חלק ב' סעיף ג'.

command line interface – CLI מאגר שרת שמאפשר לקוח מתחבר עליו יפעיל לו תפריט טקסטואלים שהשרת שולח והאמצעותם מתבצע האינטראקציה בין השרת ללקוח.

מטרת השרת שנכתב לבנות- הלקוח יוכל להעלות לשרת קובץ csv עשיר פרמטרים שם האלגוריתם השרת מקבל פרוט חריגות שנתפסו, ונתונים איזה האלגוריתם על ה data שלהם.

# השרת יעבוד לוקח במספר לקוחות בו־זמנית ויעבוד אם להפסיק את הכמות הזו כדי שהשרת לא יקרוס.

## משימה א' CLI

הספר על command pattern

לשם מימוש ה CLI נשתמש בתבנית עיצוב שנקראת Command Pattern בה לכל פקודה במערכת שלנו יש מחלקה משלה מסוג Command. הטיפוס Command יכול להגדיר כל מה שרלוונטי לכל הפקודות במערכת שלנו, ובפרט פקודת execute אבסטרקטית עבור הפעלה. היתרונות הם:

- מכנה משותף פולימורפי לכל הפקודות
  - ניתן למשל להכניס את כל הפקודות למבנה נתונים
  - כגון מפה או מערך, ובהינתן ה key (או אינדקס) מיד לשלוף את הפקודה ולהפעילה
  - לרכז את כל הבקשות לפקודות המגיעות במקביל בתור \ תור עדיפויות
- פתוח להרחבה – ניתן להוסיף עוד מחלקות Command ע"פ הצורך \ לרשת פקודות קיימות
- ליצור הרכבות עם תבניות עיצוב אחרות, למשל עם Composite. אם למשל המחלקה MacroCommand תחזיק מערך פולימורפי של Commands אז כשנקרא ל execute שלה היא בתורה תקרא ל execute של כל פקודה במערך, שבתורן יכולות להיות פקודות רגילות (עלים) או גם MacroCommands.

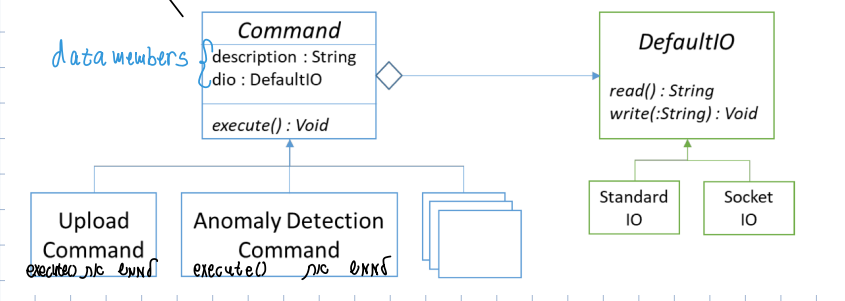
אך היתרון העיצובי הוא החשוב ביותר – והוא ההפרדה בין יוזם הפקודה (invoker) לבין מי שהולך להיות מופעל (receiver). למשל אם יש 5 דרכים שונות ליזום את אותו הדבר (נניח פעולת הדבק) אז מכולן יהיה קישור לאותו אובייקט פקודה ויצרנו מקור אחד של אמת אם נרצה לשנות בעתיד משהו.

לעוד מידע תוכלו לצפות בסרטון 13.1

נגדיר את העיצוב הבא:

מחלקה אבסטרקטית עם מתודה אחת אבסטרקטית

לכל command יש description




הספר על command

בנוסף לכל Command יש מחרוזת description. כדי ליצור תפריט למשל, נוכל להשתמש במערך של Command-ים ולעבור על כל Command ולהדפיס את ה description שלו. כאשר המשתמש יבחר אופציה $i$ נוכל ללכת ל Command ה i במערך ולקרוא ל execute שלה. הפקודה הזו בתורה תמשיך את האינטראקציה עם המשתמש לפי הצורך. היא אפילו יכולה בתורה להפעיל פקודות אחרות.

אולם, לא תמיד נרצה להדפיס למסך או לקרוא מהמקלדת... בהקשר שלנו נרצה להשתמש בעיצוב הזה בתוך שרת, כאשר את הקלט והפלט אנו מבצעים דרך socket-ים של תקשורת. לשם כך הגדרנו את הטיפוס המופשט DefaultIO שהיורשים שלו יצטרכו לממש בדרכם את המתודות הקריאה והכתיבה. כך נוכל להזין בזמן ריצה ל Command מימושים שונים של DefaultIO. בדומה לעיל, אם נרצה קלט-פלט סטנדרטי אז נזין לו את StandardIO ואילו אם נרצה באמצעי תקשורת אז נזין SocketIO. נשים לב שזה גם פתוח להרחבה, כי אם נרצה למשל לבדוק ולכתוב לדיסק אז נוכל להגיש מימוש ל DefaultIO וכל ה Command-ים לא

טיפוס מופשט DefaultIO היורשים ומממשים מתודות קריאה וכתיבה לבין סוגים נזין standart IO

**טיפ:** תחילה תעבדו עם standardIO. זה יהיה הרבה יותר נוח ונכון לדיבאג. כשהכל יעבוד ב CLI, תוכלו לממש את socketIO ואז ה CLI יזין אותו ל Command, ולראות שהכל עובד גם דרך ערוצי התקשורת.

משימה מס 1 - לכתוב פקודות בפ שרת

command 1 - יש לנו " input.txt " והמטרה היא לחלק את - ה train.csv ולאחר מכן את ה test.csv ולשמור אותם. נעבור לאחר שחילקנו אותם, שיש לנו 200 נתונים

command 2 - לקלוט אותם הקובץ את ה- קונפיג שהמשתמש הכין, אם היא עומדת בפל שבין 1-0 אפ נשנה את הקונפיג שלנו

command 3 - נריץ את האלגוריתם של learnormal על קובץ ה train ונריץ את קובץ ה test על האלגוריתם של detect.

command 4 — השרת יפרס את קובץ התוצאות בפורמט הבא:
שורת הראש "time stemp"

command 5 — נקבל את הקובץ חריגות (test) מהמשתמש, המשתמש בפה הלקוח מקליד את שם הקובץ ואיך הלקוח שולח לשרת שורה אחר שורה מהקובץ עד לאיזה done

קובץ החריגות הוא קובץ טקסט בו בכל שורה יופיעו הזמנים שבהם אכן הופיעו חריגות (לפי סדר כרונולוגי).
הפורמט הוא time step של תחילת החריגה, פסיק, time step של סיום החריגה. לדוגמה:

```
122,150
180,185
1001,1020
done
```

חריגה כוללת את זמן סיום החריגה. למשל שדה החריגה בין 122-150

היא 29 (כי סופרים גם את 150)

השרת ישווה את דיווחי הלקוח חריגות לדיווחים הללו. (detect)

1) הלקוח חריגות בנוי מ- description ו timestep. נדפה תחילה לכותב את הדיווחים שהם בלי description ורק בנות ב timestep.

לדוגמה, מימין ה detect ומשמאל הפלט

| detect ה | | הפלט | |
|---|---|---|---|
| 73 | A-B | 73-76 | A - B |
| 74 | A-B | 133-135 | C - D |
| 75 | A-B | | |
| 76 | A-B | | |
| 133 | C-D | | |
| 134 | C-D | | |
| 135 | C-D | | |

P - כמות החריגות בקובץ החריגות בעבור חיובי (Possitive)

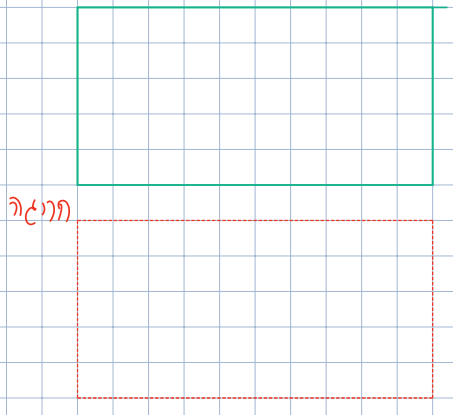
* בדוגמה לעיל P=3.

N - כמות ה time steps שבהם לא היתה חריגה (בקובץ החריגות)

* בדוגמה חישב N מתוך הדוגמה לעיל: N= n - 29 - 6 - 20

1+1001-1020 | 1+180-185 | 1+122-150 אם השורה שוב בקובץ החריגות test שקבלנו ללא הכותרת


קבל הדיווח הוא קבל שלפי ה detect שיש לנו.
קבל החריגה הוא קבל שלפי ה- CSV שקיבלנו מהאשתמש.




כלומר, בגלל שיש חפיפה שלפי סווחים בין החריגה לדיווח. אפשר יש,
נשמור במשתנה שסופר לנו כמה true positive שזה יש לנו.

# אבן דרך 3 – שרת גילוי חריגות

כיום ישנם שרתים מוכנים שמאד נוח להתממשק אליהם. אולם, כדי שההבנה שלנו תהיה עמוקה, אנו נממש שרת פשוט בכוחות עצמנו. מאוחר יותר תוכלו להשתמש בשרתים מוכנים וכבר יהיה לכם מושג טוב כיצד הם עובדים מאחורי הקלעים.

בשלב זה נרצה שלשרת שלנו יהיה Command Line Interface (CLI). כלומר, כאשר לקוח יתחבר, יופיעו לו תפריטים טקסטואליים שהשרת שלח ובאמצעותם תתבצע האינטראקציה בין השרת ללקוח.

הלקוח יוכל להעלות לשרת קובץ csv, לעדכן פרמטרים של האלגוריתם ולקבל בחזרה דו"ח חריגות שנתגלו. בנוסף הלקוח יוכל להזין היכן התרחשו החריגות בפועל ולקבל ניתוח של דיוק האלגוריתם על ה data ששלח.

השרת שלנו יצטרך לטפל במספר לקוחות במקביל ונצטרך אף להגביל את הכמות הזו כדי שהשרת לא יקרוס כתוצאה מעומס.

ייתכן ובשלב מאוחר יותר (למשל פת"מ 2) נרצה להחליף את ה CLI ולהפוך את השרת שלנו ל Service שניתן להפעלה ישרות מקוד הלקוח לפי סטנדרטים מודרניים.

**הערה חשובה**: משלב זה של הפרויקט אתם מקבלים בהדרגה יותר חופש ואחריות על ההחלטות בפרויקט. המשמעות היא שאגדיר לכם מה עליכם לממש ופחות אגדיר לכם איך לממש זאת.

נתחיל בכך שתקבלו את העיצוב בלבד, ועליכם לנתח ולגזור משמעויות לאימפלמנטציה, ולממש בהתאם.

## משימה א' CLI

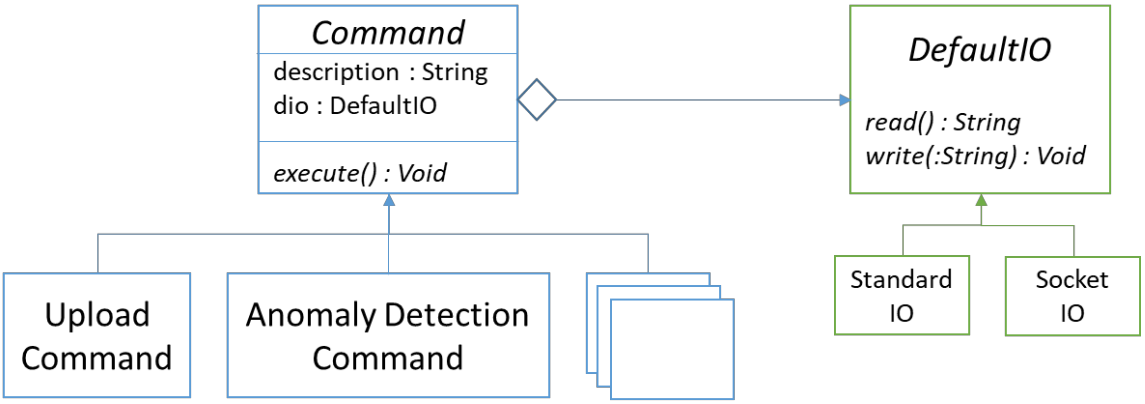
לשם מימוש ה CLI נשתמש בתבנית עיצוב שנקראת Command Pattern בה לכל פקודה במערכת שלנו יש מחלקה משלה מסוג Command. הטיפוס Command יכול להגדיר כל מה שרלוונטי לכל הפקודות במערכת שלנו, ובפרט פקודת execute אבסטרקטית עבור הפעלה. היתרונות הם:

- מכנה משותף פולימורפי לכל הפקודות
  - ניתן למשל להכניס את כל הפקודות למבנה נתונים
  - כגון מפה או מערך, ובהינתן ה key (או אינדקס) מיד לשלוף את הפקודה ולהפעילה
  - לרכז את כל הבקשות לפקודות המגיעות במקביל בתור \ תור עדיפויות
- פתוח להרחבה – ניתן להוסיף עוד מחלקות Command ע"פ הצורך \ לרשת פקודות קיימות
- ליצור הרכבות עם תבניות עיצוב אחרות, למשל עם Composite. אם למשל המחלקה MacroCommand תחזיק מערך פולימורפי של Commands אז כשנקרא ל execute שלה היא בתורה תקרא ל execute של כל פקודה במערך, שבתורן יכולות להיות פקודות רגילות (עלים) או גם MacroCommands.

אך היתרון העיצובי הוא החשוב ביותר – והוא ההפרדה בין יוזם הפקודה (invoker) לבין מי שהולך להיות מופעל (receiver). למשל אם יש 5 דרכים שונות ליזום את אותו הדבר (נניח פעולת הדבק) אז מכולן יהיה קישור לאותו אובייקט פקודה ויצרנו מקור אחד של אמת אם נרצה לשנות בעתיד משהו.

לעוד מידע תוכלו לצפות בסרטון 13.1

נגדיר את העיצוב הבא:



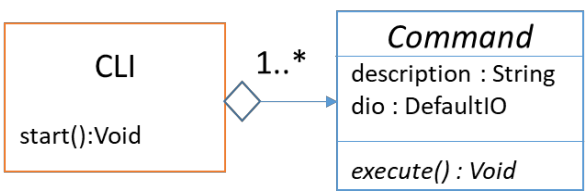

כפי שניתן לראות יש לנו מחלקה אבסטרקטית Command עם מתודה אבסטרקטית execute(). את המתודה הזו יצטרכו לממש כל היורשים – הפקודות השונות בתוכנית שלנו.

בנוסף לכל Command יש מחרוזת description. כדי ליצור תפריט למשל, נוכל להשתמש במערך של Command-ים ולעבור על כל Command ולהדפיס את ה description שלו. כאשר המשתמש יבחר אופציה $i$ נוכל ללכת ל Command ה i במערך ולקרוא ל execute שלה. הפקודה הזו בתורה תמשיך את האינטראקציה עם המשתמש לפי הצורך. היא אפילו יכולה בתורה להפעיל פקודות אחרות.

אולם, לא תמיד נרצה להדפיס למסך או לקרוא מהמקלדת... בהקשר שלנו נרצה להשתמש בעיצוב הזה בתוך שרת, כאשר את הקלט והפלט אנו מבצעים דרך socket-ים של תקשורת. לשם כך הגדרנו את הטיפוס המופשט DefaultIO שהיורשים שלו יצטרכו לממש בדרכם את המתודות הקריאה והכתיבה. כך נוכל להזין בזמן ריצה ל Command מימושים שונים של DefaultIO. בדומה לעיל, אם נרצה קלט-פלט סטנדרטי אז נזין לו את StandardIO ואילו אם נרצה באמצעי תקשורת אז נזין SocketIO. נשים לב שזה גם פתוח להרחבה, כי אם נרצה למשל לקרוא ולכתוב לקבצים אז נוכל להוסיף מימוש ל DefaultIO וכל ה Command-ים לא יצטרכו להשתנות ויעבדו בדיוק אותו הדבר.

**זכרו!** תרשים מחלקות ב UML אינו מהווה תחליף לקוד; הוא מכיל רק את מה שרלוונטי להבנת העיצוב. עליכם לגזור משמעויות נוספות שקשורות למימוש.

כעת צרו את המחלקה CLI עם המתודה void start().




**טיפ:** תחילה תעבדו עם standardIO. זה יהיה הרבה יותר נוח ונכון לדיבאג. כשהכל יעבוד ב CLI, תוכלו לממש את socketIO ואז ה CLI יזין אותו ל Command, ולראות שהכל עובד גם דרך ערוצי התקשורת.

כאשר מתודה זו תופעל יודפס ללקוח התפריט הבא:

```
Welcome to the Anomaly Detection Server.
Please choose an option:
1. upload a time series csv file
2. algorithm settings
3. detect anomalies
4. display results
5. upload anomalies and analyze results
6. exit
```

**הערה:** עבור כל הדפסה בפרויקט, ירידת שורה היא '\n' בלבד, ולא '\r\n' כמו במערכת ההפעלה חלונות.

על כל אופציה שהלקוח בוחר יופעל אובייקט Command מתאים שימשיך ע"פ הצורך את האינטראקציה עם הלקוח.

זכרו שהאינטראקציה צריכה להיעשות ע"י אובייקט ה DeafultIO כדי שמאוחר יותר תוכלו להחליף אותו עם קלט-פלט מבוסס תקשורת. כשתעשו את זה, מן הסתם תצטרכו לבנות גם צד לקוח, לעת עתה ניתן להניח לזה.

**אם הלקוח הקליד 1** ו enter, תינתן האפשרות ללקוח להקליד נתיב לקובץ csv לוקאלי אצלו במחשב, ולאחר לחיצה על enter הלקוח ישלח את הקובץ לשרת. בסיום ההעלאה השרת יכתוב חזרה ללקוח הודעת "upload complete".

זה צריך להיראות כך:

| | |
|---|---|
| Please upload your local train CSV file. | השרת מדפיס |
| C:\data\flightgear\flight1.csv | הלקוח מקליד שם קובץ |
| Upload complete. | השרת מדפיס |

תהליך זה יחזור על עצמו פעמיים, כאשר בפעם הראשונה מקבלים קובץ עבור אימון גלאי החריגות ובפעם השנייה קובץ עבור הבחינה שלו. בהתאמה, בפעם הראשונה תופיע המילה train ובשנייה test.

**<u>עבור צד השרת נגדיר את הפרוטוקול כך:</u>**

לאחר בחירה של 1 ע"י הלקוח, השרת כותב ללקוח "Please type ... CSV file" ואז יצפה לקרוא מהלקוח time series – כלומר, כותרות מופרדות בפסיק בשורה הראשונה, ואז בכל שורה לאחר מכן ערכים המופרדים בפסיק, בדיוק כפי שנראה קובץ ה csv. כאשר תופיע שורה ובה המילה "done" בלבד, ידע השרת שהסתיימה שליחת הקובץ ע"י הלקוח. השרת שומר בצד שלו את הקלט שקיבל מהלקוח כקובץ anomalyTrain.csv \ anomalyTest.csv בהתאמה לקובץ שהועלה ע"י הלקוח. השרת כותב ללקוח "Upload complete.".

**עבור צד הלקוח נגדיר את הפרוטוקול בהתאמה:**

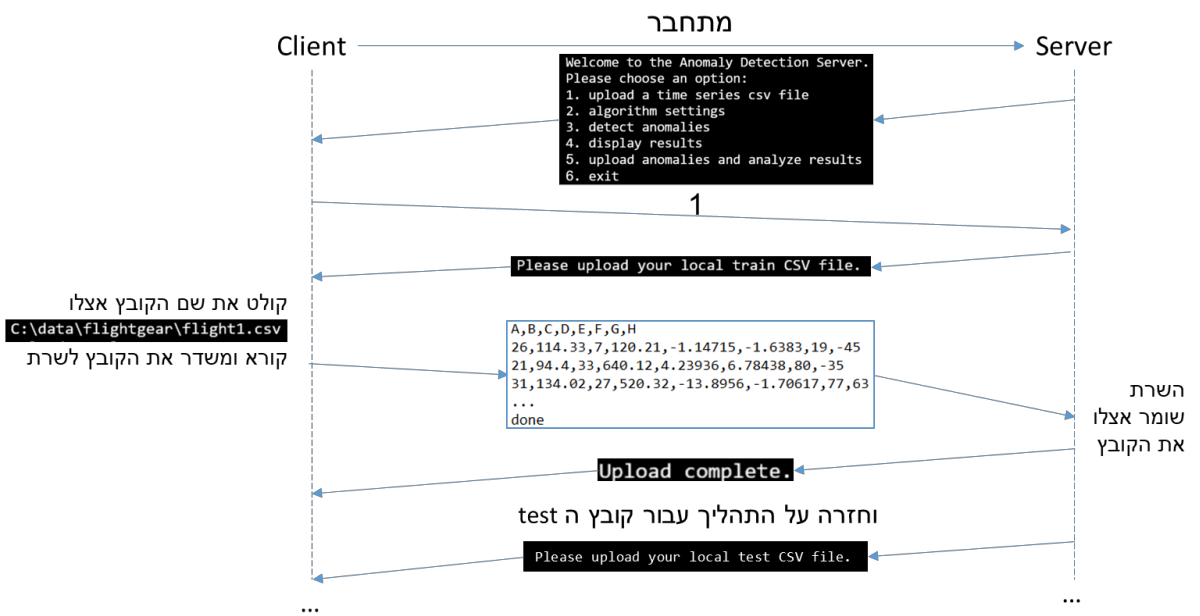
הלקוח בוחר 1 (ו enter), ומקבל מהשרת את השורה "Please ..CSV file". אז הלקוח קולט מהמשתמש (האדם שנמצא בצד הלקוח) נתיב ושם קובץ מלא עבור קובץ csv. קורא את קובץ ה csv ועל כל שורה שקרא הלקוח מהקובץ הוא שולח אותה לשרת. לאחר סיום קריאת הקובץ הלקוח כותב לשרת שורה ובה המילה "done" בלבד. הלקוח יקרא מהשרת את השורה "Upload complete".

למען הסר ספק, הקלט של נתיב הקובץ בצד הלקוח לא עובר בתקשורת אל השרת, אלא רק תוכן הקובץ.

**טיפ**: זה יכול להיות רעיון טוב להוסיף ל DefaultIO אפשרות להעלות \ להוריד קבצים (כמתודות דיפולטיביות התלויות במתודות הבסיסיות שהוגדרו).

לאחר סיום העלאת הקובץ השני יוצג שוב התפריט הראשי.

**תרשים התהליך עד כה:**




נשים לב שמנקודת מבטו של השרת הוא רואה:

```
1

A,B,C,D,E,F,G,H

26,114.33,7,120.21,-1.14715,-1.6383,19,-45

31,134.02,27,520.32,-13.8956,-1.70617,77,63

...

done
```

כלומר הוא אינו רואה את שם הקובץ שהלקוח הקליד לוקאלית אצלו.

הלקוח שלח 1 לשרת, קרא מהשרת מחרוזת .Please upload... file, קלט מהמשתמש אצלו את שם הקובץ, והתחיל לשלוח את תוכן הקובץ לשרת ולבסוף שלח לשרת את המחרוזת done.

אז נא לא להתבלבל, קלט הבדיקה האוטומטית מדמה בדיוק את זה, כלומר כשהשרת נבדק נשלח לו מה שהוא מצפה לראות כמו בדוגמה לעיל – כלומר ללא קלט שם הקובץ.

לעומת זאת המימוש של צד הלקוח (שאותו אינכם נדרשים לממש) ידרוש את קלט שם הקובץ, קריאתו ושליחתו.

יש 2 קבצים, test & Train

המטרה ← לבדוק הכיסוי על הסיסה התקינה.

סיסה תקינה

סיסה אחרת

**אם הלקוח בחר 2,** השרת יציג לו את סף הקורלציה והאפשרות להחליפו באופן הבא:

```
The current correlation threshold is 0.9
Type a new threshold
```

אם הוא בחר ערך תקין ולחץ enter אז הסף ישתנה. אם הוא לא בחר ערך בין 0 ל 1 אז יש לכתוב לו את ההודעה הבאה:

"please choose a value between 0 and 1."

ולחזור ולהציג שוב את האפשרות לשנות את הסף.

learnorma

לאחר בחירה של סף תקין יש לחזור לתפריט הראשי.

אותכי

**אם הלקוח בחר 3,** השרת יריץ את האלגוריתם על קובץ ה CSV שהועלה קודם לכן. בסוף הריצה השרת יכתוב "anomaly detection complete." ונחזור לתפריט הראשי.

**אם הלקוח בחר 4,** השרת ידפיס את דיווחי החריגות. לכל דיווח ההדפסה תהיה כך: ה time step, טאב, ה description ואז ירידת שורה. לבסוף יודפס "Done.". לאחר מכן יש לחזור לתפריט הראשי. לדוגמה:

```
73      A-B
75      C-D
8       E-F
Done.
```

**אם הלקוח בחר 5,** הוא יקליד שם מלא של קובץ חריגות. לאחר enter הלקוח יעלה את הקובץ לשרת, השרת ינתח את תוצאות האלגוריתם ויציג אותן ללקוח.

בדומה לפרוטוקול ההעלאה הקודם, המשתמש בצד הלקוח מקליד את שם הקובץ ואז הלקוח שולח לשרת שורה אחר שורה מהקובץ כאשר בסוף הוא כותב לשרת שורה ובה המילה "done" בלבד.

קובץ החריגות הוא קובץ טקסט בו בכל שורה יופיעו הזמנים שבהם אכן הופיעו חריגות (לפי סדר כרונולוגי). הפורמט הוא time step של תחילת החריגה, פסיק, time step של סיום החריגה. לדוגמה:

122,150
180,185
1001,1020
done

חריגה כוללת את זמן סיום החריגה, לדוגמה כל השורות 122 עד 150 כולל 150 מהוות שורות שבהן היתה חריגה. השרת ישווה את זמני הדיווחים של גלאי החריגות לזמנים הללו (השרת לא חייב לשמור את קובץ החריגות כקובץ בצד השרת).

תחילה, עלינו לאחד את דיווחי הגלאי שיש להם את אותו ה description ורציפות ב timestep. לדוגמה אם גלאי החריגות דיווח 15 דיווחים רציפים מ time step x ועד time step x+15 ולכולם אותו התיאור, אז נחשיב את זה כדיווח אחד רציף שהתחיל ב x והסתיים ב x+15 (כולל). את כמות החריגות (בקובץ החריגות) נסמן כ P (עבור positive). בדוגמה לעיל P=3. את כמות ה time steps שבהם לא היתה כל חריגה (בקובץ החריגות) נסמן כ N (עבור negative). בדוגמה לעיל, אם n זה מס' השורות בקלט (לא כולל הכותרת) אז N=n – 29 – 6 - 20

כעת נוכל למדוד את האלגוריתם:

- כל דיווח מחוץ לאחת ממסגרות הזמן בהן היו חריגות ייחשב כ false positive. נסמן כ FP.
- כל מסגרת זמן שבה היתה חריגה, ויש לה חיתוך גדול מ 0 עם זמן של דיווח חריגה ייחשב כ true positive. נסמן כ TP.
- ישנם מדדים נוספים שאין לנו בהם צורך כרגע, אך כדי להשלים את התמונה:
  - כל מסגרת זמן שבה היתה חריגה, ואין לי חיתוך עם אף דיווח – תיחשב כ false negative. נסמן כ FN.
  - כל זמן בו לא היתה חריגה ולא היה דיווח ייחשב כ true negative. נסמן כ TN.

המחשה ויזואלית:


נרצה לדווח שני מדדים:

True positive rate = TP / P – כמה מתוך כלל הדיווחים היו נכונים

False alarm rate = FP / N – יחס אזעקות השווא

האינטראקציה בין השרת ללקוח תיראה כך:

```
Please upload your local anomalies file.
C:\data\flightgear\flight1_anomalies.txt
Upload complete.
Analyzing...
True Positive Rate: 0.753
False Positive Rate: 0.12
```

השרת מדפיס
הלקוח מקליד
השרת מדפיס

את התוצאות יש להדפיס עם חיתוך של 3 ספרות אחרי הנקודה.

לאחר מכן נחזור לתפריט הראשי.

**אם הלקוח בחר 6,** תסתיים האינטראקציה בין השרת ללקוח.

כאמור, מומלץ לבדוק את כל ה CLI לוקאלית עם standardIO לפני שממשיכים למשימות הבאות.

## משימה ב' צד שרת

בדומה לאלמנטים הקודמים בפרויקט, גם כאן נרצה לשמור כלליות הקוד כך שיתאפשר reuse בעת הצורך. כשאנו כותבים צד שרת יש חלקים שחוזרים על עצמם מפרויקט לפרויקט ויש כמובן דברים שמשתנים. זה חשוב להפריד בין אלו כדי שנוכל לעשות reuse לקוד שאינו משתנה. לשם כך אסור לו להיות תלוי בקוד שכן צפוי להשתנות. צריך לנתק בין חלקים אלו – כלומר לעשות decupling

לאמוש SocketIO ← לנתק אות ה FileIO כך לשניהם

קונסטרקטור של SocketIO — יקבוף עם SocketInput

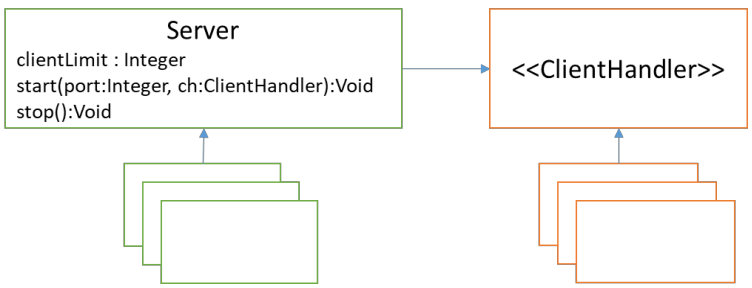
בצד השרת נרצה להפריד בין המנגנון שמפעיל את השרת, דהיינו מאזין על port כלשהו וממתין ללקוח, פותח ת'רד כדי להתנהל איתו במקביל, מגביל את כמות הת'רדים וכדומה, לבין השיחה עם הלקוח שתשתנה כמובן מפרויקט לפרויקט.

נשתמש בתבנית עיצוב בשם Bridge pattern כדי להפריד בין מימושים שונים של השרת לבין מימושים שונים לשיחה עם הלקוח.

צפו בהרצאה 9, וכן בסרטון 11.3 אודות התבנית.

כעת אתם מקבלים יותר חופש. הביטו בתרשים הכללי הבא, והפכו אותו לעיצוב שמתאים למימוש צד השרת.

לייצר server שמקבלים
את השרת, והוא מקבל אותו ה Port




בצד אחד של הגשר יש לנו את המחלקה Server. בצד השני יש לנו ממשק בשם ClientHandler שצריך להגדיר את הפונקציונאליות של טיפול בלקוח שהתחבר. במתודה ()start של Server נקבל מופע מהסוג של ClientHandler. כך, ניתן להרחיב את Server בצד אחד של הגשר בלי להיות תלויים בהרחבות השונות של ClientHandler. הלכה למעשה, נוכל לממש את Server פעם אחת, ובכל פרויקט להחליף לו ClinetHandler.

בנוסף, הערך של clientLimit יגביל את כמות הלקוחות שנטפל בהם במקביל. המתודה start תפעל את השרת על port כלשהו, ותפעיל את ה client handler שקבלה כדי להתכתב עם הלקוח שהתחבר. המתודה ()stop תסגור את השרת בצורה מסודרת. כלומר, לא יתקבלו לקוחות חדשים. כל הלקוחות שהתחברו יסיימו את הטיפול, ולאחר מכן כל המשאבים (ת'רדים, soket-ים) ייסגרו.

- עליכם להגדיר בכוחות עצמכם מה צריך להיות בממשק של Client Handler.
- עליכם לממש את Client Handler במחלקה בשם AnomalyDetectionHandler
- עליכם לשבץ בצורה נכונה במחלקה זו את השימוש ב CLI, ה Command-ים שלו, ובפרט את ה DfaultIO שלו שצריך עתה להתכתב באמצעות ה Socket-ים שהשרת פתח.
- השרת שלכם צריך לטפל בלקוחות אחד אחרי השני ולא במקביל. אולם,
  - המתודה start כן צריכה לרוץ ברקע כפי שראינו בשיעור
  - בפת"מ 2 תוסיפו מימוש ל Server שמטפל בכמה לקוחות במקביל.
- הפעם, כאשר הלקוח בחר exit השרת יכתוב לו את המילה "bye" בשורה משלה.

במסמך ההגשות מפורט תאריך היעד להגשה, כיצד להגיש ולאן.

הבדיקה בודקת את ה *CLI* בנפרד, ולאחר מכן באמצעות לקוח שיתחבר לשרת שלכם.

בהצלחה!

פרומו לאבן דרך 4 בסמסטר ב: ☺