L[n] is the nodes to which n connects.
let Lt be a list of terminal nodes.

while L is not empty loop
    n = L.pop()
    Traverse(n)
end loop
stop and return true

function Traverse(n)
    for each node m in L[n] loop
        # Check if there exists an m that has been visited in this cycle.
        # If so, haut and return false.
    end loop
    Lt.push(n)
end function
```

Refinement 2

```
let L  be the adjacency list, where L[n] is the nodes to which n connects.
let Lt be a list of terminal nodes.

while L is not empty loop
    n = L.pop()
    let Lv be a list of visited nodes (in this cycle).
    Traverse(n, Lv)
end loop
stop and return true

function Traverse(n, Lv)
    Lv.push(n)
    for each node m in L[n] loop
        if m in Lt then
            continue
        else if m in Lv then
            stop and return false
        else
            Traverse(m, Lv)
        end if
    end loop
    L.pull(n)
    Lt.push(n)
end function
```

Note

L.Pull(n): n is pulled/removed from L in case n is invoked within the traversion and not by the while loop to avoid traversing repeated nodes.

Time Complexity

O(V+E) since every node and edge is traversed once only.

> QUESTION 4.1

```
# Distance of three sides
dap = math.sqrt((xa-xp)**2 + (ya-yp)**2)
dbp = math.sqrt((xb-xp)**2 + (yb-yp)**2)
dab = math.sqrt((xa-xb)**2 + (ya-yb)**2)

# Heron's formula
s = (dap + dbp + dab) / 2
area = math.sqrt(s * (s-a) * (s-b) * (s-c))

# Perpendicular distance from p to ab
d = 2 * area / dab
```

> QUESTION 4.2-4.4

```
def pDistance(p, a, b): # p, a, b are of type Point
    # return the distance from p to line ab

def farthestPoint(points): # points is a list of type Point
    maxIdx = 1
    maxDist = 0
    for idx in range(1, len(points) - 1):
        dist = pDistance(points[idx], points[0], points[-1])
        if dist > maxDist:
            maxIdx = idx
            maxDist = dist
    # returns the index and distance of the farthest point
    return maxIdx, maxDist

# Assume all points are selected at the beginning
def DouglasPeuker_R(points, threshold):
    if len(points) <= 1:
        return
    maxIdx, maxDist = farthestPoint(points)
    if maxDist >= threshold:
        DouglasPeuker_R(points[:maxIdx+1], threshold)
        DouglasPeuker_R(points[maxIdx:],   threshold)
    else:
        for p in points[1:-1]:
            p.selected = False
        return

# Assume all points are not selected at the beginning
def DouglasPeuker_NR(points, threshold):
    stack = Stack() # initialize a stack
    # push a tuple indicating the index of the starting and ending point
    stack.push((0, len(points)-1))
    while not not stack:
        e = stack.pop()
        if e[0] == e[1]:
            points[e[0]].selected = True
            points[e[1]].selected = True
        else:
            maxIdx, maxDist = farthestPoint(points[e[0]:e[1]+1])
            maxIdx += e[0]
            if maxDist >= threshold:
                stack.push((e[0], maxIdx))
                stack.push((maxIdx, e[1]))
            else:
                points[e[0]].selected = True
                points[e[1]].selected = True
```