236703 – תכנות מונחה עצמים, חורף 2019-20

תרגיל בית 2 – היכרות עם Java

כללי

1. מועד ההגשה : **3/12/2019** בשעה **23:59**
2. מטרת התרגיל היא היכרות עם Java, עבודה עם Comparable/Comparator, Streams, Exceptions, Collections, ומימוש ממשקים.
3. קראו היטב את ההוראות, במסמך זה ובקוד שניתן לכם.
4. אחראי על התרגיל: דניאל. שאלות על התרגיל יש לשלוח במייל danytnt@campus עם הנושא: "HW2 236703". שאלות אדמיניסטרטיביות, כגון מילואים, יש לשלוח לנתן, תחת אותה כותרת.
5. הקפידו על קוד ברור, קריא ומתועד ברמה סבירה. עליכם לתעד כל חלק שאינו טריוויאלי בקוד שלכם.
6. מהירות ביצוע אינה נושא מרכזי בתרגילי הבית בקורס. בכל מקרה של התלבטות בין פשטות לבין ביצועים, העדיפו את המימוש הפשוט.
7. הימנעו משכפול קוד והשתמשו במידת האפשר בקוד שכבר מימשתם.
8. כדי להימנע מטעויות, אנא עיינו ברשימת ה FAQ המתפרסמת באתר באופן שוטף.

מבוא

בתרגיל זה תידרשו לממש רשת חברתית של פרופסורים רעבים בטכניון, המאפשרת להם ליצור חברויות אחד עם השני, לדרג בתי בוריטו בקרבה לטכניון, לקבל המלצות מחברים על בתי בוריטו (מעתה ייקראו "מסעדות") ועוד.

חלק א'

בחלק זה נממש את בסיס המערכת: פרופסורים ומסעדות.

`CasaDeBurritoImpl`

מחלקה זו תממש את ההתנהגות של הממשק `CasaDeBurrito` אשר סופק לכם. מחלקה זו מייצגת מסעדה בודדת. הממשק `CasaDeBurrito` מרחיב את הממשק `Comparable<CasaDeBurrito>` המאפשר השוואה בין אובייקטים בעזרת המתודה `compareTo`. אופי המימוש יתואר בהמשך. המחלקה תספק את ההתנהגות הבאה:

- CasaDeBurritoImpl(int id, String name, int dist, Set<String> menu) - בנאי המקבל את מזהה המסעדה, את שמה, את מרחקה מהטכניון וקבוצה של מנות בתפריט המסעדה (שמות של מנות). ניתן להניח שיתקבלו פרמטרים חוקיים (מספרים אי שליליים, תפריט לא ריק וכו').

- getId() – מחזירה את מזהה המסעדה.

- getName() – מחזירה את שם המסעדה.

- distance() - מחזירה את מרחק המסעדה מהטכניון.

- isRatedBy(Profesor p) – מחזירה true אמ"מ המסעדה הנוכחית דורגה ע"י הפרופסור p.

- rate(Profesor p, int r) - הפרופסור p מדרג את המסעדה. דירוג הוא מספר שלם בתחום [0,...,5]. אם r אינו דירוג חוקי, תיזרק חריגת RateRangeException. אם הפרופסור כבר דירג את המסעדה, דירוגו מתעדכן. על המתודה להחזיר את המופע של האובייקט (this) כדי לאפשר שרשור פעולות.

- numberOfRates() - מחזירה את מספר הדירוגים של המסעדה.

- averageRating() - מחזירה את ממוצע הדירוג של המסעדה. אם אין כלל דירוגים, המתודה תחזיר 0 .

- equals(Object o) - יש לדרוס את המתודה equals שהוגדרה לראשונה ב-Object , כפי שנלמד בתרגול. מסעדות יחשבו שוות אמ"מ יש להן אותו מספר מזהה.

- toString() - מחזירה מחרוזת המתארת את המסעדה - מוסבר בהמשך.

- compareTo(CasaDeBurrito c) - משווה את המסעדה עם המסעדה c, לפי הסדר הטבעי שיוגדר בהמשך.

ProfesorImpl

מחלקה זו תממש את ההתנהגות של הממשק Profesor אשר סופק לכם (**שימו לב שהשם נכתב בספרדית ולא באנגלית לאורך התרגיל**). מחלקה זו מייצגת פרופסור בודד. בנוסף, הממשק Profesor מרחיב את הממשק Comparable<Profesor> המאפשר השוואה בין אובייקטים בעזרת המתודה compareTo. אופי המימוש מתואר בהמשך. המחלקה תספק את ההתנהגות הבאה:

- ProfesorImpl(int id, String name) - בנאי המקבל מספר זהות ושמו של הפרופסור ומאתחל פרופסור עם ערכים אלו. ניתן להניח שיתקבלו פרמטרים חוקיים.

- getId() – מחזירה את מזהה הפרופסור.

- favorite(CasaDeBurrito c) - המסעדה c **נהיית** מועדפת ע"י הפרופסור. אם המסעדה אינה מדורגת ע"י הפרופסור, יש לזרוק חריגה מסוג UnratedFavoriteCasaDeBurritoException. המתודה תחזיר את המופע של האובייקט על מנת לאפשר שרשור של פעולות.

- favorites() - מחזירה את המסעדות המועדפות ע"י הפרופסור. על שינויים באוסף זה לא להשפיע על אוסף המסעדות המועדפות על הפרופסור.

- addFriend(Profesor p) - מוסיפה קשר חברות ברשת החברתית אל הפרופסור p. קשר החברות **אינו סימטרי**; המשמעות של p1.addFriend(p2) היא ש-p1 חבר של p2, אבל לא בהכרח להפך. קשר "חברות עצמית" אינו חוקי. במקרה כזה יש לזרוק חריגה מסוג SameProfesorException. אם קיים כבר קשר חברות בינם, יש לזרוק חריגה מסוג ConnectionAlreadyExistsException. המתודה תחזיר את המופע של האובייקט על מנת לאפשר שרשור של פעולות.
- getFriends() - מחזירה אוסף המכיל את חבריו של הפרופסור. <u>על שינויים באוסף זה לא להשפיע על אוסף החברים של הפרופסור</u>, כלומר, הוספת פרופסור אחר לאוסף זה לא תהפוך אותו לחבר של מקבל ההודעה.

- filteredFriends(Predicate<Profesor> p) – מחזירה אוסף המכיל את חבריו של הפרופסור שעונים על הפרדיקט p. על שינויים באוסף זה לא להשפיע על אוסף החברים של הפרופסור.

- filterAndSortFavorites(Comparator<CasaDeBurrito> c, Predicate<CasaDeBurrito> p **- מחזירה אוסף של כל המסעדות המועדפות ע"י הפרופסור, אשר עונות על תנאי הפרדיקט** p**. סדר המעבר על האוסף המוחזר הוא לפי ה** Comparator c**. כלומר,** זהו לא הסדר הטבעי המוגדר ע"י **compareTo!**

- favoritesByRating(int r) - מחזירה אוסף של כל המסעדות המועדפות ע"י הפרופסור, **עם דירוג ממוצע (של המסעדה) גבוה מ (או שווה ל) r**. סדר המעבר על האוסף המוחזר הוא לפי דירוג מסעדות אלו בסדר יורד. עבור מסעדות עם דירוג זהה, הסידור המשני יהיה לפי מרחק מהטכניון, בסדר עולה. סידור שלישי (עבור מסעדות שנמצאות במרחק זהה, ובעלות דירוג זהה) יהיה לפי מספר מזהה בסדר עולה. **כלומר,** זהו לא הסדר הטבעי המוגדר ע"י **compareTo**!

- favoritesByDist(int r) - מחזירה אוסף של כל המסעדות המועדפות ע"י הפרופסור, **אשר נמצאות במרחק קצר מ (או שווה ל) r**. סדר המעבר על האוסף המוחזר הוא לפי המרחק של המסעדות מהטכניון לפי סדר עולה. עבור מסעדות שנמצאות במרחק שווה מהטכניון, סידור משני של המסעדות יהיה לפי דירוג ממוצע בסדר יורד, וסידור שלישי (עבור מסעדות שנמצאות במרחק זהה, ובעלות דירוג ממוצע זהה) יהיה לפי מספר מזהה בסדר עולה. **כלומר,** זהו לא הסדר הטבעי המוגדר ע"י **compareTo!**

- **רמז**: ניתן להשתמש במתודה `filterAndSortFavorites` **למימוש** `favoritesByRating`,`favoritesByDist`. כמו כן, היעזרו בפעולות `filter`,`sorted` של שטפים (`Streams`).

- `equals(Object o)` - יש לדרוס את המתודה `equals` שהוגדרה לראשונה ב `Object`, כפי שנלמד בתרגול. שני פרופסורים יחשבו שווים אמ"מ יש להם אותו מספר מזהה.

- `toString()` - מחזירה מחרוזת המתארת את המסעדה - מוסבר בהמשך.

- `compareTo(Profesor p)` - משווה את הפרופסור עם פרופסור p, לפי הסדר הטבעי שיוגדר בהמשך.

הסדר הטבעי: יש לדרוס את המתודה `compareTo`, המוגדרת בממשק `Comparable` כפי שראיתם בכיתה. המתודה מגדירה יחס סדר בין הפרופסור/המסעדה הנוכחי/ת לפרופסור/למסעדה שהתקבל/ה כפרמטר. על המתודה להחזיר ערך שלילי אם הפרופסור/המסעדה הנוכחי/ת בעל/ת מזהה קטן משל הפרופסור/המסעדה שהתקבל/ה כפרמטר, חיובי אם להיפך, ו-0 אם המזהים שווים.

חלק ב' - מימוש הממשק `CartelDeNachos`

בחלק זה נממש את המערכת הכללית. במערכת נוכל להוסיף פרופסורים, להגדיר קשרי חברויות בין פרופסורים, ולקבל מידע על קשרים בין פרופסורים, ועל המסעדות המועדפות עליהם. קשר החברות בין פרופסורים:

- **אינו רפלקסיבי**: A לא יכול להיות חבר של עצמו.
- **סימטרי**: אם A חבר של B, אז גם B חבר של A. **בשונה מ- `Profesor::addFriend`**.
- **אינו טרנזיטיבי**: אם A חבר של B ו-B חבר של C, <u>לא גורר</u> ש- A בהכרח חבר של C.

בהמשך נראה שהמערכת מדמה גרף של חברויות בו כל צומת הוא פרופסור, והקשתות הן קשרי החברות בין הפרופסורים. נשתמש באבחנה זו בהמשך.

`CartelDeNachosImpl`

המחלקה תספק את ההתנהגות הבאה:

- `CartelDeNachosImpl()` - מאתחל את המערכת.

- `joinCartel(int id, String name)` - מקבלת נתונים של פרופסור, מייצרת מופע של `Profesor` בהתאם, מוסיפה אותו למערכת ולבסוף מחזירה את המופע שיצרה. אם קיים כבר פרופסור עם אותו מזהה במערכת, יש לזרוק חריגה `ProfesorAlreadyInSystemException`.

- addCasaDeBurrito(int id, String name, int dist, Set<String> menu) - מקבלת נתונים של מסעדה, מייצרת מופע של CasaDeBurrito בהתאם, מוסיפה אותה למערכת ולבסוף מחזירה את המופע שיצרה. אם קיימת כבר מסעדה עם אותו מזהה במערכת, יש לזרוק חריגה CasaDeBurritoAlreadyInSystemException.

- registeredProfesores() - מחזירה אוסף של הפרופסורים שבמערכת. על שינויים באוסף זה לא להשפיע על המערכת.

- registeredCasasDeBurrito() - מחזירה אוסף של המסעדות שבמערכת. על שינויים באוסף זה לא להשפיע על המערכת.

- getProfesor(int id) - מחזירה רפרנס ל-Profesor שבמערכת עם מספר המזהה id.
אם לא קיים פרופסור כזה במערכת, יש לזרוק חריגת ProfesorNotInSystemException.

- getCasaDeBurrito(int id) - מחזירה רפרנס ל-CasaDeBurrito שבמערכת עם מספר המזהה id. אם לא קיימת מסעדה כזו במערכת, יש לזרוק חריגת CasaDeBurritoNotInSystemException.

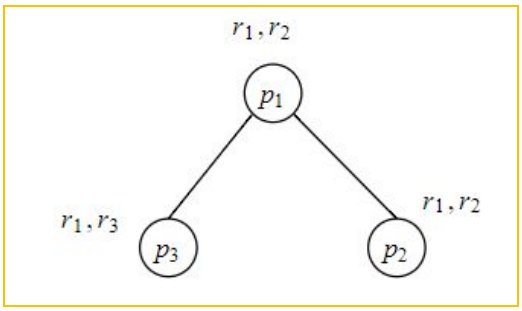
- addConnection(Profesor p1, Profesor p2) - מוסיפה קשר חברות (סימטרי) בין שני פרופסורים רעבים במערכת. אם אחד מהם לא קיים במערכת, יש לזרוק חריגת ProfesorNotInSystemException. אם שני הפרמטרים מייצגים אותו פרופסור, יש לזרוק חריגת SameProfesorException. אם החברות כבר קיימת, יש לזרוק ConnectionAlreadyExistsException. המתודה תחזיר את המופע של האובייקט על מנת לאפשר שרשור של פעולות.

- favoritesByRating(Profesor p) - מחזירה אוסף של מסעדות, אשר יכלול את כל המסעדות (הנמצאות במערכת) המועדפות על כל חבריו של הפרופסור p. על האיטרציה ב-stream הנוצר מהאוסף לעמוד בדרישות הבאות:
  - סדר המעבר על החברים הוא לפי סדר עולה של מספרי זהות.
  - עבור כל חבר, סדר המעבר על המסעדות המועדפות הוא לפי דירוג ממוצע (של המסעדה), בסדר יורד. עבור מסעדות עם דירוג ממוצע זהה, הסידור הוא לפי מרחק מהטכניון, בסדר עולה. סידור שלישי יהיה לפי מספר מזהה, בסדר עולה.
  - יש לעבור על כל המסעדות המועדפות של חבר אחד לפני המעבר לחבר הבא.
  - אין כפילויות באוסף; אם חבר מעדיף מסעדה R, וחבר אחר גם כן מעדיף את R, אזי R מופיע בסדר פעם אחת בדיוק (בנוסף לשאר המסעדות המועדפות ע"י החבר הראשון).
  - האוסף מכיל מסעדות מועדפות של חברים **ישירים** של הפרופסור בלבד (קשר חברות במרחק 1). כלומר, לא יופיעו באוסף מסעדות המועדפות על הפרופסור עצמו (קשר חברות במרחק 0) ולא על חברים של חברי הפרופסור. (קשר חברות במרחק 2 לפחות). אם הפרופסור לא נמצא במערכת, יש לזרוק חריגה מסוג ProfesorNotInSystemException.
- favoritesByDist(Profesor p) - מחזירה אוסף של מסעדות, אשר יכלול את המסעדות המועדפות של חבריו של הפרופסור p, הנמצאות במערכת. על האיטרציה ב-stream הנוצר מהאוסף לעמוד בדרישות הבאות:
  - סדר המעבר על החברים הוא לפי סדר עולה של מספרי זהות.

- עבור כל חבר, סדר המעבר על המסעדות המועדפות הוא לפי מרחק מהטכניון, בסדר עולה. עבור מסעדות במרחק זהה, הסידור המשני יהיה לפי דירוג ממוצע בסדר יורד. סידור שלישי יהיה לפי מספר מזהה, בסדר עולה.
- יש לעבור על כל המסעדות המועדפות של חבר אחד לפני המעבר לחבר הבא.
- אין כפילויות באוסף; אם חבר מעדיף מסעדה R, וחבר אחר גם כן מעדיף את R, אזי R מופיע בסדר פעם אחת בדיוק (בנוסף לשאר המסעדות המועדפות ע"י החבר הראשון).
- האוסף מכיל מסעדות מועדפות של חברים **ישירים** של הפרופסור בלבד (קשר חברות במרחק 1). כלומר, לא יופיעו באוסף מסעדות המועדפות על הפרופסור עצמו (קשר חברות במרחק 0) ולא על חברים של חברי הפרופסור. (קשר חברות במרחק 2 לפחות). אם הפרופסור לא נמצא במערכת, יש לזרוק חריגה מסוגProfesorNotInSystemException.

- `getRecommendation(Profesor p, CasaDeBurrito c, int t)` - נאמר כי מסעדה היא ***מומלצת-t*** ע"י פרופסור, אם הוא נמצא במרחק <u>לכל היותר</u> t קשרי חברויות מפרופסור אחר המעדיף מסעדה זו. המתודה מחזירה true אמ"מ המסעדה c מומלצת-t ע"י הפרופסור p. אם p לא נמצא במערכת, תיזרק חריגת ProfesorNotInSystemException. אם c לא נמצאת במערכת, תיזרק חריגת CasaDeBurritoNotInSystemException. אם t שלילי, תיזרק חריגת ImpossibleConnectionException.
  - מספר קשרי החברויות בין פרופסור לעצמו מוגדר להיות 0.
  - יש להזהר מקריאות רקורסיביות עמוקות בלתי נשלטות (לולאות אינסופיות).
  - רמז: חשבו על הדרכים למעבר על גרפים.

- (getMostPopularRestaurantsIds – נאמר ש***פרופסור תורם t נקודות למסעדה***, אם יש לו t חברים (לא כולל עצמו) שמסעדה זו מועדפת עליהם. כמות הנקודות הכוללת של מסעדה במערכת היא סכום התרומות של כל הפרופסורים הנמצאים במערכת.
המתודה תחזיר את רשימת המזהים של המסעדות הפופולריות ביותר (ניקוד מקסימלי במערכת).
הרשימה תהיה ממוינת בסדר עולה (לפי מזהה המסעדה).
<u>שימו לב</u>: אם מסעדה לא מועדפת על אף פרופסור, הניקוד שלה הוא 0.
דוגמה: p1 תורם 2 נקודות למסעדה r1 (החברים שלו p2,p3 מעדיפים אותה), נקודה אחת ל-r2 ונקודה אחת ל-r3 (חבר p2 מעדיף את r2 וחבר p3 מעדיף את r3).
פרופסור p2 תורם נקודה אחת ל-r1,r2 (החבר p1 מעדיף את r1,r2).
פרופסור p3 תורם נקודה אחת ל-r1,r2 (החבר p1 מעדיף את r1,r2).
סך כל הניקוד: ארבע נק' ל-r1, שלוש נקודות ל-r2, נקודה אחת ל-r3.
לכן תוחזר רשימה המכילה את המזהה 1.



- `toString()` - מחזירה מחרוזת המתארת את המסעדה - מוסבר בהמשך.

טיפול בשגיאות

על מנת לפתור את התרגיל, תצטרכו להשתמש ב-Java Exceptions. נושא זה יילמד לעומק בהמשך הקורס, ולכן בתרגיל תתבקשו להשתמש בתכונות בסיסיות בלבד. בדומה ל ++C, על מנת להצהיר על קטע קוד בתכנית שעלול לזרוק חריגה יש לעטוף אותו בבלוק **try**, ועל מנת לטפל בשגיאה יש ליצור בלוק **catch**. בניגוד ל ++C, ניתן "לזרוק" אך ורק אובייקטים מטיפוס היורש (ישירות או בעקיפין) מהטיפוס Throwable. ישנם כמה סוגים של

טיפוסי חריגות, כאשר בתרגיל זה עליכם להשתמש אך ורק ב-checked exceptions. בחריגות מסוג זה, מתודה המכילה קטע קוד אשר עלול לייצר חריגה, חייבת להצהיר על כך. ההצהרה היא חלק בלתי נפרד מהחתימה של המתודה.
לדוגמה:

```
class AwesomeException extends Exception { ... }

class A {
      public void f() throws AwesomeException { ... }
}
```

המתודה f, מצהירה על כך שהיא עלולה לייצר חריגת AwesomeException בעת ריצתה.

על מנת לזרוק חריגה מהטיפוס AwesomeException יש להשתמש במילה השמורה throw :

```
if (x < 0)
      throw new AwesomeException();
```

הנחיות ורמזים למימוש

- המתודה toString(): כנהוג בJava , על המתודה להחזיר תיאור של האובייקט **בהתאם לפורמט המוגדר מראש בממשק המסופק**. שימו לב, על מנת לממש את מתודות אלו בקלות, עליכם לחשוב כיצד לבנות את המחלקות ואילו שדות יהיו לאובייקטים מטעמן. (הערה: לא צריך להוסיף שורה חדשה בסוף המחרוזת)
- **מומלץ** להשתמש בתכונות החדשות של Java 8 בפרט – , streams $\lambda - expr$.
- ניתן להוסיף מחלקות עזר ו/או מחלקות אבסטרקטיות (במידת הצורך).
- במתודות המחזירות אוסף (כמו registeredProfesores, favorites, getFriends) יש לבצע shallow copy.

הנחות במהלך התרגיל

- ניתן להניח שבעת איטרציה על אוספים לא יתווספו איברים חדשים לאוסף. כמו כן, ניתן להניח שלא ישתנה האוסף שעליו עוברים. למשל, בעת המעבר על אוסף המסעדות שמתקבל מקריאה ל-favoritesByRate לא יתווספו דירוגים למסעדות אחרות.
- ניתן להניח שבזמן בדיקת המערכת, יתווספו קשרי חברות אך ורק באמצעות קריאה ל-addConnection **ולא ישירות** דרך הפעלת addFriend על מופע של Profesor.
- בכל המתודות המקבלות אובייקט, ניתן להניח שלא יישלח null.

- ניתן להניח שבזמן בדיקת המערכת לא ניצור מחלקות נוספות (פרט ל-3 שתוארו) שיממשו את הממשקים המצורפים.
- ניתן להניח שאם יש פרופסור עם `id` הרשום למערכת, לא תיקרא המתודה `addConnection` עם פרופסור בעל אותו `id`, אך שאינו רשום למערכת.

בדיקות אוטומטיות ע"י JUnit

עם התרגיל סופקה לכם מחלקת בדיקות הנקראת `Example`, המשתמשת בספרייה JUnit. אנו ממליצים להפעיל את הבדיקות האלו על הפתרון שלכם, ולכתוב בדיקות נוספות באמצעות ספרייה זו כדי להקל על מלאכת הבדיקה.
<u>אין חובה לבדוק את התרגיל כלל, וכך או כך אין להגיש בדיקות.</u>

לנוחיותכם, המצגת מסדנת ה git + IDE, נמצאת באתר הקורס, ובה הסבר מפורט על התחלה עם JUnit ב IntelliJ.

דרישות והערות כלליות

1. אין לשנות את הקבצים המצורפים (של `package OOP.Provided`). הבודק האוטומטי דורס את הקבצים עם הגרסה המצורפת.
2. עליכם לוודא שהמתודה `equals` עונה על החוזה שלה כפי שנלמד בתרגול עבור המחלקות בהן נדרשתם להגדיר אותה.
3. יש לתעד את כל המחלקות ואת כל המתודות בפתרון באופן סביר – יש לתעד כל דבר שאינו מובן מאליו.
4. כל המחלקות שלכם צריכות להיות ממומשות ב-`package OOP.Solution`.
5. אין להשתמש בספריות חיצוניות לצורך הפתרון. ניתן להשתמש במחלקות מתוך `java.util`.
6. אין להדפיס לערוץ פלט הסטנדרטי או לערוץ השגיאות הסטנדרטי. אם אתם משתמשים בפלט לצורך בדיקות, הקפידו להסיר את ההדפסות לפני ההגשה.
7. הקפידו להסיר שורות ייבוא לקבצים או ספריות שאינם חלק מהקוד שניתן לכם או שהנכם משתמשים בו (למשל ייבוא לקבצי בדיקות).
8. לתרגיל מצורף קובץ בשם `Example.java` שמכיל דוגמה להרצה של המערכת. חובה לוודא שה-test שבקובץ מתקמפל ועובר עם ההגשה (אם הוא לא, אז בסיכוי גבוה ה-test-ים הרשמיים לא יעברו). אין לצרף את `Example.java` להגשה.

הוראות הגשה

- בקשות לדחייה יש לשלוח למתרגל האחראי על הקורס (נתן) בלבד. מכיוון שבקורס מדיניות איחורים - ראו מידע באתר - דחיות יאושרו רק מסיבות לא צפויות או לא נשלטות (כמו מילואים).
- יש להגיש קובץ בשם $OOP2\_ID1\_ID2.zip$ המכיל:
  - קובץ בשם readme.txt בפורמט הבא:

Name1 id1 email1
Name2 id2 email2

  - על ה-zip להכיל את כל קבצי הקוד שכתבתם לצורך התרגיל (ללא תיקיות, אלא את קבצי הקוד ישירות תחת ה-zip).

- **הימנעו** משימוש בתיקיות בתוך ה-zip ומהגשת קבצים שבחבילות אחרות.
- אין להגיש את הקבצים המצורפים לתרגיל (מוץ מאלה של `package OOP.Solution` כמובן) ואין להגיש test-ים.
- הגשה שלא לפי ההוראות תגרור הורדת ציון בהתאם.

בהצלחה !!!