



# פיתוח תוכנה מתקדם 1 מותאם לקורס קיץ

אבני דרך 1-3 בפרויקט

ד"ר אליהו חלסצ'י  
[eliahukh@colman.ac.il](mailto:eliahukh@colman.ac.il)

## רקע

פרויקט זה בקורס פת"מ הוא חלון הראווה שלכם כשתרצו להציג את הניסיון התכנותי שצברתם. פרויקט זה מכיל את האלמנטים הבאים:

- שימוש בתבניות עיצוב וארכיטקטורה
- תקשורת וארכיטקטורת שרת-לקוח
- שימוש במבני נתונים ובמסד נתונים
- הזרמת נתונים (קבצים ותקשורת)
- השוואה, בחירה והטמעה של אלגוריתמים בתוך המערכת שניצור
- תכנות מקבילי באמצעות ת'רדים
- תכנות מוכוון אירועים, אפליקציית desktop עם GUI
- תכנות מוכוון אירועים, אפליקציית Web

בסמסטר זה (פת"מ 1) עליכם להגיש 3 אבני דרך, עבור צד שרת בפרויקט שלנו.

1. יצירת תשתית מחלקות עבור צד השרת; כך נטמיע עקרונות עיצוב קוד
2. מימוש שרת "פותר הבעיות" עבור בעיה פשוטה לבדיקה; כך נטמיע הזרמת מידע דרך קבצים ותקשורת
3. מימוש של מספר אלגוריתמי חיפוש, השוואה ביניהם מי הכי מוצלח, והטמעת המנצח כפותר הבעיות בצד השרת. כך נטמיע את צורת העבודה המלאה של בוגר מדעי המחשב.

בסמסטר הבא (פת"מ 2) נממש את האפליקציות בצד הלקוח שיעשו שימוש בשרת הכללי שיצרנו עבור פתרון בעיות חיפוש ספציפיות.

לפרויקט קיימת בדיקה אוטומטית. כל הפרטים כגון אופן ותאריכי ההגשה ימצאו באתר הקורס.

בהצלחה!

## אבן דרך 1 – חובה – שמירה על עקרונות Solid

בכיתה למדנו אודות עקרונות עיצוב שונים, ובפרט על העקרונות single responsibility ו open/close של SOLID. בבואנו לכתוב את צד השרת נרצה לקיים את העקרונות האלה. מכיוון שזו אבן הדרך הראשונה, נעשה זאת יחד באופן מודרך.

ברצוננו לכתוב שרת כללי, כלומר שרת שיוכל לשמש אותנו שוב ושוב בפרויקטים שונים. לשם כך, עליו לבצע הפרדה עקרונית וחשובה, מהי לדעתכם?

- א. הפרדה בין צד השרת ללקוח
- ב. הפרדה בין מה שמשתנה בין פרויקט לפרויקט, לבין מה שלא.
- ג. הפרדה בין פרוטוקולים שונים לתקשורת

התשובה מופיעה בעמוד הבא. נסו לחשוב לבד – איזו הפרדה עקרונית אנו צריכים?

התשובה היא ב'. אנו עוסקים כרגע בצד השרת. ברור שזו ישות נפרדת מצד הלקוח. הרצון שלנו הוא ליצור שרת כללי – שרת שנוכל להשתמש בו שוב ושוב. לכן זו הפרדה שצריכה לקרות במקום כלשהו בצד השרת. כמו כן, בוודאי שיתכנו פרוטוקולים שונים של תקשורת בין פרויקט לפרויקט, אך ההפרדה היא עקרונית ומהותית יותר. עלינו להפריד בין מה שמשתנה לבין מה שקבוע.

את המגננונים שאינם משתנים נוכל לממש כבר כעת. אולם, פעולות שעשויות להשתנות מפרויקט לפרויקט לא נרצה לממש, אלא נרצה רק לדעת להפעיל אותן בבוא העת כשנצטרך. לשם כך נוכל לנצל את הקיום של ממשקים (interfaces).

חישבו כיצד?

תשובה:

```
interface Socket {
    void connect(String host, int port);
    void write(byte[] data);
    void read(byte[] data);
    void close();
}

class MyServer implements Socket {
    // ...
}
```

אז בואו נתחיל.

אילו הינו מתחילים עם מחלקה שמממשת שרת, דהיינו מנגנון שמאזין וממתין ללקוחות שיתחברו ואז מטפל בבקשות שלהם, אז הינו כותבים מנגנון אחד שלא בהכרח היה מתאים לכל פרויקט. יותר נכון זה **להגדיר את הפונקציונאליות של השרת באמצעות ממשק**, ובכל פרויקט יכולה להיות מחלקה אחרת שתממש את אותה הפונקציונאליות בדרך שונה. לדוגמא, הסמסטר נממש מחלקה שתטפל בלקוחות אחד אחרי השני ואילו בפת"מ 2 תוכלו להוסיף מחלקה המהווה שרת שמטפל בכל הלקוחות במקביל.

נשים לב שמתקיימת כאן שמירה על עקרון open/ close שכן, הוספה של מחלקה שמימשה את אותו הממשק היא הרחבה של הפונקציונאליות הנדרשת (open) מבלי שהיינו צריכים לשנות קוד שכבר כתבנו (close) – פתוח להרחבה אך סגור לשינויים.

צרו פרויקט ובו package בשם server\_side ובתוכו את הממשק Server. נתחיל מפונקציונאליות פשוטה:

- המתודה open תקבל פרמטר int port להאזנה ותפקידה יהיה לפתוח את השרת ולהמתין ללקוחות.
- המתודה stop תסגור את השרת.

חישבו האם צריך מתודות נוספות.

כעת, צרו את המחלקה MySerialServer מהסוג של Server. השאירו הערות TODO, את הקוד תממשו באופן הדרך הבאה; כעת זו רק התשתית.

## שיחה עם הלקוח

תארו לכם מצב שבו במחלקה MySerialServer גם מימשנו את פרוטוקול השיחה בין הלקוח לשרת. הרי בכל פרויקט תיתכן שיחה שונה בפורמט שונה ועם ציפיות שונות בין הלקוח לשרת. כך לא נוכל להשתמש במחלקה זו בפרויקטים אחרים. בין מה למה עלינו להפריד כדי לשמור על עקרונות open close ו single responsibility?

- א. עלינו להפריד בין מחלקות שונות שמימשו את Server, בכל מחלקה נממש פרוטוקול שונה
- ב. עלינו להפריד בין המנגנון של השרת שמומש ב MySerialServer לבין צורות שיחה שונות עם הלקוח.

א' היא כמובן תשובה לא נכונה, מפני שגם עם אותו המנגנון נרצה לקיים שיחות שונות. למשל, תארו מצב שבו יש לנו שני מנגנונים של שרת – MySerialServer מטפל בלקוחות אחד אחרי השני ואילו MyParallelServer מטפל בהם במקביל. כמו כן, יש לנו שני פרוטוקולים של תקשורת – באחד הלקוח שולח מחרוזת לשרת והשרת מחזיר מחרוזת הפוכה, ואילו בפרוטוקול השני הלקוח שולח משוואות מתמטיות והשרת מחזיר פתרון. לפי תשובה א' נצטרך 4 מחלקות (!) שרת טורי שהופך מחרוזות, שרת טורי שפותר משוואות, שרת מקבילי שהופך מחרוזות ושרת מקבילי שפותר משוואות. לא הגיוני.

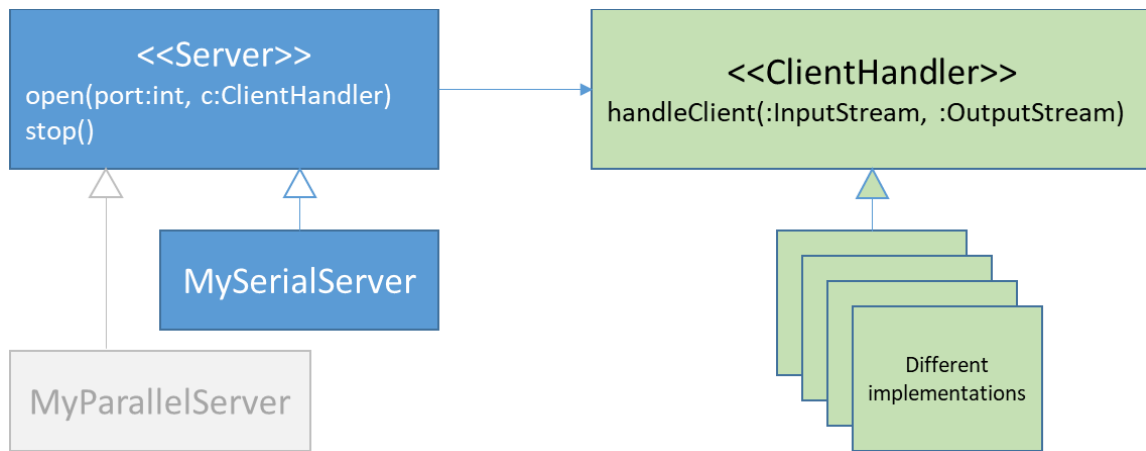
לפי ב', לעומת זאת, אם ניצור ממשק בשם ClientHandler שמטרתו לקבוע את סוג השיחה עם הלקוח והטיפול בה, אז נוכל להסתפק רק בשתי מחלקות עבור מנגנונים של שרת (MySerialServer, MyParallelServer) ולכל אחת מהן נוכל להזריק איזה מימוש שנרצה עבור ClientHandler. בפרט, לכל מימשו של Server נוכל להזריק שיחה של היפוך מחרוזות או פתרון משוואות. באותו האופן אם מחר נרצה לממש פרוטוקולים נוספים אז נצטרך רק להוסיף מימוש של ClientHandler מבלי לשנות או להעתיק שוב את הקוד של המנגנונים השונים לשרת.

נשים לב שבשיטה זו שמרנו גם על single responsibility וגם על open close.

הגדירו את הממשק ClientHandler עם המתודה handleClient שמקבלת InputStream (ממנו נקרא את הודעות הלקוח) ו OutputStream (שאליו נכתוב את תשובת השרת).

## היכן נטמיע את ההזרקה של ClientHandler?

הביטו בתרשים הבא ונסו לענות מדוע נכון יותר שזה יהיה כפרמטר של מתודה ב Server ולא data member באחת המחלקות שירשו אותו?



על איזה עיקרון של SOLID שומר הפרמטר c במתודה open של Server?

תשובה: Dependency Inversion – המחלקות שירשו את Server מקבלות תלות מסוג ClientHandler. הן מחליטות מתי להפעיל את ה ClientHandler ואילו הוא מחליט על המימוש.

כעת, ממשו את המחלקה MyTestClientHandler כסוג של ClientHandler שתשמש אותנו בהמשך לבדיקת התשתית.

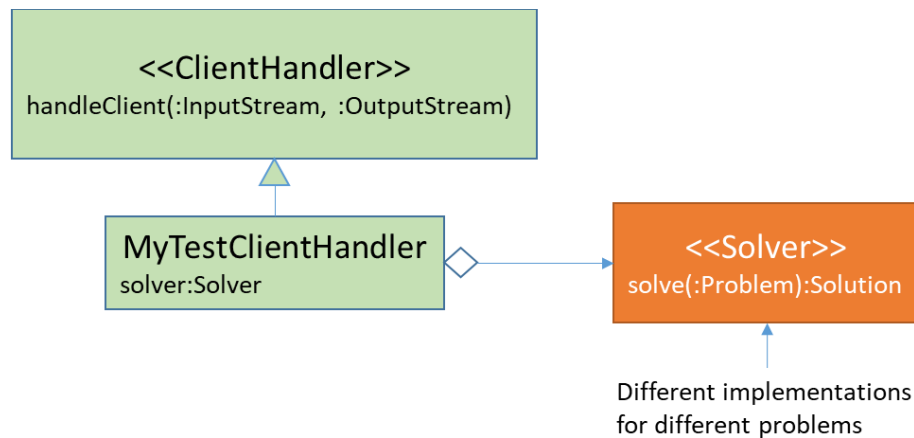
### פתרון בעיות

בהמשך לדרך בה אנו שומרים על single responsibility ו open close נרצה להפריד גם בין פרוטוקול השיחה בין השרת ללקוח (שמומש אצל יורש כלשהו של ClientHandler) לבין האלגוריתם שפותר את הבעיה. אחרת, נצטרך מחלקה אחת עבור כל שילוב אפשרי של פיענוח פרוטוקול שיחה עם כל אלגוריתם...

לדוגמא, נניח שקיימים שני אלגוריתמים A ו B לפתרון של משוואות מתמטיות. ללא הפרדה, נצטרך שני מימושים של ClientHandler עבור אותו פרוטוקול תקשורת. בשני המימושים יהיה **קוד כפול** שקורא למשל מחרוזת שהגיע מהלקוח ושולף מתוכה את הנתונים. אך מימוש אחד מבצע את אלגוריתם A ואילו השני את B. מיותר.

הביטו בתרשים הבא וענו על השאלות הבאות

- הסבירו במילים שלכם כיצד ומדוע העיצוב בתרשים הבא נמנע מהקוד הכפול המתואר לעיל
- מדוע בחרנו לבצע את ההכלה ב MyTestClientHandler ולא בממשק ClientHandler עצמו?



רמז:

- האם משהו ישנה בקוד של MyTestClientHandler אם נרצה להשתמש באלגוריתם אחר כדי לפתור את אותה הבעיה?
- האם בהכרח כל ClientHandler פותר בעיות אלגוריתמיות?

## שמירה של פתרונות (Caching)

ייתכן וחשוב הפתרון לוקח המון זמן. יהיה זה מיותר לחשב פתרון עבור בעיה שכבר פתרנו בעבר. במקום זאת, נוכל לשמור פתרונות שחישבנו בדיסק. במידה ומגיעה בעיה, נצטרך לבדוק במהירות האם כבר פתרנו אותה בעבר, אם כן, נשלוף את הפתרון מהדיסק במקום לחשב אותו. אחרת נפתור את הבעיה ונשמור את הפתרון בדיסק.

בשלב זה, אתם כבר מבינים שיתכנו מספר מימושים שונים לשמירת הפתרונות, לדוגמא בקבצים או במסד נתונים. לכן, ניישם שוב את אותה הטקטיקה של שימוש בממשק כדי לשמור על העקרונות השונים של SOLID. נגידר את הממשק CacheManager שינהל עבורנו את ה cache (בעברית - מטמון).

חישבו, מהי הפונקציונאליות של CacheManager? אלו מתודות תגדירו לו?

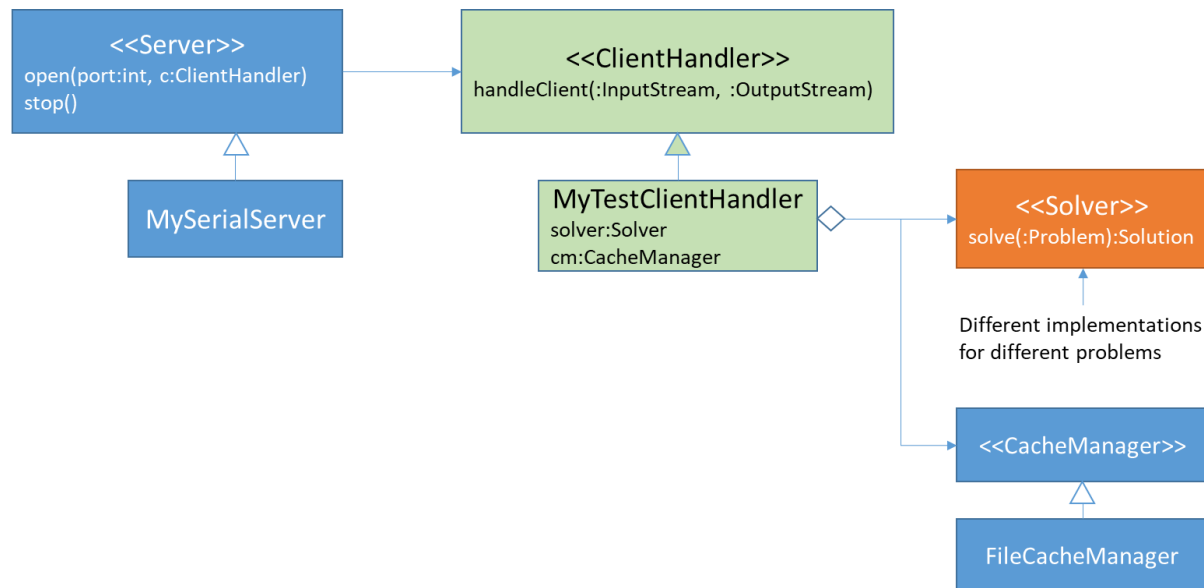
- P להכניס פתרון לcache
  - P להוציא פתרון מcache
  - P להאמת אם פתרון קיים בcache
- ה CacheManager יצטרך לממש את הממשק:

תגדירו את המתודות לממשק CacheManager. ממשו את הממשק במחלקה שנקראת FileCacheManager. מחלקה זו באבן הדרך הבאה תשמור פתרונות בתוך קבצים בדיסק.

צרו את העיצוב המלא בתרשים הבא ותגישו לבדיקה במערכת הבדיקות לפי ההוראות הכתובות באתר הקורס.

בנוסף, יהיה אוסף של שאלות אמריקאיות אודות העיצוב שיצרנו על מנת לוודא הבנה.

בהצלחה.



## אבן דרך 2 – לא חובה – streaming

רקע:

החל מאבן דרך זו, בהדרגה מסמך זה יגדיר יותר **מה** לעשות (דרישות) ופחות **איך** לעשות זאת. הבחירה עוברת לידיים שלכם.

בהמשך לשיעורים אודות Streaming של מידע לקבצים ולערוצי תקשורת עליכם לממש את המחלקות השונות ע"פ העיצוב שלנו לצד השרת מאבן דרך 1. אולם, הבעיה שנפתור תהיה לעת עתה בעיית צעצוע רק כדי שנוכל לבדוק את התשתית שיצרנו. בהינתן מחרוזת מהלקוח, השרת יצטרך להחזיר מחרוזת הפוכה. בהמשך, נעדכן את השרת שלנו כך שיפתור בעיות חיפוש.

### MySerialServer

בהתאם לקוד שהדגמנו בשיעור, במתודה `start` ממשו לולאה שמאזינה ללקוח על ה `port` שקבלנו. לאחר שהלקוח מתחבר היא תפעיל את ה `ClientHandler` שהזון לה כדי לשוחח עם הלקוח. בתום השיחה נחזור לתחילת הלולאה ונאזין ללקוח הבא. כך נטפל בלקוחות אחד אחרי השני באופן טורי.

דגשים:

- שימו לב שיש להריץ את הלולאה הזו בת'רד נפרד כפי שהודגם בשיעור, אחרת מי שמפעיל את start מפעיל לולאה שלעולם לא תסתיים ולכן לא יוכל להגיע לשורה הבאה ולקרוא ל stop.
- הקפידו להגדיר time out להמתנה ללקוח, אחרת גם הפעלה של stop לא תועיל מפני שנחכה לנצח ללקוח שלעולם לא יגיע, והשרת יישאר פתוח לעד.

## MyTestClientHandler

מחלקה זו תקרא מה InputStream של הלקוח שהתחבר שורה אחר שורה באמצעות BufferedReader. לשם כך עליכם להשתמש בעוטפים המתאימים כפי שלמדנו בשיעור. שימו לב מי מהם מימש Decorator ומי מהם מימש Object Adapter.

בהינתן שורה, נשאל את ה CacheManager שהוזן לנו האם יש לו פתרון שמור למחרוזת זו. אם כן, נשלוף באמצעותו את הפתרון ונכתוב אותו חזרה ללקוח. אם לא, נפעיל את ה Solver שניתן לנו על מנת לקבל את הפתרון (מחרוזת הפוכה לזו שהלקוח שלח). נבקש מה CacheManager לשמור את הפתרון ונשלח אותו חזרה ללקוח.

תהליך זה יחזור על עצמו עם כל שורה המגיעה מהלקוח.

שימו לב שהלקוח הוא זה שכותב ראשון, ושלאחר כל שורה הוא מצפה לקרוא תגובה לפני שהוא שולח את השורה הבאה שלו.

טיפ: flush

פרוטוקול התקשורת מסתיים כאשר הלקוח כותב את המילה "end" השרת לא יחזיר תשובה ויסיים את ההתקשרות עם אותו הלקוח.

## Solver

שנו את הממשק כך שבמקום ש Problem ו Solution יהיו מחלקות, הם יהיו טיפוסים פרמטריים של Solver. כלומר `Solver<Problem, Solution>`. כך של מחלקה שתממש את Solver תוכל לבחור מהו הטיפוס המייצג את הבעיה ומהו הטיפוס שמייצג את הפתרון. מחקו את המחלקות המיותרות.

## StringReverser

צרו את המחלקה StringReverser כמימוש של Solver המקבל מחרוזת ומחזיר מחרוזת הפוכה. תוכלו כמובן להשתמש במופע של StringBuilder לשם כך.

טיפ: לא באמת צריך לממש מחלקה כזו, אפשר לבצע את הכל באמצעות ביטוי למדה במקום המתאים.

## FileCacheManager

אתם מחליטים כיצד הכי נכון לממש אותו. בהינתן מחרוזת הוא צריך לדעת ב  $O(1)$  האם שמור לו פתרון. כמובן עליו לתמוך בשמירה ושליפה של פתרונות מהדיסק.

## Main

בצד השרת צרו package בשם boot ובתוכו מחלקה בשם Main עם מתודת main. ה main תפעיל את השרת MySerialServer כ Server

- על port לפי האורגומנט הראשון של פונקציית ה main (`args[0]`)



- עם StringReverser כ Solver
- עם FileCacheManager כ CacheManager

**אופי הבדיקה במערכת הבדיקות:**

- נפעיל את השרת שבניתם על פורט אקראי
- נריץ מספר לקוחות בזה אחר זה
- כל לקוח ישלח מספר מסוים של מחרוזות לפני שישלח end
- הבדיקה תצפה לקבל את המחרוזות ההפוכות.

בהצלחה!

## אבן דרך 3 – חובה - מימוש אלגוריתם

בהמשך לשיעור מפסאודו קוד של אלגוריתם ל OOP, עליכם להשתמש בתבנית העיצוב Bridge כדי להפריד בין האלגוריתם לבין הבעיה שאותה הוא פותר. בפרט, השתמשו בממשק Searchable כדי להגדיר מהי הפונקציונאליות של בעיית חיפוש, ובממשק Searcher עבור אלגוריתם חיפוש.

- השלימו את האלגוריתם BestFirstSearch
- באופן דומה ממשו צרו את האלגוריתמים הבאים:

DFS

BFS (breadth first search)

Hill Climbing

A\* (A star)

כעת ערכו ניסוי אמפירי שבודק מי מהם עובד הכי טוב:

1. תגדירו אוסף של 10 מטריצות בגודל  $N \times N$  עבור  $N$  הולך וגדל החל מ  $N=10$  ועד  $N=50$ . בכל תא תגדירו איזשהו ערך שלם המבטא את המחיר לדרך בתא זה. למשל 0 זה חינום או מישור, ככל הערך גדול יותר אז זה כמו עליה קשה יותר, ואינסוף זה קיר שלא ניתן לעבור דרכו. הכניסה לשטח תוגדר בתא 0,0 ואילו היציאה בתא  $N-1, N-1$ .
2. תריצו כל אחד מהאלגוריתמים 10 פעמים על כל אחת מ 10 המטריצות (בסך הכל 5 אלג'  $10 \times 10$  ריצות  $10 \times 10$  מטריצות) ובדקו כמה קודקודים פיתח כל אלגוריתם והאם הוא הגיע לפתרון – כלומר המסלול הזול ביותר
3. הציגו את הנתונים בגרף שבו ציר ה  $X$  הוא ערכי ה  $N$  וציר ה  $Y$  הוא מס' הקודקודים שפיתח כל אלגוריתם בממוצע על פני 10 הריצות.

לפי הקווים שנוצרו בגרף תוכלו לראות איזה מימוש הוא היעיל ביותר. זה המימוש שתבחרו להטמיע בצד השרת. דגש: כל המימושים קיימים בצד השרת, אולם ב main נחבר רק את הטוב ביותר.

## הטמעת האלגוריתם בצד השרת

האלגוריתם שלנו מימש את הממשק Searcher ואילו ה ClientHandler שלנו מצפה לאובייקט מסוג Solver. אילו הממשק Searcher היה יורש את Solver הינו פותרים באופן טכני את הבעיה הזו אך הפתרון היה ממש לא נכון, שכן, כל המחלקות מסוג Searchable צריכות להיות כעת גם Solver בסתירה לעקרון ה Interface segregation. אילו הינו רוצים לקחת את כל ה searchers שלנו לפרויקט אחר שבו הם לא היו צריכים להיות גם solvers אז הינו מוצאים את עצמנו בבעיה.

מצד שני, אילו מחלקה מסוימת כמו למשל BFS היתה מממשת את Solver בנוסף ל Searcher אז יצרנו סתירה לעיקרון של Open/Close כי נאלצנו לשנות קוד קיים.

לכן, השתמשו ב Object Adapter כדי לפתור את הבעיה וראו כיצד הוא גורם לשמירה על כל עקרונות SOLID שלמדנו.

## MyClientHandler

צרו את המחלקה MyClientHandler כך שתתאים לפרוטוקול התקשורת הבא:

- הלקוח שולח שורה אחר שורה עד שמתקבלת שורה עם הערך "end"
- כל שורה מכילה ערכים מספריים המופרדים ע"י פסיק. כך אוסף השורות יוצר מטריצה של ערכים.
- לאחר מכן הלקוח שולח שתי שורות נוספות. בכל שורה שני ערכים המופרדים ע"י פסיק: שורה ועמודה.  
Row, Col
  - הערכים בשורה הראשונה מציינים את הכניסה לשטח
  - הערכים בשורה השנייה מציינים את היציאה מהשטח
- כעת השרת יחזיר מחרוזת אחת בלבד, עם ערכים המופרדים בפסיק. הערכים יהיו מסוג המילים { Up, Down, Left, Right } שמציינות את הכיוון שיש לנוע לפיו על מנת לחצות את השטח במסלול הזול ביותר. אופי הבדיקה במערכת הבדיקות – בדומה לאבן דרך 2, פרט לכך שנבדוק האם באמת קבלנו את המסלול הזול ביותר.

בהצלחה!