תיכון תוכנה - תרגיל מס' 1

סמסטר חורף תשפ"ה

תאריך פרסום: 26/11/2024

תאריך הגשה: 17/12/2024 (עד 23:55)

1 הערות כלליות

- ניתנת לכם האפשרות להגיש את המטלה באיחור של עד 3 ימים, כאשר על כל יום איחור יורדו 5 נק' מהציון המקסימלי. כלומר הציון המקסימלי עבור מטלה שהוגשה באיחור של עד 24 שעות הוא 95, עבור מטלה שהוגשה באיחור של בין 24 ל-48 שעות הציון המקסימלי הוא 90 ועבור איחור של בין 48 ל-72 שעות הציון המקסימלי הוא 85.
- בכל שאלה תוכלו לפנות למתרגל במייל.
- **נא לציין את מס' הקורס (2360700) בשורת הנושא במייל.**
- מומלץ לקרוא את כל המסמכים/קבצים הרלוונטיים לתרגיל לפני תחילת הפתרון.
- הגשה אלקטרונית בלבד דרך אתר הקורס.
- הגשה בזוגות בלבד.
- סביבת העבודה הנתמכת ע"י סגל הקורס היא IntelliJ IDEA.
- שינויים/עדכונים בנוגע לתרגיל ימצאו במערכת ה-Piazza של הקורס. עליכם לעקוב אחר החדשות והעדכונים שיפורסמו שם.
- במידה ויש לכם שאלה כלשהי לגבי התרגיל, בדקו ראשית אם היא נענתה במערכת ה-Piazza לפני פנייתכם למתרגל. הדבר יחסוך לכם זמן רב בהמתנה לתשובה. בנוסף, הדבר יקל על המתרגל לתת תשובות עקביות לסטודנטים שונים.
- מטרת תרגיל זה היא "חימום" המערכת והיכרות עם חלק מה-Frameworks אשר נעבוד איתן הסמסטר: Gradle וספריות unit testing שונות.
- בחלק 2 תוגדר הארכיטקטורה אשר בה נשתמש בכל תרגילי הבית בסמסטר.
- בחלק 3 מתוארת האפליקציה שעליכם לממש בתרגיל זה.

2 ארכיטקטורה

בכל התרגילים הסמסטר יהיה עליכם לממש **אפליקציה** כלשהי מעל מסד נתונים. להרצת האפליקציה ישנם שני שלבים:

1. שלב ה-setup. בשלב זה האפליקציה שלכם תטען את הנתונים אשר ינתנו בפורמט כלשהו ממקור חיצוני, ותבנה את מסד הנתונים שבו תשתמש בשלב השני. עליכם לשמור את המידע בצורה בלתי-נדיפה (persistent) ע"י שימוש בספרייה חיצונית אשר תסופק לכם.
2. שלב השאילתות. מענה על שאילתות על המידע ועדכון המידע במקרה הצורך. בשלב זה, תהיה הגבלה של זמן. על כן, כדאי לשמור את המידע בצורה חכמה בשלב 1, על מנת לאפשר גישה מהירה אליו. אנו נספק לכם **מערכת שמירת נתונים פרימיטיבית** שמהווה שכבת אבסטרקציה מעל עבודה עם קבצים. עליכם לעבוד עם מערכת זו בלבד, ולא לעבוד עם מערכת הקבצים או מנגנון אחר באופן ישיר! המערכת חושפת מספר מצומצם מאוד של פעולות בסיסיות (בתרגיל זה: כתיבה וקריאה של מחרוזות; ה-API הספציפי ישתנה בין תרגיל לתרגיל). **שימו לב**, עליכם לממש **ספריה** יותר עשירה **מעל** המערכת המסופקת. ספרייה זו תשמש אתכם בתרגיל הנוכחי וכן בתרגילים עתידיים ואת האפליקציה לכתוב **מעל** ספרייה זו. אידיאלית, האפליקציה כלל לא תהיה מודעת לשכבה שאנו נספק לכם.

הטסטים של הצוות יכתבו **מעל** שכבת האפליקציה, ולא יהיה תלויים כלל בספרייה.

2.1. מבנה הפרויקט ב-Gradle

מורכב ממודול האב, **שלושה**[1] מודולים פונקציונליים ומודול המגדיר את לוגיקת הבנייה של הפרויקט כולו. כלומר ישנם חמישה מודולים בסך הכל, המנוהלים כולם ע"י Gradle:

1. **מודול אב** (base[2]): כל המודולים יורשים מפרויקט זה.
2. **מודול ספריית פעולות מעל מערכת קבצים** (library): ספרייה זו (אותה תממשו בתרגיל בית זה) תספק פונקציונליות כללית של מסד נתונים לאפליקציות שתיבנו במהלך הסמסטר, והיא תשמש אתכם גם לתרגילים עתידיים.
3. **מודול אפליקציה** (X[3]-app): כוללת את הלוגיקה העסקית של התרגיל. הדרישות בתרגיל ינתנו ברמת האפליקציה.
4. **מודול טסטים** (X-test): מטרתו היא בדיקת הקוד שלכם ע"י צוות הקורס. מצורפים בו מספר טסטים **בסיסיים**.
5. **מודול לוגיקת הבנייה** (buildSrc): מכיל הגדרות plugins ואת התלויות השונות (פנימיות וחיצוניות) בפרויקט כולו.

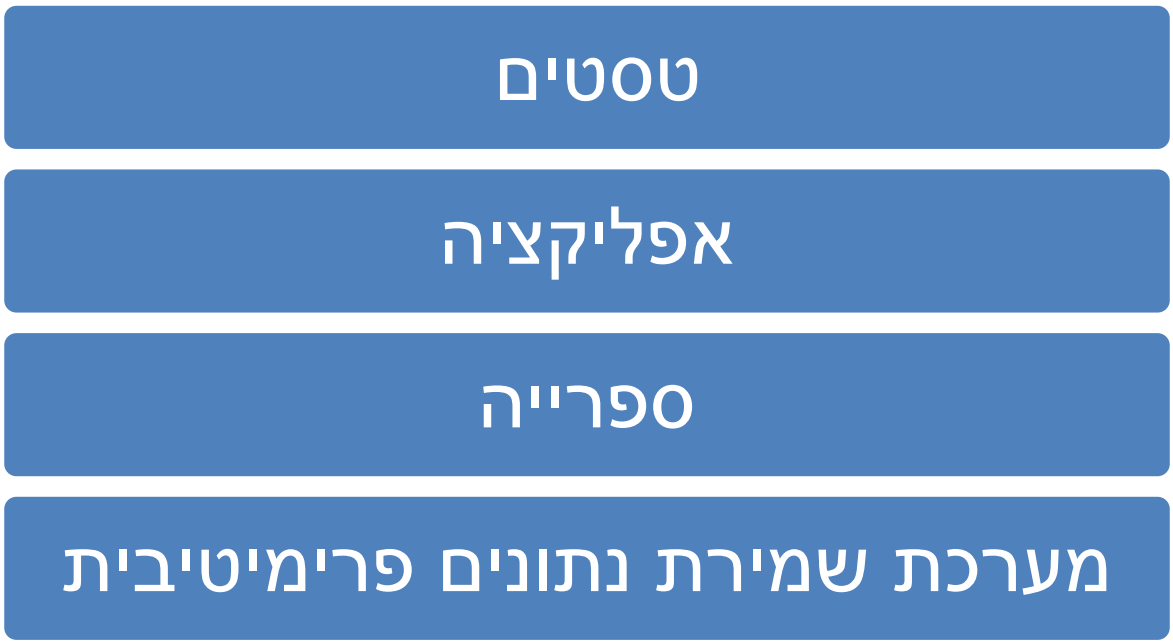
---

[1] **אל תיבהלו!** ההפרדה למספר מודולים היא על מנת לעזור לכם לבצע חלוקה נכונה ונקייה של התוכנית לחלקים נפרדים. היתרון המרכזי של שימוש במודולים על פני שימוש בחבילות הוא שקל הרבה יותר לאכוף תלויות בין המודולים השונים.

[2] זהו השם של מודול העל בפרויקט—מה שנקרא Gradle Multiproject—שמכיל את שאר המודולים. אין ספרייה תחת השם הזה.

[3] באשר X הוא שם האפליקציה; בתרגיל זה X=grades

הארכיטקטורה המומלצת בכל תרגיל תראה כך:



איור 1: מבנה התוכנית הכללי בכל אחד מהתרגילים

2.2. תלות בין פרויקטים

על מנת לעודד הפרדה נכונה למודולים, נעשה שימוש ב-Gradle כדי לאכוף תלויות בין הפרויקטים. בפרט, מודול הטסטים תלוי במודול האפליקציה שתלוי במודול הספרייה שתלוי בספרייה החיצונית. הספרייה החיצונית אינה מופיעה כמודול נפרד, אלא מיובאת על-ידי Gradle.

כאמור, פרויקט הטסטים מכיר רק את פרויקט האפליקציה. באופן כללי, המשימות והטסטים יוגדרו כדרישות **על האפליקציה**. כדי לאפשר בדיקה של הקוד שלכם, יהיה עליכם לממש מחלקה/מספר מחלקות בפרויקט האפליקציה. בתרגילים **עתידיים** אתם תשתמשו ב-Guice על מנת לקנפג את האפליקציה.

2.3. קוד חיצוני

על מנת לממש את הפונקציונאליות הנדרשת בכל תרגיל, יהיה צורך להשתמש בספרייה חיצונית אשר תסופק ע"י צוות הקורס. הספרייה תספק שכבת אבסטרקציה **פרימיטיבית ודלה** לשמירת מידע בקבצים (או מימוש כלשהו אחר). לצורך עבודה על התרגיל, תסופק לכם גרסה **מנוונת** של הקוד. **במהלך הבדיקות של התוכנית שלכם לאחר ההגשה, האפליקציה תיבדק ע"י קוד אחר.** התפקיד של פרויקט הספרייה הוא להרחיב את הפונקציונליות של הקוד החיצוני על מנת לאפשר בניית אפליקציות מעליו. שימו לב כי ייתכן שה-API החיצוני ישתנה בין התרגילים.

הקוד מיובא ע"י Gradle, והתלות של פרויקט הספרייה בו כבר הוגדרה. כל שאר המודולים מכירים את הספריות Junit, Hamkrest, kotest.

2.4. מבנה התיקיות ב-Gradle

כל אחד מהמודולים הפונקציונליים של Gradle מכיל ארבע תיקיות קוד:

`src/main/kotlin` - עבור קוד המקור של הפרויקט

`src/main/resources` - עבור קבצי קונפיגורציה, פלט, קלט או כל משאב חיצוני שאינו קוד שבשימוש קוד המקור.

`src/test/kotlin` – עבור קוד הבדיקות.

`src/test/resources` – עבור קבצי קונפיגורציה, פלט, קלט או כל משאב חיצוני שאינו קוד שבשימוש הבדיקות.

כאשר אתם מייבאים את הפרויקט לסביבת העבודה שלכם, וודאו כי אתם מייבאים אותו כפרוייקט Gradle!

3 אפליקציית Grades

בתרגיל בית זה עליכם לממש "אפליקציה" בסיסית אשר שומרת שמות של סטודנטים, ומאפשרת למשוך אותם **ביעילות** לאחר מכן. זמן החיים של האפליקציה יורכב משני חלקים:

1. שלב קריאת הנתונים: בשלב זה האפליקציה תקבל את הקלט כמחרוזת.
2. שלב השירות: בשלב זה, האפליקציה תקבל כקלט תעודות זהות של סטודנט, ויהיה עליה להחזיר את הציונים שלו.

עליכם להשתמש בספרייה החיצונית אשר תוגדר למטה (**ורק** בספרייה החיצונית! אין לגשת למערכת הקבצים, או לשמור מידע בין השלבים בצורה אחרת).
עליכם לממש את המחלקות `GradesInitializer` ו-`GradesReader`, אשר תאפשר לטסטים של הצוות לבדוק את הקוד שלכם. קראו את הקוד עבור החתימות המדויקות.

3.1. *קלט לאפליקציה:*

הקלט לאפליקציה יהיה String בפורמט CSV (כלומר שדות אשר מופרדים ע"י פסיקים). כל שורה תכיל את מספר הזהות של הסטודנט, והציון שלו בקורס. אתם יכולים להניח שתעודת הזהות של הסטודנט תהיה 9 ספרות לכל היותר, והציון הוא מספר טבעי בן 3 ספרות לכל היותר. סטודנט יכול להופיע מספר פעמים; במקרה זה, הציון **האחרון** יחשב.

3.2. *ספרייה חיצונית*

ה-API של הקוד החיצוני מסופק ע"י המחלקה הבאה:

```
class LineStorage {
  companion object {
    fun appendLine(s: String)
    fun read(lineNumber: Int): String
    fun numberOfLines(): Int
  }
}
```

`appendLine()` מוסיפה שורה נוספת **בסוף** הקובץ. `read()` מחזירה את התוכן של שורה מסוימת (השורה הראשונה היא מספר 0, כמובן). `numberOfLines()` מחזירה את מספר השורות הכולל בקובץ. השימוש ב-

`companion object` מאפשר גישה למתודות המחלקה ע"י פירוט שם המחלקה בתור ה-receiver. למשל קריאה ל-read תתבצע ע"י `LineStorage.read(...)`.

3.3. *הגבלות זמנים*

פעולת ה-`append` מסיימת מיד. פעולת ה-`read` לוקחת זמן **לינארי** באורך השורה המוחזר: 1ms עבור כל תו. למשל, אם התוכן המוחזר הוא "foobar123", הפונקציה תחזור לאחר 9 מילישניות. שימו לב שהזמן לא תלוי **במספר** השורה. פונקציית ה-`numberOfLines` חוזרת לאחר 100 מילישניות. זמנים אלה מתייחסים להתנהגות המובנית של `LineStorage` ועליכם להתייחס אליהם כנתונים. יחד עם זאת, המימוש שלכם נדרש לתמוך במספר מקסימאלי של עד **מיליון** סטודנטים, ועל כל טסט לקחת לכל היותר **10 שניות**. כל טסט יכלול שליפה אחת לכל היותר.

3.4. *מימוש מנוון ובדיקות יחידה*

לצורך פיתוח, המימוש של `LineStorage` יהיה מימוש מנוון: פונקציית ה-`appendLine` לא תעשה דבר, פונקציית ה-`read` תחזיר מחרוזת ריקה, ו-`numberOfLines` תחזיר 0. על מנת לבדוק את הקוד שלכם בראוי אתם נדרשים לספק מימוש למחלקת דמה, גם היא בשם `LineStorage`, הנמצאת בחבילה הבאה:
`base/library/src/test/5echni/il.ac.technion.cs.sd.grades.external`
המחלקה מיובאת כבר למחלקה `StorageLibrary` בה תממשו את הספרייה. אתם חופשיים לשנות את השם של מחלקת הספרייה אך שימו לב כי עליכם לשנות את שם המחלקה המיובאת ל-`GradesInitializer` ול-`GradesReader`.

בנוסף, על מנת לוודא שהמימושים שלכם יעמדו בהגבלות הזמנים של הטסטים, עליכם לוודא שהמימוש של `LineStorage` יקיים את אילוצי הזמנים הנתונים ב-3.3.

לבסוף, עליכם לוודא כי אתם משיגים כיסוי (coverage) של **לפחות 80%** בכל היחידיות הלוגיות[4] הנבדקות.

3.5. *הערות ודגשים נוספים*

- בעבודה זו תיבדק בין היתר את איכות הבדיקות שלכם. עבור כל יחידה לוגית שתכתבו עליכם לכתוב סט של בדיקות באחת מספריות ה-unit testing שראיתם בהרצאות ובתרגולים המוודא את התנהגותה. עליכם לוודא כי הבדיקות שתכתבו תואמות את הסטנדרטים שנלמדו בכיתה.
- אם תחליטו לכתוב את הבדיקות שלכם עם kotest עליכם להתקין plugin ב-IntelliJ IDEA. לשם כך, לחצו על file -> settings -> plugins, לחצו על לשונית Marketplace, הכניסו לשורת החיפוש את המילה kotest והתקינו.
- באופן כללי, לא **חובה** להשתמש בדברים אשר נלמדו בכיתה. תחת זאת, כן **מומלץ** להשתמש בהם; שנים רבות של ניסיון הובילו למסקנה ששימוש נכון בעקרונות שנלמדו וילמדו מוביל לפיתוח מהיר יותר וגם *נכון* יותר (מבחינת תכן).
- ניתן לכם חופש רב בתכנון המערכת. לרוב שאלות התכן שתיתקלו בהן **אין** תשובה של נכון/לא נכון. לכן, שלב ההתדיינות וההתייעצות בין השותפים הוא *קריטי* להפנמת החומר הנלמד והפקת המקסימום מהקורס.
- אכיפת התלויות בין הפרויקטים מתבצעת ע"י Gradle plugins במודול buildSrc. למרות שניתן, מאוד מומלץ **לא** לשנות את התלויות בין הפרויקטים.
- כפי שצוין בחלק 2, עליכם לבנות את האפליקציה מעל ספריה כללית. בתרגילים הבאים, ואף בתרגיל זה, השקיעו מחשבה רבה בחלוקת המשימות בין האפליקציה לספריה. אם תכתבו יותר מדי קוד בצד

[4] יחידה לוגית מוגדרת להיות כל מחלקה המבצעת חישוב כלשהו. אין צורך לבדוק עצמים שרק מחזיקים מידע (POJO), כלומר שמכילים רק `getters/setters`.

האפליקציה, יהיה לכם קשה יותר בתרגילים עתידיים. אם תכתבו יותר מדי קוד בצד הספרייה, ייתכן שתבזבזו זמן בבניית ממשק "גנרי מדי", במקום לפתור את הבעיה ישירות.

- את כל הספריות החיצוניות יש לייבא באמצעות Gradle בלבד, ע"י שימוש במנגנון ה-dependencies שלו.

4 סיכום

עליכם לבצע את המשימות הבאות:

- מימוש המחלקות `GradesInitializer` ו-`GradesReader`.
- מימוש הספרייה.
- מימוש מחלקת הדמה `LineStorage` כך שתעמוד בהגבלות הזמנים המפורטות ב-3.3.
- כתיבת בדיקות יחידה עבור כל יחידה לוגית עם כיסוי של לפחות 80% לכל יחידה נבדקת.

5 תרגילים עתידיים

- משימה זו הינה מבוא לשתי משימות (איטרציות) בהן נעבוד ונשכלל את ספריית הנתונים שלנו.
- בכל תרגיל יהיה עליכם לבנות אפליקציה נפרדת מעל הספרייה. התיאור שבחלק 2 ישמר ברובו בין האפליקציות. תכן של הקוד לפי הפרמטרים שנלמדו בכיתה יעזור לכם במשימות עתידיות.

6 הוראות הגשה

את הפרויקט יש לייצא כקובץ zip. על מנת להקל על המשימה, הוגדר task ב-base/build.gradle.kts העושה בדיוק זאת. על מנת להשתמש בו יש לבצע את הפעולות הבאות:

- בקובץ `base/gradle.properties` הורידו את ההערה (מחקו את ה-#) משתי השורות האחרונות ומלאו שם את מספרי הזהות שלכם כך שהן יראו כך:

```
4  gpr.id1=123456789
5  gpr.id2=987654321
```

- פתחו את הטרמינל ע"י לחיצה על 
- הריצו את הפקודה הבאה:
  - עבור Windows: `.\gradlew.bat zip`
  - עבור Linux/Mac: `./gradlew zip`
- כעת נוצרה תיקייה חדשה בשם archives תחת התיקייה הראשית של הפרויקט base ובתוכה תמצאו את קובץ ה-zip אותו עליכם להגיש.
- יש באפשרותכם לספק ל-Gradle מספר tasks ב-command line של הטרמינל, כך שמשימה מסוימת תתבצע רק אם המשימה הקודמת (אם קיימת) התבצעה בהצלחה. למשל, על מנת להריץ את כל הטסטים ובמידה וכולם עברו ליצור את קובץ ה-zip הריצו את הפקודות הבאות:
  - עבור Windows: `.\gradlew.bat test zip`
  - עבור Linux/Mac: `./gradlew test zip`

בהצלחה !!