



# פרויקט תכנות בסביבת אינטרנט


מגישים: מקסים בויאר, ארטיום זיננקו

מוגש עבור מר נתן דילברי-פברואר 2021



# חלק 1

# חלק 1 - הסבר



בחלק הראשון אנחנו ממשים מחלקה MyUUID ו 2 ממשקים Node ו HasUUID מחלקה זו מייצגת טיפוס חדש שמטרתו לקשר בין מחרוזת המייצגת טיפוס בעולם למזהה ייחודי המשוך לאובייקט מהסוג של הטיפוס. המחלקה לא ניתנת להרחבה מכילה פרטים אודות שם הטיפוס ומזהה ייחודי אשר ייקשר למחרוזת באמצעות הכלה של משתנה המחלקה מכילה פרטים אודות שם הטיפוס ומזהה ייחודי אשר ייקשר למחרוזת באמצעות הכלה של משתנה מסוג String בשם Key ומשתנה מסוג UUID בשם uuid מסוג data member-ה- UUID .

## הממשק HasUUID

ממשק זה מגדיר מתודה אחת בלבד בשם getUUID אשר לא מקבלת אף ארגומנט ומחזירה UUID

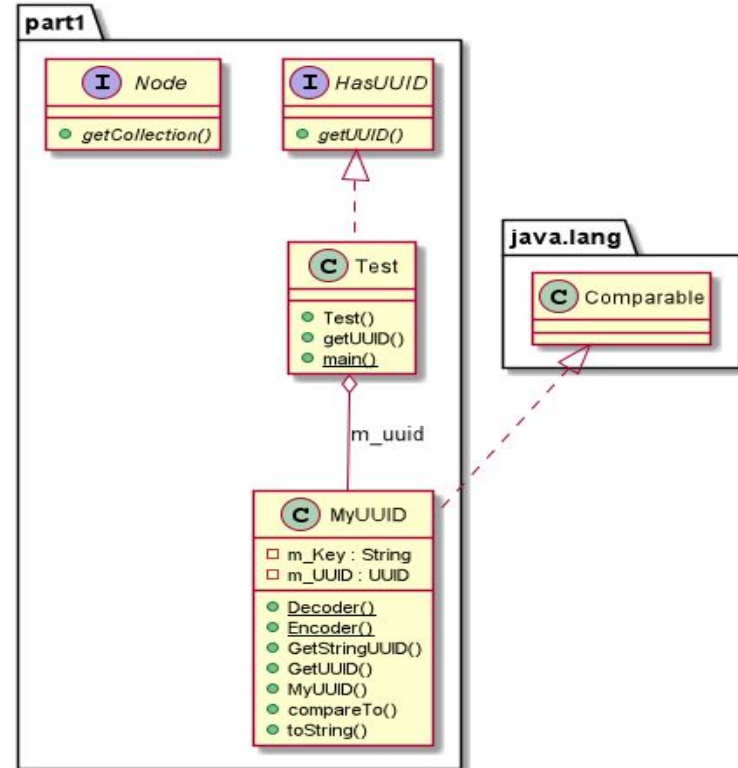
## הממשק Node

מתודה אחת בשם getCollection אשר מקבלת משתנה מסוג מחלקה הכולל מגבלה על סוג המחלקה אותה היא יכולה לקבל: מחלקה ממשת את הממשק HasUUID

# Class diagram - part 1



PART1's Class Diagram





# חלק 2

## חלק 2 - הסבר כללי

החלק הזה בפרויקט מורכב ממספר פקג'ים שכל אחד מהם אחראי ללוגיקה מסויימת

### TCP SERVER

צד שרת אשר עובד בת'רד ראשי אשר מאזין לבקשות ומטפל בלקוחות כל אחד בת'רד ייעודי ללקוח

### CLIENT

צד לקוח אחראי ללוגיקה של הלקוחות -מעביר בקשות לשרת על פי דרישת הלקוח

### COMPONENTS

מימושים של החלקים הראשיים כגון גרף (אשר עוטף מטריצה) וקודקוד

### ALGORITHMS

החלק אחראי לממש אלגוריתמים לפי הדרישה כאשר כל האלגוריתמים ממומשים בעזרת תורת הגרפים

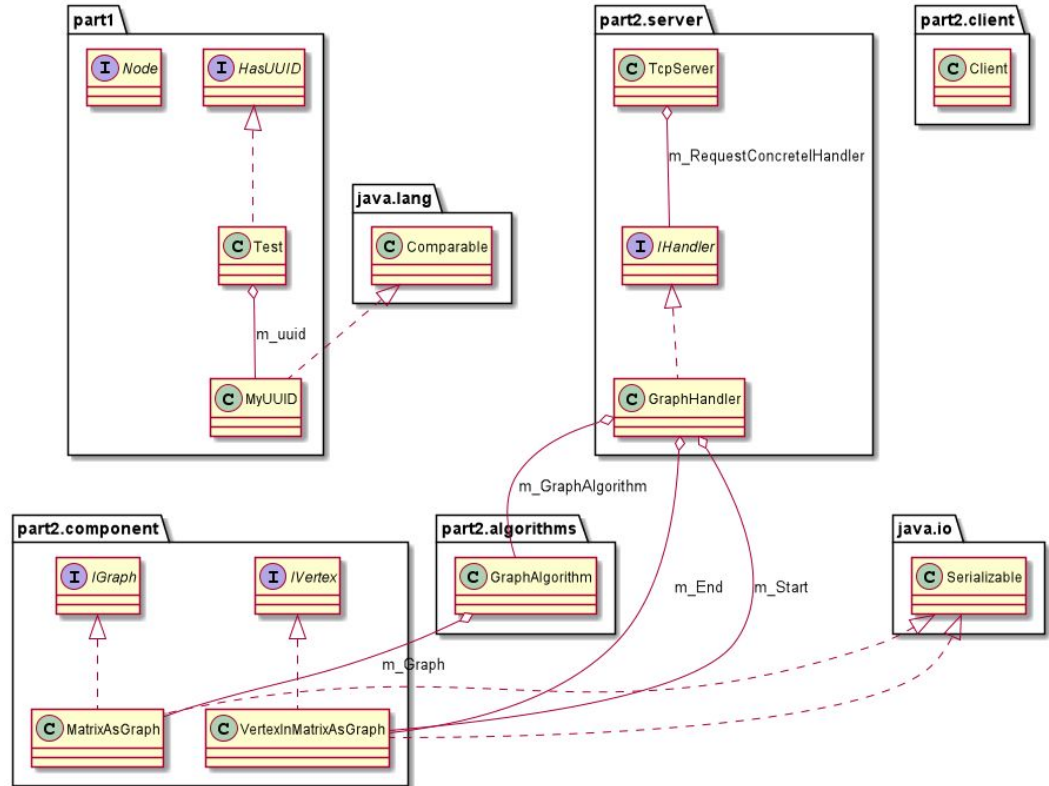
-מציאת רכיבי קשירות

-מציאת כל המסלולים מקודקוד לקודקוד

-מציאת צוללות תקינות על פי גיאומטריה

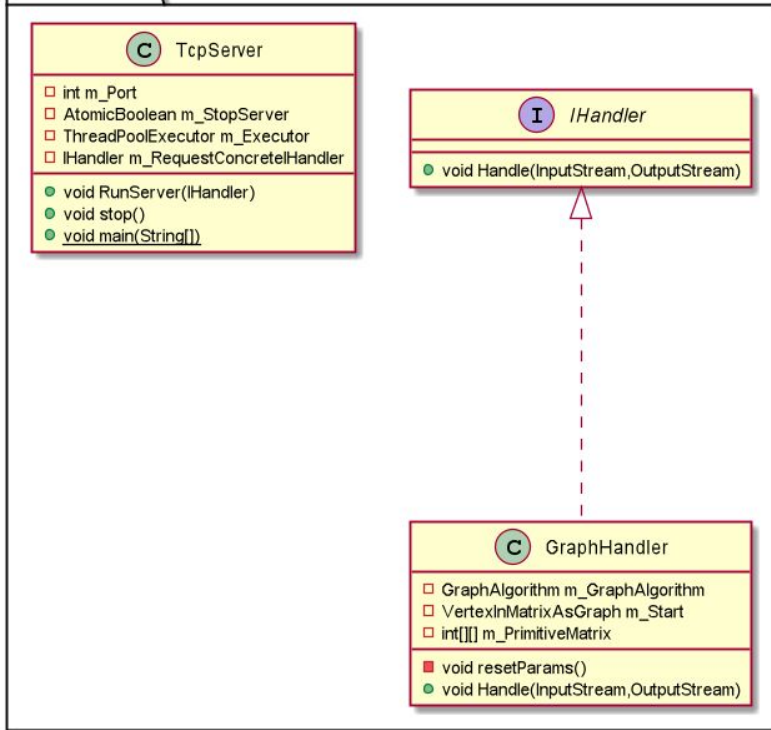
# Class diagram - part 2

FINALPROJECT's Class Diagram

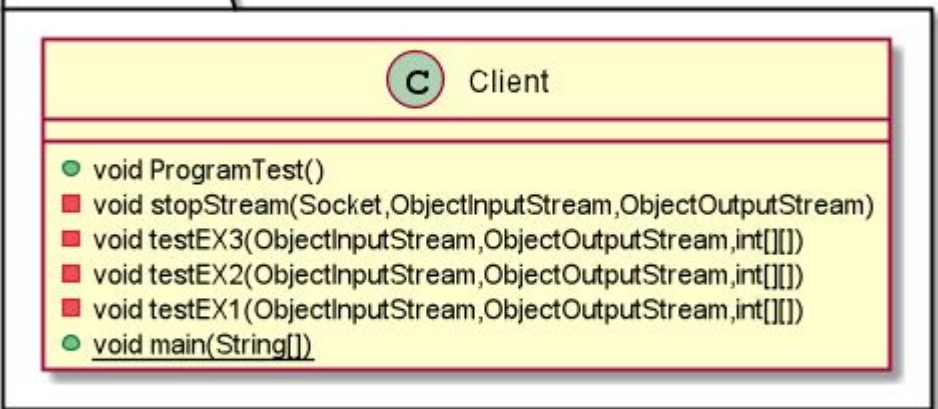


# Class diagram - Server and Client

part2.server



part2.client





# Class diagram - Server and Client

## הסבר על המחלקות:

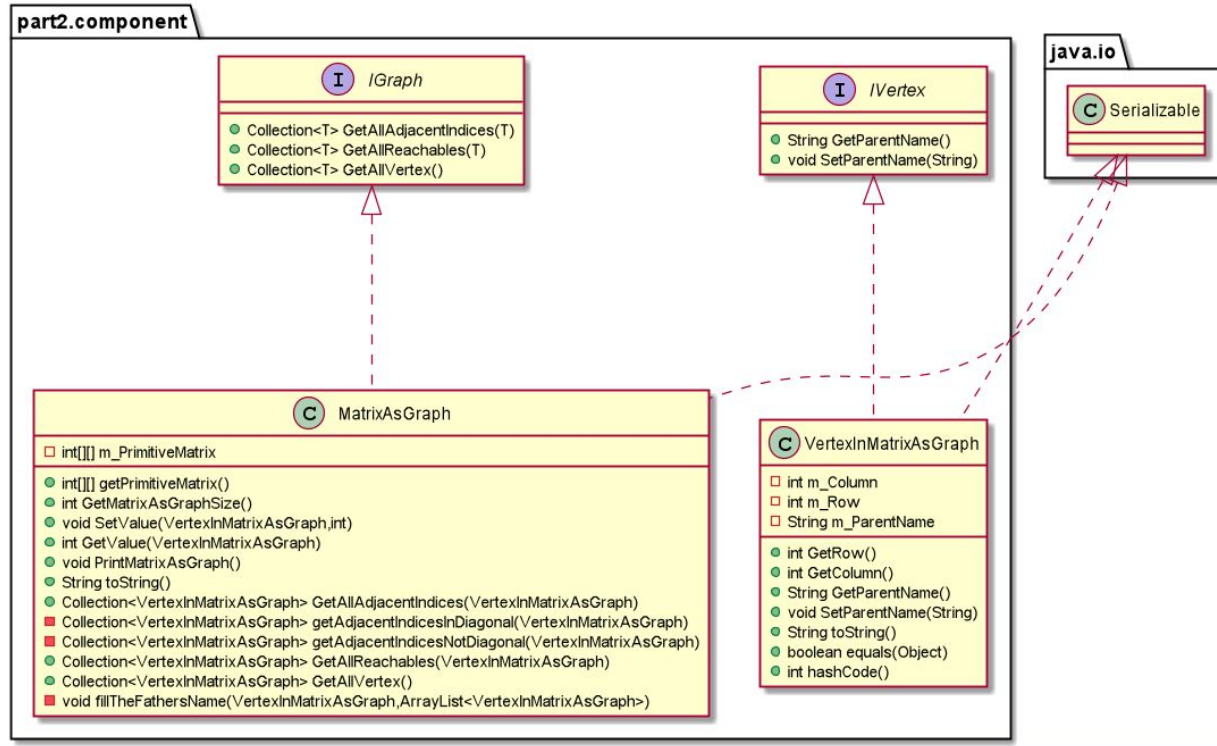
### Client

המחלקה מממשת את הצד של הלקוח ממנה ניתן להעביר נתונים דרך השרת ולקבל את התשובות הרצויות. המחלקה מבצעת תקשורת רק מול השרבר אנו פותחים בשביל זה SOCKET תפעולי כדי להפשר הזרמת מידעה בין שתי הצדדים ה SOCKET מדמה עבורנו כצינור להעברת נתונים בין שתי הצדדים

### Server

בפקג' הזה קיימים 2 מחלקות וממשק שביחד נותנים לנו את הצעד של השרת. מחלקת TCP SERVER מחלקה השרת שלנו שיודעת להזין לבקשות של המשתמשים ולשרת אותם מחלקה זו משתמשת ב Multithreading כדי לשרת מספר של לקוחות בו זמנית, מחלקה זו יש מתודה בשם RunServer אשר מקבלת את הממשק IHandler שבו מצוינים המשימות ששרת זה יודעה לבצעה. ממשק IHandler הוא ממשק כללי שבו קיימת מתודה אחת בשם Handle כל מי שמממש את הממשק צריך גם לממש את המתודה לפי צרכו וכך מחלקת TCP SERVER יכולה לקבל קלאסיים שונים שמממשים את הממשק IHandler ולשנות את המשימות שהשרבר יודעה לבצעה, תבנית עיצוב זו נקראת strategy design pattern אשר נותנת לנו את האופציה מבלי לשנות את קוד המקור של השרבר (CLASS) יכולת לשנות את המשימות שהשרבר יודעה לבצעה. במקרה שלנו המחלקת GraphHandler זוהי המחלקה שבעצמם יוצרת סוג של תפריט משימות לביצוע של השרת מחלקה זו מממשת את IHandle ואת המתודה Handle מחלקה זו מחזיקה בעזרת קומפוזיציה מחלקה בשם GraphAlgorithm אשר אחראית בפועל על ביצועה של המשימות על מחלקה זו יש הסבר בהמשך המצגת

# Class diagram - components



# Class diagram - components

## הסבר על המחלקות:

בפקג' זה קיימות שתי מחלקות ושתי ממשקים  
אובייקטים אלו מדמים לנו את הרכיבים בפרויקט

### המחלקה **MatrixAsGraph**

מממשת את הממשק  $\text{IGraph} < T >$  שמדמה לנו גרף עם התכונות שלו, זהו ממשק כללי שמשתמש ב generics כדי לקבל מימוש של הקודקודים בגרף  
במקרה שלנו T הוא `VertexInMatrixAsGraph`  
המחלקה `MatrixAsGraph` מממשת את הגרף במקרה שלנו בעזרת מטריצה (`MatrixAsGraph` עוטף מטריצה)

### המחלקה **VertexInMatrixAsGraph**

מממשת את הממשק `IVertex` אשר נותן תכונות בסיסיות של קודקוד בגרף  
המחלקה ממומשת על ידי זוג של שתי INT -ים (זוג סדור) אשר מציין את המיקום של האינדקס במטריצה במחלקת `MatrixAsGraph`

# Class diagram - Algorithms

## part2.algorithms

### GraphAlgorithm

- MatrixAsGraph m\_Graph
- Queue<VertexInMatrixAsGraph> m\_Queue
- List<VertexInMatrixAsGraph> m\_ListWithOnesVertices

- Collection<HashSet<VertexInMatrixAsGraph>> Ex1FindAllComponents(int[][])
- Collection<HashSet<VertexInMatrixAsGraph>> FindAllComponents()
- Collection<VertexInMatrixAsGraph> FindComponent(VertexInMatrixAsGraph)
- Set<Collection<VertexInMatrixAsGraph>> Ex2FindAllShortestRoutes(int[][], VertexInMatrixAsGraph, VertexInMatrixAsGraph)
- HashSet<Collection<VertexInMatrixAsGraph>> FindAllShortestRoutes(VertexInMatrixAsGraph, VertexInMatrixAsGraph)
- HashSet<Collection<VertexInMatrixAsGraph>> checkAnotherShortestRoutes(VertexInMatrixAsGraph, VertexInMatrixAsGraph, int, Collection<VertexInMatrixAsGraph>)
- Collection<VertexInMatrixAsGraph> getAllCircleVertex(VertexInMatrixAsGraph, VertexInMatrixAsGraph)
- Collection<VertexInMatrixAsGraph> findShortRoute(VertexInMatrixAsGraph, Collection<VertexInMatrixAsGraph>)
- int Ex3FindNumberOfTheCorrectSubmarines(int[][])
- int FindNumberOfTheCorrectSubmarines()
- Boolean CheckIfSubmarine(HashSet<VertexInMatrixAsGraph>)
- int getSubmarineSize(HashSet<VertexInMatrixAsGraph>)

# חלק 2 משימה 1- הסבר אלגוריתם

## חלק 1-מציאת רשימת קשירויות

מקבלים : מטריצה פרימיטיבית מהלקוח

מייצרים ממנה אובייקט גרף לאחר מכן מחלצים מהגרף(מטריצה) רשימת קודקודים שבהם יש 1 נכנסים ללולאה : כל עוד הרשימה של 1-דים לא ריקה מחלצים קודקוד ראשון -מחלצים ממנו את כל הישיגים לאורך, לרוחב ובאלכסון ודוחפים לרשימת קשירויות (רשימת קשירות ראשונה) לאחר מכן מורידים את כל הקודקודים שכבר מצאנו מרשימת הקודקודים וחוזרים על הלולאה מחדש על מנת לחפש עבור קודקודים אחרים רשימת קשירויות נוספת

מחזירים : רשימה של כל רשימת קשירויות

**הסבר על נכונות האלגוריתם:** באלגוריתם זה אנו בעצם מחפשים את כל רכיבי הקשירות של הגרף זאת אומרת אחרי ביצועה BFS מקודקוד מסוים אנחנו נקבל את הרכיב הקשירות של קודקוד זה כאשר הקודקוד הוא השורש של העץ הפורש אחרי ביצועה BFS

כדי למצוא את כל רכיבי הקשירות נרצה לעבור על כל הקודקודים שיש בגרף ולמצוא את כל הרכיבים, במקרה שלנו אנחנו בעצמם מורידים את הקודקודים שכבר נמצאו ברכיב הקשירות וכך בעצם אנחנו מייעלים את האלגוריתם זאת אומר שלא נעבור כבר על הקודקודים שנמצאים באיזשהו רכיב קשירות כבר

## חלק 2 משימה 2- הסבר אלגוריתם

### חלק 2-מציאת מספר מסלולים הכי קצרים מקודקוד לקודקוד

מקבלים : מטריצה פרימיטיבית מהלקוח, אינדקס התחלה ואינדקס סוף

בדוק כי המטריצה שקיבלנו היא לא חורגת מגודל 50 על 50

מצא עץ פורש עבור אינדקס התחלה

בדוק האם אינדקס סוף נמצא בעץ הפורש שמצאנו אם לא החזר רשימה ריקה

מצא מסלול בתוך העץ הפורש עם קודקוד סוף, תוסיף אותו לרשימת המסלולים

בדוק האם יש מעגלים: אם כן : נסה לחפש עוד מסלולים כי קיים בוודאות 1 או יותר

מחזירים : רשימה של כל רשימת קשירות

**הסבר על נכונות האלגוריתם:** בשאלה זו אנו רוצים למצוא את המסלולים הקצרים ביותר בין קודקוד התחלה לקודקוד סוף אנו יודעים שאם בגרף קיים עץ פורש ובתוך העץ הפורש הזה קיים מסלול בין שתי קודקודים אנחנו יודעים שזה המסלול הקצר ביותר, בגרף יכולים להיות מעגלים ולכן מספר המסלולים הקצרים ביותר יהיו כפל של גודל המעגלים בגרף בגלל שעברו כל מעגל ניתן להשתמש בצלע אחד שלו כדי ליצור עץ פורש

ולכן אנחנו צריכים להשתמש כל פעם בצלע אחת מתוך המעגלים ולבדוק האם דרכה יעבור המסלול האופטימלי ביותר (הקצר ביותר) ולכן נעבור על כל צלעות המעגלים שקיימים ונחפש את המסלולים הקצרים ביותר

## חלק 2 משימה 3- הסבר אלגוריתם

### חלק 3-מציאת מספר צוללות תקינות

**מקבלים : מטריצה פרימיטיבית מהלקוח**

מוצאים רשימת קשירויות (כל רשימת קשירות חשודה לצוללת) בעזרת פונקציה של חלק 1 מהמטלה עבור על כל רשימת קשירות מוך הרשימה וודו כי כמות הקודקודים גדולה מ-1 -אחרת נסיר מהרשימה לאחר מכן עבור על רשימת כל רשימת קשירויות (אם קיימת כזאת בכלל בעקבות הסעיף הקודם) וודא כי גודל משוערך של רשימת קשירות 1 שווה לכמות הקודקודים

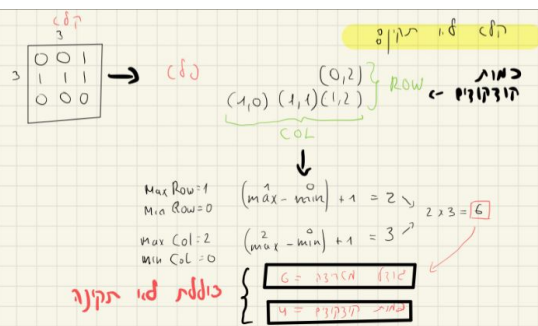
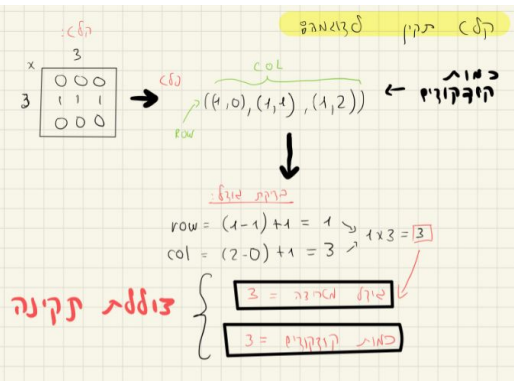
**מחזירים : רשימה של כל רשימת קשירויות**

**הסבר על נכונות האלגוריתם:** אנו נרצה למצוא את כל הצוללות על סמך הגיאומטריה שלהם שאנחנו יודעים שהיא חייבת להיות מרובעת לפי כללי המשחק

אם נקבל אוסף של קודקודים נרצה למצוא את מספר השורה המקסימלית והמינימלי ואת מספר העמודות המקסימלית והמינימלי ביצועה כפל שלהם נותנת לנו מרובע

וכך נקבל את הצורה המרובעת של הצוללת (הצורה שאם היא באמת מתקיימת הצוללת תקינה)

אם מספר זה שווה למספר של כמות הקודקודים באוסף זה אומר שהצוללת תקינה אחרת זה צוללת לא תקינה





# הסוף

# תודה על ההקשבה