



# תכנות מתקדם 1

אבן דרך 2

ד"ר אליהו חלסטצ'י  
[Eliahu.khalastchi.biu@gmail.com](mailto:Eliahu.khalastchi.biu@gmail.com)

## רקע

פרויקט זה בקורס תכנות מתקדם 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 בסגנון REST
- אפליקציית מובייל (אנדרואיד)

בסמסטר זה עליכם להגיש 2 אבני דרך:

1. מפרש קוד המאפשר שליטה מרחוק בסימולטור טיסה
2. 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.

## אבן דרך 2 – שמירה על עקרונות Solid

בכיתה למדנו אודות עקרונות עיצוב שונים, ובפרט על העקרונות open/close ו single responsibility של SOLID. בבואנו לכתוב את צד השרת נרצה לקיים את העקרונות האלה. מכיוון שזו הפעם 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

נסו לחשוב לבד – איזו הפרדה עקרונית אנו צריכים?

תשובה:

[illegible]

את המנגנונים 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:

1. Երբ Դավիթը և Բաղդասարը հասան Երուսաղեմ, Դավիթը  
 2. ասաց Բաղդասարին՝ «Երբեք չես հեռացել ինձից, քանի որ  
 3. ես եմ քեզ համարում իմ հավատարմագրյալը»։  
 4. Երբ Դավիթը և Բաղդասարը հասան Երուսաղեմ, Դավիթը  
 5. ասաց Բաղդասարին՝ «Երբեք չես հեռացել ինձից, քանի որ  
 6. ես եմ քեզ համարում իմ հավատարմագրյալը»։

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 שמיימשה את אותו הממשק היא הרחבה של הפונקציונליות הנדרשת (open) מבלי שהיינו צריכים לשנות קוד שכבר כתבנו (close) – פתוח להרחבה אך סגור לשינויים.

צרו פרויקט ובו namespace 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 `MySqlServer` מהסוג של `Server`. את הקוד תממשו בהמשך; 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 single open close 1 responsibility?

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

א' היא כמובן תשובה לא נכונה, מפני שגם עם אותו המנגנון נרצה לקיים שיחות שונות. למשל, תארנו 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.

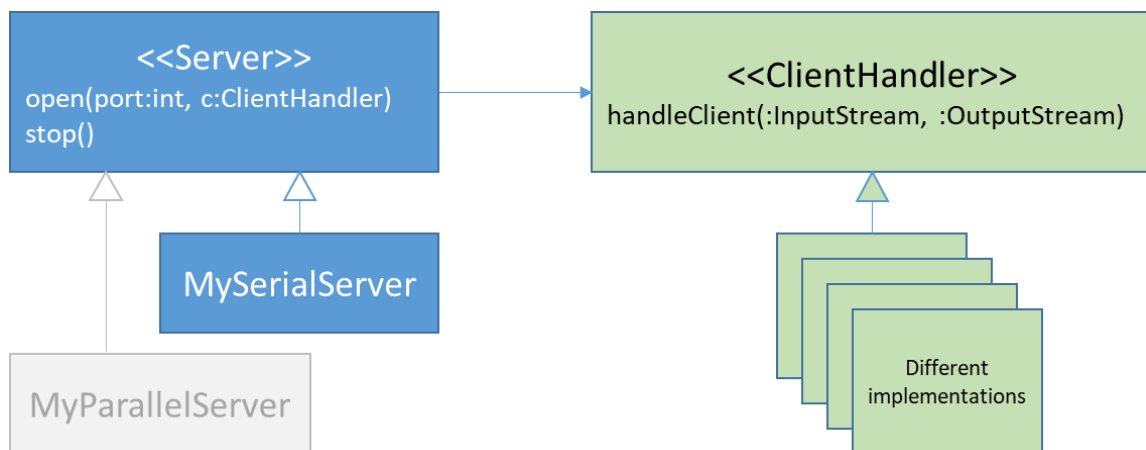
לפי ב', לעומת זאת, אם ניצור ממשק בשם `ClientHandler` שמטרתו לקבוע את סוג השיחה עם הלקוח והטיפול בה, אז נוכל להסתפק רק בשתי מחלקות עבור מנגנונים של שרת (`MySerialServer`, `MyParallelServer`) ולכל אחת מהן נוכל להזריק איזה מימוש שנרצה עבור `ClientHandler`. בפרט, לכל מימשו של `Server` נוכל להזריק שיחה של 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 `stream` של קלט (ממנו נקרא את הודעות הלקוח) ו `stream` של פלט (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.

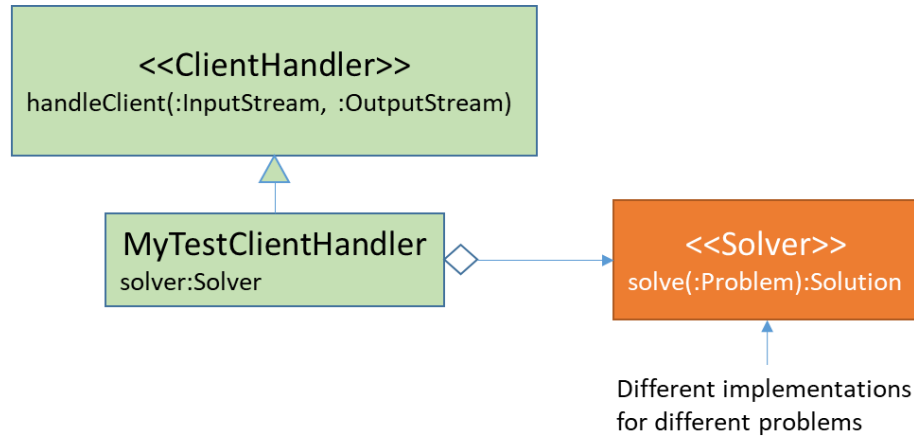
### פתרון בעיות

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

- א. הסבירו במילים שלכם כיצד ומדוע העיצוב בתרשים הבא נמנע מהקוד הכפול המתואר לעיל  
ב. 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 וחישוב 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. נגדיר את הממשק CacheManager שינהל עבורנו את ה cache (בעברית - מטמון).

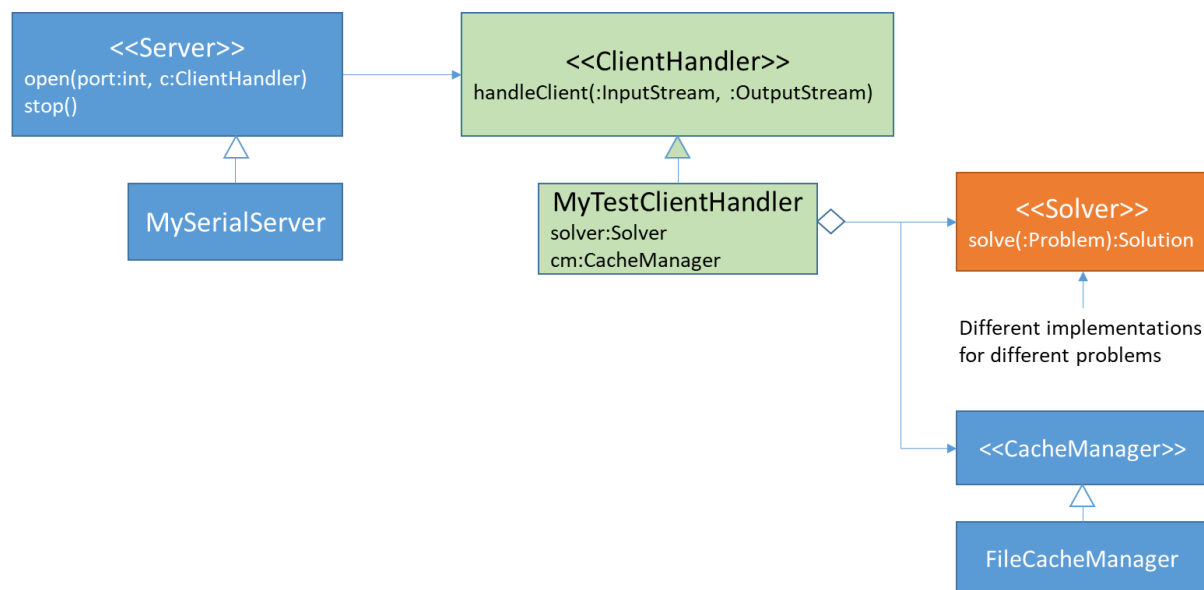
חישבו, מהי הפונקציונליות של CacheManager? אלו מתודות תגדירו לו?

- ה-Get מוצא לנו את הפתרון שאנחנו מחפשים
- ה-Set מציב לנו פתרון שאנחנו רוצים לשמור
- ה-IsExist מציב לנו פתרון שאנחנו רוצים לבדוק האם הוא קיים

ה CacheManager יתנהל לנו את כל הפונקציות האלו

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

בהצלחה.



## אבן דרך 2 חלק ב' – streaming

### רקע:

החל מחלק זה, 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 מחלק א'. 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 שהזן 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 שנכחה לנצח ללקוח שלעולם לא יגיע, והשרת יישאר פתוח לעד.

## MyTestClientHandler

מחלקה זו תקרא מה `InputStream` של הלקוח שהתחבר שורה אחר שורה. בהינתן שורה, נשאל את ה `CacheManager` שהחזן לנו האם יש לו פתרון שמור למחרת זו. אם כן, נשלוף באמצעותו את הפתרון ונכתוב אותו חזרה ללקוח. אם לא, נפעיל את ה `Solver` שניתן לנו על מנת לקבל את הפתרון (מחרת 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 מחרת ומחזיר מחרת הפוכה. תוכלו כמובן להשתמש במופע של `StringBuilder` לשם כך.

## FileCacheManager

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

## Main

בצד השרת צרו namespace 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`

בהצלחה!

## אבן דרך 2 חלק ג' – מימוש אלגוריתם

בהמשך לשיעור מפסאדו 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' X 10 ריצות X 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 מחילה 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.

כעת ממשו את MyParallelServer כך שיטפל בכל הלקוחות במקביל.

בסמסטר הבא, נשתמש בשרת הזה כדי לחשב למטוס מסלול טיסה אידיאלי. נכתוב אפליקציות שונות שמטיסות את המטוס, ומתחברות כלקוח לשרת זה.

בהצלחה!