תרגיל מספר 4

תאריך הגשה: 22.01.23

בתרגיל זה השרת שלנו יצטרך לטפל במספר לקוחות במקביל. כלומר, ברגע נתון, השרת יכול לדבר עם מספר לקוחות שונים בו זמנית.

כאשר בתרגיל כתוב לקוח - הכוונה לclient המדבר עם server, כלומר, מדובר על הקוד שאתם כותבים. כאשר כתוב משתמש - הכוונה ל"בן אדם" אשר מתקשר בעזרת הלקוח מול השרת.

הלקוח (לא המשתמש) בתרגיל זה הוא לרוב "טיפש" - כלומר, שולח מידע לשרת שהמשתמש הזין ומדפיס למשתמש מידע שהשרת שלח לו.

כאשר לקוח מתחבר לשרת, השרת ישלח ללקוח טקסט אשר יפרט את הפונק' של השרת.
הלקוח ידפיס למסך את הטקסט שקיבל מהשרת.

ספציפית, על השרת לשלוח ללקוח את הטקסט הבא:

```
Welcome to the KNN Classifier Server. Please choose an option:
1. upload an unclassified csv data file
2. algorithm settings
3. classify data
4. display results
5. download results
```

```
8. exit
```

אם המשתמש הקליד 1 וenter,תינתן האפשרות למשתמש להקליד נתיב לקובץ csv לוקאלי אצלו במחשב, ולאחר לחיצה על enter הלקוח ישלח את תוכן הקובץ לשרת. בסיום השליחה השרת ישלח בחזרה ללקוח הודעת "Upload complete.". אם הנתיב לא חוקי יודפס "invalid input" (וחוזרים לתפריט הראשי).
זה צריך להיראות כך:

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

תהליך זה יחזור על עצמו פעמיים, כאשר בפעם הראשונה מקבלים קובץ עבור אימון המסווג ובפעם השנייה קובץ עבור הבחינה שלו. בהתאמה, בפעם הראשונה יופיע:
Please upload your local train CSV file.

ובשנייה:

Please upload your local test CSV file.

לאחר סיום העלאת הקובץ השני יוצג שוב התפריט הראשי.

אם המשתמש בחר 2, השרת ישלח את ערכי פרמטרי המסווג העדכניים שהם ערך הפרמטר K ומטריקת המרחק הנוכחית.

```
The current KNN parameters are: K = 5, distance metric = EUC
```

אם המשתמש לחץ enter, יש להשאיר את הפרמטרים ללא שינוי (וודאו שהערכים מאותחלים לערכים שבדוגמא לעיל). אחרת, יוכל המשתמש להזין ערכים חדשים מופרדים ברווח כאשר K יכול לקבל טווח ערכים של מספרים שלמים ומטריקת המרחק יכולה לקבל את הערכים שראינו בתרגיל הקודם (5 מטריקות).
אם הכל תקין - חוזרים לתפריט הראשי. אם המשתמש הזין ערך לא חוקי - תוחזר הודעת השגיאה: **invalid value for K** או **invalid value for metric** (אם צריך - יש להדפיס את שניהם).
לאחר מכן - חוזרים לתפריט הראשי (ללא שינוי - גם אם אחד הפרמטרים היה חוקי).

אם המשתמש בחר 3, השרת יריץ את האלגוריתם על קבצי הCSV שהועלו קודם לכן. בסוף הריצה השרת יכתוב "classifying data complete" ונחזור לתפריט הראשי.
אם עוד לא הועלו קבצים יש להדפיס "please upload data".

אם המשתמש בחר 4, השרת יחזיר את רשימת הסיווגים. לכל סיווג ההדפסה תהיה כך: מספר השורה בקובץ הtest, טאב, הסיווג ואז ירידת שורה. לבסוף יודפס ".Done" לאחר enter של המשתמש יש לחזור לתפריט הראשי.

```
1  A
2  B
3  C
4  A
Done.
```

כאשרA, B, C הן מחלקות הסיווג.
אם עוד לא הועלו קבצים יש להדפיס "please upload data".
אם עוד לא סווגו הקבצים ש להדפיס "please classify the data".

אם המשתמש בחר 5, ההתנהגות תהיה דומה לזו של אופציה 4, רק שבמקום להדפיס את התוצאות המשתמש יזין נתיב ליצירת הקובץ אצלו לוקאלית ולשם הלקוח ישמור את התוצאות, בדיוק באותו הפורמט (ללא Done).

<u>שימו לב:</u> בשונה משאר הפקודות, בפקודה זאת לאחר enter של המשתמש (שהזין נתיב לקובץ) יופיע מיד שוב התפריט הראשי והמשתמש יוכל לשלוח פקודות נוספות מיד, גם אם הורדת הקובץ טרם הסתיימה (כלומר - תהליך זה יקרה בthread נפרד).

אם המשתמש בחר 8, תסתיים האינטראקציה בין השרת ללקוח (לא לשכוח לסגור ולשחרר חיבור ומשאבים).

חובה לממש את התרגיל בעזרת תבנית העיצוב Command, כפי שמתואר בנספח לתרגיל.

צד שרת

כל התקשורת בין השרת ללקוח תתקיים באמצעות פרוטוקול TCP.
עליכם לממש באמצעות מקביליות אפשרות של טיפול במספר לקוחות במקביל. צורת המימוש היא לבחירתכם בין אם תבחרו להריץ תרד חדש עבור כל לקוח שיתחבר או מנגנון אחר.

צד לקוח

צרו צד לקוח שיודע לתקשר עם השרת שיצרתם. המקום היחיד בו יש צורך להשתמש בthreads הוא רק בעת קבלת קובץ שמירה (אופציה 5 בתפריט) מעבר לכך - אין צורך בשימוש בת'רדים או במקביליות כלשהי בצד לקוח.
על מנת שהלקוח יוכל לשלוח פקודות במקביל לקבלת נתונים מהשרת עליכם לממש זאת בצורה מקבילית. בה תרד אחד אצל הלקוח יהיה מיועד לשליחה ותרד נוסף יהיה מיועד להאזנה וקבלת הודעות. בצורה זו הלקוח תמיד יכול לשלוח הודעות נוספות לשרת ולא תלוי בקצב עבודת השרת.

דאטא

אתם יכולים להניח שקבצי הדאטא שהמשתמש מזין יהיו מסוג הקבצים שאיתם עבדתם עד כה. כלומר, בדומה לדאטא האירוסים וכו', שצורף כדוגמאות לתרגילים הקודמים, העמודות יהוו עמודות ערכי מאפיינים והעמודה האחרונה היא עמודת שמות המחלקות, כאשר כל שורה מייצגת פריט נפרד.

הנחיות:

<u>על התרגיל להתקמפל ולרוץ על שרתי האוניברסיטה (U2 / Planet).</u> שימו לב שבתרגיל זה מכיוון שאנו משתמשים בthreads הנושא הרבה יותר עדין, הlinker של linux מאוד עדין עם הספריה הזאת. אין להשתמש בספריות אלגוריתמים מוכנים אלא לממש בעצמכם את האלגוריתמים. יש לממש במבנה קוד תקין, כלומר הצהרות בקובץ h בנפרד מהמימוש בקבצי cpp. יש להקפיד לתעד את הקוד לכל אורכו. במידה ושם המחלקה/משתנה/פונק׳ מורכב ממספר מילים, האות הראשונה של כל מילה היא אות גדולה. שמות של מחלקות חייבים להתחיל באות גדולה. שמות של משתנים ופונק׳ מתחילים באות קטנה.

יש להגיש בגיט קובץ MakeFile כך שנוכל לקמפל בעזרת הפקודה make בלבד (גם את השרת וגם את הלקוח) - **יש לוודא שהקמפול פועל על שרתי האוניברסיטה**.

חובה לעבוד בגיט לאורך כל התרגיל. כדי להגיש את התרגיל ניתן ליצור repo חדש ולהוסיף אותנו כcollaborator, או לפתוח בראנץ' חדש מהתרגיל הקודם לתרגיל זה. בשיטה השניה יש לשים לב לא לדחוף שינויים לתרגיל הראשון עד לאחר הבדיקה!
כל סטודנט עובד מהמחשב שלו ומהמשתמש הנפרד שלו. לא ניתן לעבוד ביחד מאותו מחשב או מאותו משתמש או ללא גיט. יש לעבוד בשיטת feature branches. חובה לתכנן ולחלק את המשימות בGithub project and issues כפי שהוצג בכיתה, ולתחזק אותן בהתאם להתקדמות בפרויקט. אלה דרישות מהותיות ומשקלם בתרגיל משמעותי.
חובה להגיש את התרגיל בזוגות. הגשה ביחידים (מכל סיבה שהיא), תגרור תקרה של ציון מקסימלי של 75 בכל תרגיל שיוגש ביחידים. לא ניתן יהיה לקבל ציון מעל 75 בתרגיל שהוגש ביחידים.

את התרגיל יש להגיש על ידי הגשת קובץ טקסט בשם details.txt עם שמות ות.ז. של המגישים וקישור לגיט.
שימו לב, חובה על הקובץ להיות בפורמט הבא:

Israel Israeli 123456789
Israela Israeli 012345678
LINK TO GITHUB HERE

בלי רווחים נוספים, בלי שורות נוספות, ובשפה האנגלית בלבד. אי הגשה של קובץ ה details.txt הנ״ל או הגשתו באופן שונה ממה שהוגדר, **<u>תגרור הורדה של 20 נקודות בציון התרגיל.</u>**

שאלות לגבי התרגיל יש לשאול בפורום בלבד. פניות בנושאים פרטיים יש לשלוח למייל הקורס. העתקות יבדקו ע״י מערכת אוטומטית, ויטופלו ישירות על ידי המחלקה.

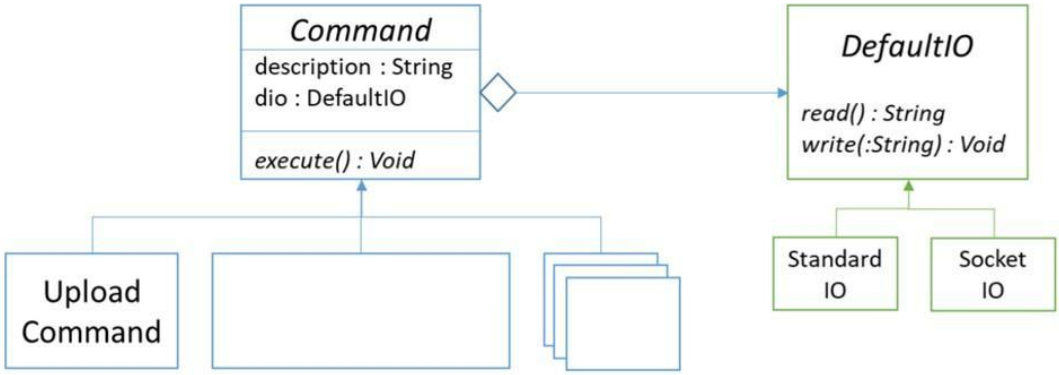
על הגיט להכיל קובץ README שמסביר איך לקמפל את הקוד וכן מסביר את המימוש שלכם ואילו אופטימיזציות הכנסתם אם בכלל. שימו לב שאם לא נצליח לקמפל ולהריץ את הקוד- לא נוכל לבדוק, וודאו כי ההוראות ברורות ופשוטות. חשוב שתהיה דוקומנטציה מלווה גם בקוד עצמו. החשיבות של README בגיט היא עצומה, ויש לדאוג לפירוט הרב ביותר שניתן לתת - דבר זה יובא לידי ביטוי בציון התרגיל.

בהצלחה!

נספח.

נשתמש בתבנית עיצוב שנקראת Command, בה לכל פקודה בתוכנה שלנו יש מחלקה משלה מסוג Command. המחלקה Command יכולה להגדיר כל מה שרלוונטי לכל הפקודות במערכת שלנו, ובפרט פקודת execute אבסטרקטית עבור הפעלה. היתרונות בתבנית עיצוב זו הם:

- מכנה משותף פולימורפי לכל הפקודות:
  - ניתן למשל להכניס את כל הפקודות למבנה נתונים כגון מפה או מערך, ובהינתן מפתח או אינדקס מיד לשלוף את הפקודה ולהפעילה.
  - ניתן לרכז את כל הבקשות לפקודות המגיעות במקביל בתור \ תור עדיפויות.
- פתוח להרחבה – ניתן להוסיף עוד מחלקות Command ע"פ הצורך \ לרשת פקודות קיימות
- היתרון העיצובי (החשוב ביותר): ההפרדה בין יוזם הפקודה (invoker) לבין מי שהולך להיות מופעל receiver. למשל אם יש 5 דרכים שונות ליזום את אותו הדבר (נניח פעולת הדבק – ניתן ליזום ע"י מקש ימני והדבק או ע"י ctrl+p) אז מכולן יהיה קישור לאותו אובייקט פקודה ויצרנו מקור אחד אם נרצה לשנות בעתיד משהו.



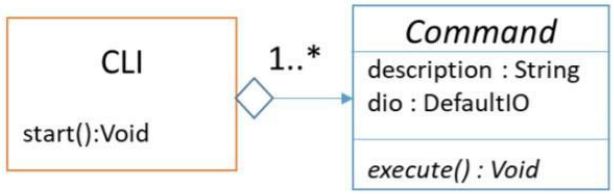
כפי שניתן לראות יש לנו מחלקה אבסטרקטית Command עם מתודה אבסטרקטית ()execute. את המתודה הזו יצטרכו לממש כל היורשים, במקרה שלנו - הפקודות השונות בתוכנית.
בנוסף לכל Command יש מחרוזת description. כדי ליצור תפריט למשל, נוכל להשתמש במערך של Command** ולעבור על כל Command* ולהדפיס את הdescription שלו. כאשר המשתמש יבחר באופציה i נוכל ללכת לCommand במקום ה-i במערך ולקרוא לexecute שלה. הפקודה הזו בתורה תמשיך את האינטראקציה עם המשתמש לפי הצורך. היא אפילו יכולה בתורה להפעיל פקודות אחרות.

נרצה לנתק את התלות בין הפקודה לבין מקור הקלט/פלט. הרי, לא בהכרח שנרצה להדפיס למסך (standard output) או לקרוא מהמקלדת (standard input). נרצה להשתמש בעיצוב הזה בתוך השרת, 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 לCommand 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 לDefaultIO וכל הCommand-ים לא יצטרכו להשתנות ויעבדו בדיוק אותו הדבר.

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

אחד האתגרים למשל, הוא כיצד להעביר מידע בין אובייקטי הCommand בלי להשתמש במשתנים גלובאליים או סטטיים. עליכם לחשוב בכוחות עצמכם על פתרון מונחה עצמים לכך.

צרו את המחלקה CLI עם המתודה ()start



טיפ: תחילה תעבדו עם standardIO. זה יהיה הרבה יותר נוח לדיבאג. כשהכל יעבוד בCLI, תוכלו לממש אתsocketIO ואז הCLI יזין אותו לCommand, ואז לראות שהכל עובד גם דרך ערוצי התקשורת. (אין חובה להגיש או ליצור את standardIO אבל זה יקל משמעותית על תהליך הפיתוח).

על כל פקודה שהלקוח שולח לשרת (1-5) יופעל אובייקט Command מתאים שימשיך ע"פ הצורך את האינטראקציה עם הלקוח. זכרו שהאינטראקציה צריכה להיעשות ע"י אובייקט הDeafultIO כדי שמאוחר יותר תוכלו להחליף אותו עם קלט-פלט מבוסס תקשורת. כשתעשו את זה, תצטרכו לבנות גם צד לקוח.

כאמור, מומלץ לבדוק לוקאלית עם standardIO לפני שממשיכים למימוש מעל הרשת.