קורס פיתוח מתקדם ב- JAVA

פרויקט בנית שרת\לקוח

ויחידת אלגוריתמיקה

HIT – מכון טכנולוגי חולון

מרצה: ניסים ברמי

סמסטר א 2025

תוכן עניינים

הסבר כללי – בנית שרת\לקוח ומודול אלגוריתמי

פרויקט זה יכלול 4 אבני דרך שכל אחת מהן תהווה מטלה בפני עצמה. לכל מטלה יהיה מועד הגשה ואתם **מחויבים** להגיש את מטלה 2 במועד שיקבע על מנת לקבל פידבק, מלבד המטלה הסופית (תרגיל 4).

שימו לב על מנת לסיים את הפרויקט שהוא 80% מהציון הסופי ולהגיע לתוצאה הסופית (מערכת עובדת והצגתה) תצטרכו לבצע את כל המטלות (אבני הדרך) אך כאמור לא להגיש את כולן.

ההמלצה שלי היא את המטלה הראשונה כדי להתנסות קצת בשפה תעשו לבד ואת שאר המטלות כולל הגנת הפרויקט בזוגות (לא ניתן להגיש בשלשות וכן ניתן להגיש בבודדים).

כל קטעי הקוד שלכם (מחלקות, ממשקים וכו') צריכים להיות כתובים ומעוצבים ע"פ כל עקרונות התכנות שנלמדו בקורס תכנות מונחה עצמים והקורס הנוכחי (יעילה, נקיה ומתועדת היטב). בנוסף בחלק מהתרגילים תצטרכו ע"פ דרישה להוסיף קבצי בדיקה (Unit test), שהם בפני עצמם נדבך מאוד חשוב בעולם התוכנה, שנועד לבדוק את הקוד שלכם לפני שהוא עובר לבדיקה חיצונית.

הערה חשובה: במהלך הפרויקט אשתדל לחשוף אתכם לכמה שיותר עקרונות תכנות, כלים ושיטות עבודה, הסברים לכך יינתנו כמובן במהלך השיעורים וכחלק מהמטלות ובנוסף ינתנו references לחומרי לימוד המרחיבים את אותם נושאים, הרחבה זו היא **חובה** וחלק בלתי נפרד מהקורס, עליכם ללמוד זאת ליישם בפרויקט ולהיות **מוכנים** להיבחן עליהם.

בהצלחה לכולכם

תיאור הפרויקט

בפרויקט זה עליכם לכתוב אפליקציה בארכיטקטורת שרת\לקוח (Client/Server) וכמו כן באפליקציה זו עליכם להכיל מודול אלגוריתמי שיסייע לה וייעל אותה בעבודתה.

את נושא האפליקציה **עליכם לבחור בעצמכם**, השתדלו לא לבחור נושא ולהגדיר דרישות שאינכם יכולים לעמוד בהם.

המטרה היא להשתמש בעקרונות התכנות והנושאים שילמדו בקורס ולא מעבר לכך.

החלקים של הפרויקט יכתבו בשפה הנלמדת בקורס בלבד (Java) ולא נשתמש

בטכנולוגיות שרת\לקוח נוספות לכתיבת האפליקציה כגון: Spring, react, angular וכו'. החלק האלגוריתמי יכלול "משפחה" של אלגוריתמים, ע"פ הדוגמה שתינתן בהמשך ועליכם **לממש לפחות שניים מהם**.

בנוסף תצטרכו לספק תסריטי בדיקה ולעשות שימוש בבדיקות יחידה לקוד על מנת להראות יציבות וכיסוי טוב לקוד שלכם.

בסיום הפרויקט, תגישו גם תוצרים נוספים כגון מצגת קצרה לתיאור ארכיטקטורת הפרויקט וסרטון בו תציגו את המערכת בפעולתה.

חלק א' – מודול האלגוריתמים

בחלק זה, עליכם ליישם מודול אלגוריתמי בדומה לניהול ה – Cache ולהשתמש בו בהמשך הפרויקט.

מצאו משפחה של אלגוריתמים שמבצעים אופטימיזציה כלשהי ויישמו אותם (שימו לב ! **לא ניתן לבחור את האלגוריתמים לניהול Cache כיוון שהם הדוגמה במסמך זה**)

להלן דוגמה לאלגוריתמים לניהול זכרון:

ראשית יוסבר הצורך בניהול זיכרון, ההסבר כיצד מנהלים אותו ולבסוף הדוגמה ל"**משפחה**" של האלגוריתמים שעושים זאת.

ההסבר:

Paging (דפדוף) היא שיטה לניהול זיכרון המאפשרת למערכת הפעלה להעביר קטעי זיכרון בין הזיכרון הראשי לזיכרון המשני .העברת הנתונים מתבצעת במקטעי זיכרון בעלי גודל זהה המכונים **דפים**. הדפדוף מהווה נדבך חשוב במימוש זיכרון וירטואלי בכל מערכות ההפעלה המודרניות, ומאפשר להן להשתמש בדיסק הקשיח עבור אחסון נתונים גדולים מדי מבלי להישמר בזיכרון הראשי.

על מנת לבצע את תהליך הדפדוף עושה מערכת ההפעלה שימוש ביחידת ה – MMU שהינה חלק אינטגרלי ממנה ותפקידה הוא תרגום מרחב הכתובות הווירטואלי אותו "מכיר" התהליך למרחב הכתובות הפיזי (המייצג את הזיכרון הראשי והמשני).

במידה ובקר הזיכרון מגלה שהדף המבוקש אינו ממופה לזיכרון הראשי נוצר Page fault (ליקוי דף) ובקר הזיכרון מעלה פסיקה מתאימה כדי לגרום למערכת ההפעלה לטעון את הדף המבוקש מהזיכרון המשני. מערכת ההפעלה מבצעת את הפעולות הבאות:

- קביעת מיקום הנתונים בהתקני האחסון המשני.
- במידה והזיכרון הראשי מלא, בחירת דף להסרה מהזיכרון הראשי.
- טעינת הנתונים המבוקשים לזיכרון הראשי.
- עדכון טבלת הדפים עם הנתונים החדשים.
- סיום הטיפול בפסיקה.

הצורך בפניה לכתובת מסוימת בזיכרון נובע משני מקורות עיקריים:

- גישה להוראת התוכנית הבאה לביצוע.
- גישה לנתונים על ידי הוראה מהתוכנית.

כאשר יש לטעון דף מהזיכרון המשני אך כל הדפים הקיימים בזיכרון הפיזי תפוסים יש להחליף את אחד הדפים עם הדף המבוקש. מערכת הדפדוף משתמשת באלגוריתם החלפה כדי לקבוע מהו הדף שיוחלף. קיימים **מספר אלגוריתמים** המנסים לענות על בעיה זו לכל אלגוריתם שיטת ניהול Cache משלו, אלגוריתמים אלה נקראים Cache Algorithm .

כאמור, עליכם לממש מספר אלגוריתמים בדומה לאלגוריתמי ה – Cache שתפקידם יהיה לסייע לאפליקציה שלכם לבצע את תפקידה בצורה יעילה וטובה יותר.

ראשית צרו ממשק (interface) שמגדיר את ה – API הכללי של ה"משפחה" של האלגוריתמים שלכם ותנו לו את השם הכללי הבא - **IAlgoNameOfAlgoFamily** כפי שניתן בדוגמה – IAlgoCache.

```
public interface IAlgoNameOfAlgo{

    // Relevant function...

}
```

לאחר מכן צרו את **שתי מחלקות נוספות לפחות** שיממשו את ה – interface שיצרתם כל אחת ע"פ הלוגיקה שלה למימוש האלגוריתם, למשל בדוגמה של ה – Caching ישנן 3 סוגי מימושים LRU/Random/SecondChance **כפי שמוצג בסוף החלק.**

זכרו שמי שישתמש בהמשך באלגוריתמים שסיפקתם צריך להכיר את האלגוריתם שהוא מפעיל (להכיר את ה – API של האלגוריתם) על מנת להשתמש בו אך בהחלט לא חייב להכיר את המימוש שלו (Concrete Classes).

בנוסף אנו נרצה לאפשר גמישות מוחלטת בשינוי מימוש האלגוריתם ולהוסיף מימושים נוספים ל - **IAlgoNameOfAlgoFamily** ככל שנרצה בכל שלב מבלי לשנות את ה – API שחשפנו. ה – **Design Pattern** שיאפשר לנו לעשות זאת בצורה הטובה ביותר הוא ה – **Strategy Pattern** אותו נכיר ונלמד בכיתה.

המחלקה האחרונה אותה אתם צריכים לכתוב בחלק זה היא ה – **IAlgoNameOfTheAlgoFamilyTest** שכל תפקידה הוא לבדוק את שלושת האלגוריתמים שמימשתם ופעולתם. מחלקה זו תשתמש ב- Junit framework ותכיל 2 מטודות @test **לפחות** לכל אחד מהאלגוריתמים.

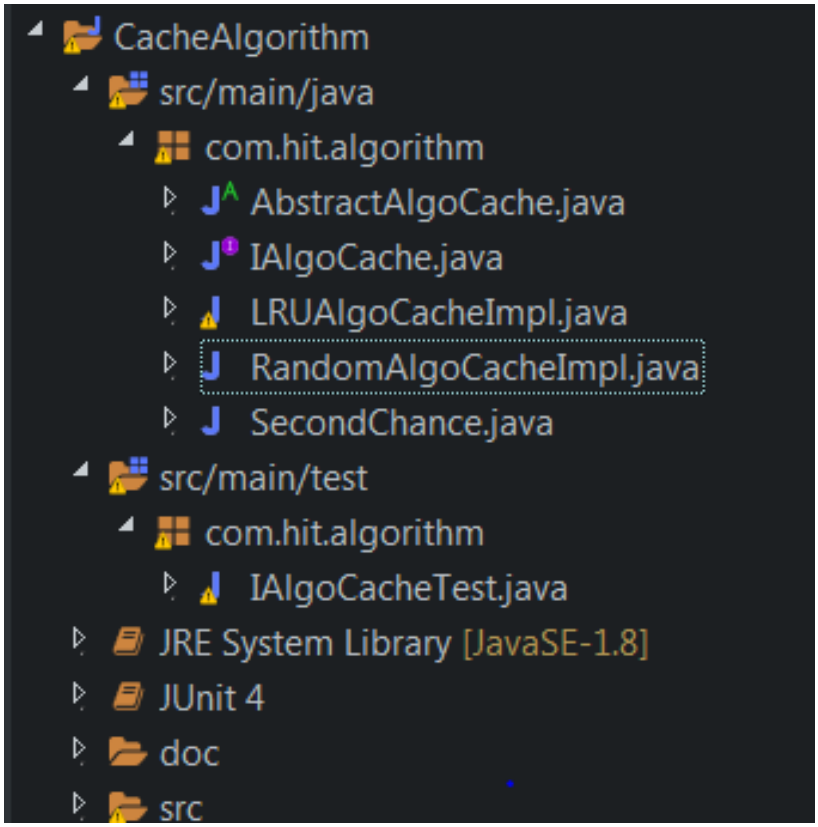
דמיינו שהמוצר הזה צריך להגיע ולרצות את הלקוח (במקרה זה הבודק)

לכן ככל שתבדקו יותר תגיעו לרמה טובה יותר ומוצר טוב יותר.

לבסוף ארזו **ע"פ הדוגמה** את כל המחלקות שלכם (צריכות להיות לפחות 3) ב – packages בדיוק

במבנה של הדוגמה (שימו לב היטב למבנה התיקיות)

ובשמות הבאים (שימו לב לשמות האלגוריתמים, החליפו לשמות של האלגוריתמים שלכם) :

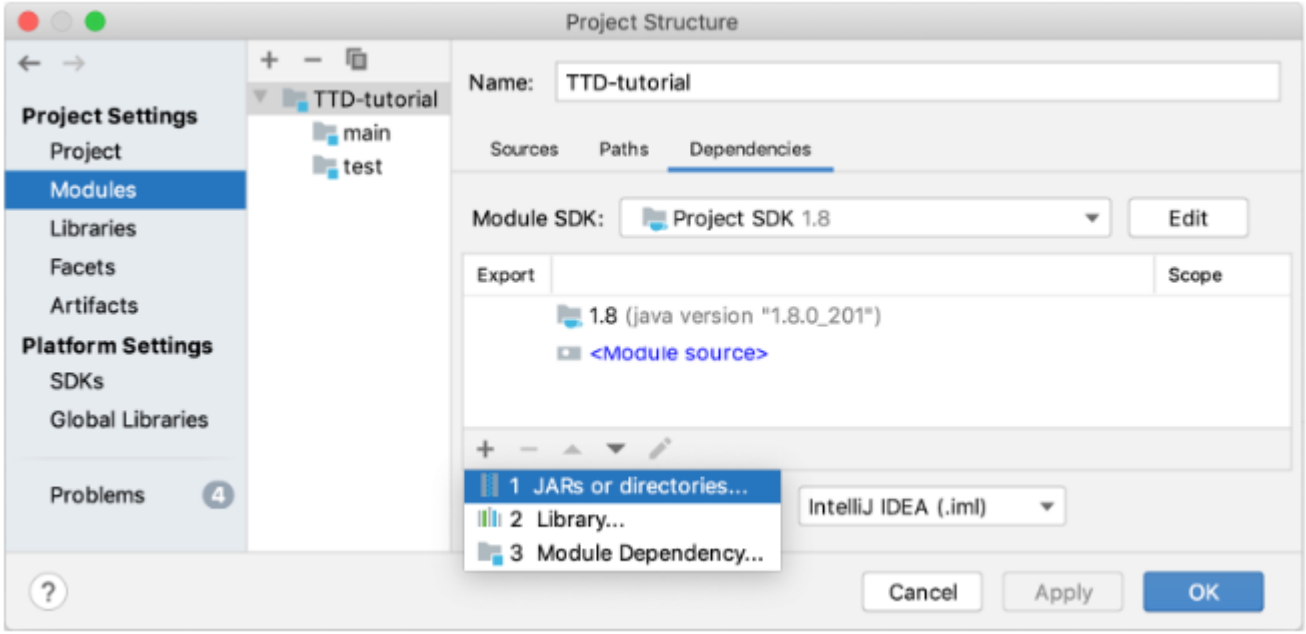


ארזו את הפרויקט שלכם **AlgorithmModule** הכולל את מחלקות האלגוריתמים שלכם וה – Test לתוכו וארזו אותו כ – jar נפרד ע"פ המדריך הבא:

1. Create jar file from intellij

צרו פרויקט חדש בשם – YourAppNameProject ולאחר מכן עשו include ל – **AlgorithmModule** .jar לפרויקט זה ע"פ המדריך הבא:

2. Adding Internal JARs (Method 1)



עכשיו הגיע הזמן להתחיל ולפתח את שאר האפליקציה שלנו

חלק חשוב ובלתי נפרד בפיתוח תוכנה הוא ה – design. לאחר שהדרישות מהלקוח הובנו ולפני שמתחילים לכתוב קוד, חייבים לעבור לשלב בו מתכננים את מבנה המערכת (**Architecture Design**).

בחלק זה של הפרויקט, נפתח את יחידות המערכת ונעמוד על הקשר ביניהן. בנוסף אנו נעצב את המערכת תוך שימוש בעקרונות בסיסיים ב- OOP **ושימוש בפתרונות סטנדרטיים** לבעיות מוכרות מעולם התוכנה שהם ה – design patterns.

העקרון הראשון והבסיסי אליו נצמד הוא היכולת לתת גמישות למערכת מבחינת הוספת יכולות other implementations בצורה פשוטה וקלה אך בד בבד גם לא לאפשר שינויים של ה – Application Programming Interface

עקרון חשוב זה נקרא:

Open/Close principal - open for extension, but closed for modification

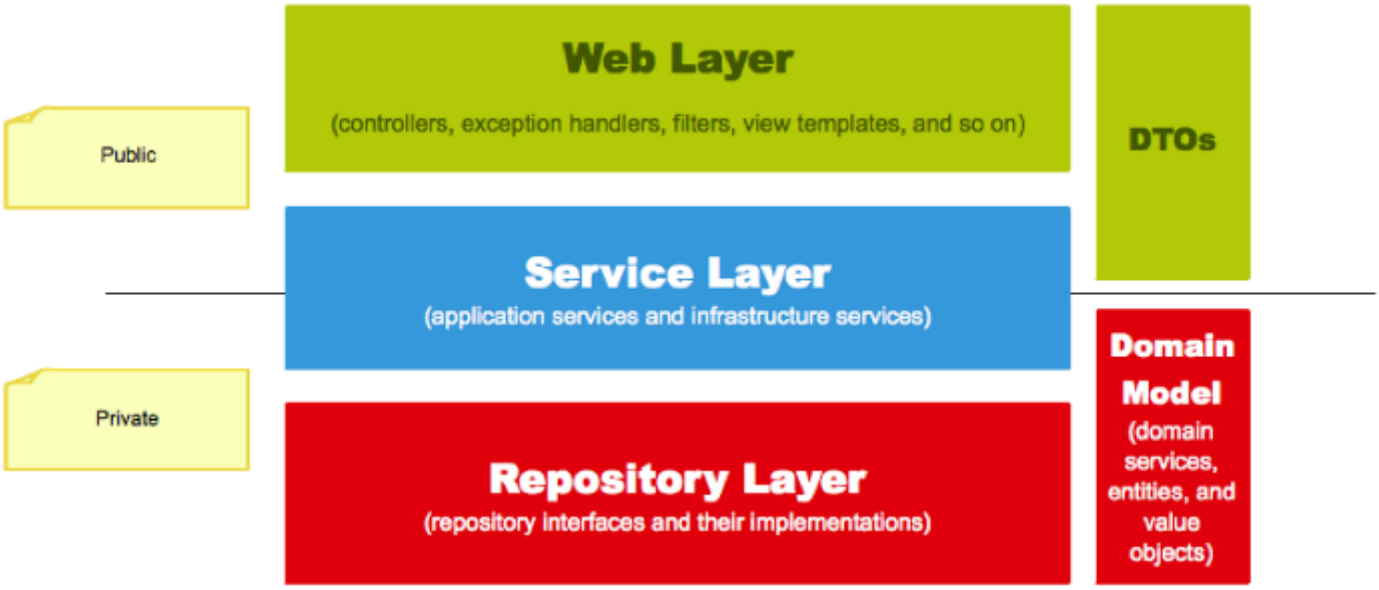
המחלקות הראשונות שנממש בחלק זה של הפרויקט הן ה – **YourServiceNameServices**.

שימו לב להחליף את שם ה – **YourServiceName**Service בשם הרלוונטי לפרויקט שלכם, מחלקות אלו אחראיות על ה – **Business Logic** של המערכת שלכם והן צריכות לדעת לגשת למחלקות שפונות ל – Repository לאיחזור \שמירה ומחיקה של המידע ולאחר מכן **לעבד** אותו ולדעת לאחזר את המידע למי שדורש אותו,

ולכן **YourServiceNameServices** צריכות להכיל (כ – members) את השיכבה (יהיו מספר מחלקות כאלה) שפונה ל – Repository (כפי שנאמר) שיכבה זו תיקרא IDao (ונתאר את הפעולה שלה בהמשך) ובנוסף את רכיב האלגוריתם IAlgoNameOfTheAlgoFamily במידת בצורך.

כל אחד מה - **YourServiceName**Service יתמוך בפעולות ע"פ צרכי האפליקציה שלכם אך בכל מקרה הוא יתמוך בפעולות הבאות שמירה, מחיקה והחזרה של ה – DataModels שלכם.

שימו לב לדיאגרמה הבאה (אנו נשלים בהמשך את שיכבת ה – Presentation/web):



השימוש של ה – **YourServiceName**Service ב- IAlgoNameOfTheAlgoFamily הוא שימוש ב – API בלבד (get, add ..etc), ללא הבנה כיצד מומשו אותן API וללא אפשרות לשנות אותו (closed for modification). אותן מחלקות ש"יוצקות" implementation לאותו API מועברות ל – **YourServiceName**Service מבחוץ ולכן ניתן בקלות להעביר סוגים שונים של מימושים, להוסיף בכל שלב אחרים ובאותה צורה להעביר גם אותם (open for extension). בכך השלמנו את ה – strategy pattern ושמרנו על עקרון ה – open/close principal.

צרו והוסיפו את מחלקות ה – DataModels שלכם, מחלקות אלו יהיו המחלקות שישמרו\יאוחזרו\יעודכנו\ימחקו במהלך פעולתה של האפליקציה שלכם, **לדוגמה** באפליקציה מסויימת יכולים להיות

ה – DataModel:

Customer, Product, Book etc…

שימו לב, מחלקות אילו אינן דורשות לוגיקה כלל הן פשוט צריכות להכיל את data שרלוונטי להן.

ממשק (interface) נוסף אותו תוסיפו לפרויקט הוא ה – IDao (מצורף בתיקיית ה – API), ממשק זה ישמש לקריאה וכתיבה של ה – DataModel מה – Data Source(Repository) אליו נתממשק, בפרויקט זה הוא יהיה פשוט **file מקומי** בשם DataSource.txt (**ראו בסוף** התרגיל היכן אתם אמורים למקם קובץ זה).

IDao.html

מחלקות נוספות שעליכם להוסיף הן ה – MyDMFileImpl, מחלקות אלו הן מימוש של ממשק ה – IDao והן כאמור, אחראיות לשמור את המידע ל – File שניתן לה ע"י המשתמש.

ה – MyDMFileImpl יקבל את ה – pathFile (דרך **ה – CTOR**) אליו תיגש המחלקה בכדי לקרוא ולכתוב.
על מנת לכתוב ולקרוא מ – file אנו זקוקים ל- input/output stream שמתאימים לקריאה\כתיבה של קבצים, בנוסף אנו גם זקוקים ל - streams שמתאימים לקריאת\כתיבת אובייקטים. כפי שלמדנו בכיתה, על מנת להקל ולייעל את תהליכי ה – streaming בשל שכיחותם וחשיבותם הוסיפו ב – java מחלקות מובנות שהותאמו לכל צורך השתמשו בהם...

גם בחלק זה המחלקה האחרונה אותה אתם צריכים לכתוב היא טסטר שיבדוק את יחידת ניהול הזיכרון שם הטסטר תהיה – **YourServiceName**Service**Test** ותפקידה הוא לבדוק את תקינות פעולת המערכת.

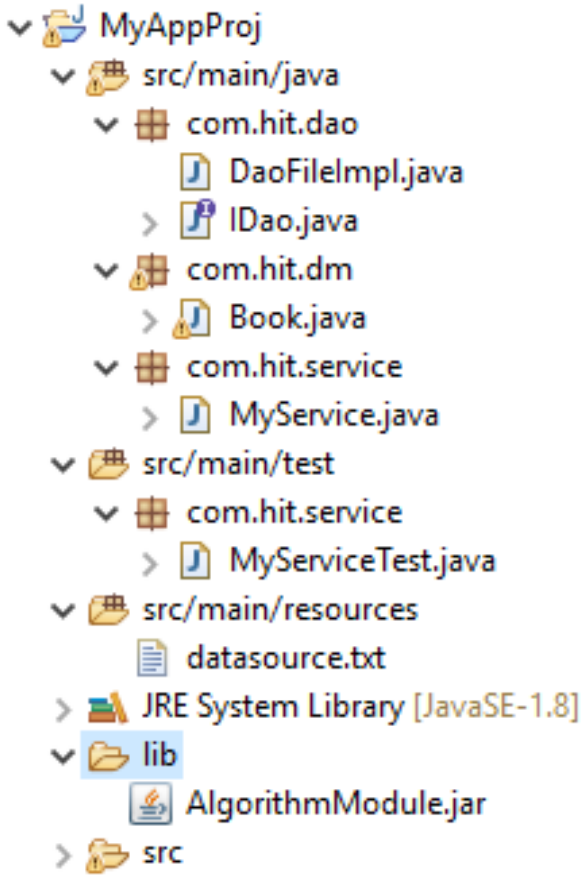
הטסטר צריך לבנות את כל האובייקטים הרלוונטיים (**YourServiceName**Service, IAlgoNameOfTheAlgoFamily, IDao), ויחבר אותם על מנת שיוכל להפעיל את המערכת.

נקודות חשובות בעת בדיקת המערכת:

- אין צורך בטסטר **YourServiceNameServiceTest** כדי לבדוק שוב את כל האלגוריתמים, מספיק להשתמש באחד (מהסיבה שכבר בדקתם אותם בתרגיל הקודם).
- תצרו file אליו יכתוב ה – Dao כחלק מהפרויקט שלכם על מנת להקל את השימוש בו.
- **אתם רשאים להוסיף מחלקות ע"פ הצורך שלכם**

לבסוף ארזו את כל המחלקות **החדשות** המתוארות ב – packages במבנה ובשמות הבאים (**שמות אלו הן רק דוגמה**)

ובנוסף שימו לב לקובץ ה – datasource.txt וה- include של ה –AlgorithmModule.jar:



חלק שלישי – תקשורת

ארכיטקטורת Client/Server היא אחת מתצורות ההתקשורת הנפוצות ביותר ברשתות מחשבים. תצורה זו פשוטה ומבוססת בעיקרה על רעיון שבו קיימים שני צדדים בארכיטקטורה ולכל צד תפקיד משלו:

צד שרת - תפקידו לספק **שירותים** כלשהם עליהם הוא מצהיר שהוא יודע לספק.

צד לקוח – תפקידו **להפעיל** את אותם שירותים **לקבל את המידע** ולעשות איתו את **שהוגדר לו**.

בחלק זה של הפרויקט אנו נבנה שרת שיעבוד ע"ג פרוטוקול TCP כפי שנלמד בכיתה ונשלב את יחידת האלגוריתם כחלק משרת זה:

בחלק זה תצטרכו לממש את החלק של ה – Server שגם יכיל את כל שאר החלקים שפיתחתם לפני.

המחלקה הראשונה שפשוט תקרא Server תכיל אוביקט מסוג **ServerSocket שיאזין ל – port למתקבל ב – constructor של המחלקה,** ותנהל את התקשורת עם הלקוחות.
מחלקה זו (בדומה למה שעשינו בכיתה) תכיל מטודה שחתימתה היא:

```
public void run(){
      //... Add implementation here
}
```

במטודה זו עליכם לאתחל את כל הרכיבים שרלוונטיים לשרת שלכם וכמו כן לדאוג להאזין לבקשות

עליכם לחשוף API בשרת שידע לענות לכל הלוגיקה של האפליקציה שלכם (כל מה שאתם מאפשרים לבצע באפליקציה שלכם), רצוי שהאפליקציה שלכם תדע לענות **לפחות** לבקשות שעובדות על ה – DataModels שלכם כגון שמירה\עדכון\מחיקה וכו'.

לדוגמא: במידה והאפליקציה שלכם היא חנות ספרים, רצוי שחלק מה – API ידע לאחזר\לשמור\לעדכן ספרים בחנות.

שימו לב !!

כל המידע בבקשות לשרת שלכם צריך להיות ב**מבנה json** כפי שלמדנו בכיתה, מה שיאפשר לכל צד לקוח (אותו גם אתם תממשו בהמשך) לפנות לשרת שלכם ולהפעיל את ה – API ללא קשר לשפת התכנות והטכנולוגיה בו הוא ממומש.

המחלקה השניה היא ה – HandleRequest. תפקידה של מחלקה זו הוא לקבל את הבקשה (Socket) מה –Server, לקרוא ממנה את המידע שיהיה תמיד אובייקט מסוג Request המגיע במבנה json בעזרת המחלקות הרלוונטיות הארוזות ב – gson.jar כפי שלמדנו בכיתה, לקרוא מה – Header של ה- Request את השדה action ולהעביר אותה למטודה הרלוונטית ב –Controller הרלוונטי (שיתואר כעת).
מציאת ה – Controller הרלוונטי יעשה באמצעות Factory Pattern שיטען כאשר האפליקציה עולה.

המחלקה הבאות אותן תצטרכו לייצר הן ה –Controllers מטרתן של מחלקות אלו היא ליצור שכבת הפרדה בין ה - Services שלכם לשכבת ה – Networking, וכמו כן לחשוף את ה – API הרלוונטי של האפליקציה שלכם ע"פ הצורך. לדוגמה:

```
public void saveBook(Book book)

public void deleteBook(Book book)

public Book getBook(Long id)
```

ובנוסף כל מה שנדרש כדי לקיים את האפליקציה !

לבסוף צרו את המחלקות הבאות–Response/Request מחלקות אלו כפי שתואר לעיל יועברו מהלקוח דרך כל רכיבי השרת עד שמירת\איחזור\מחיקת ה – Data (Request) וחזרה (Response).

<u>להלן דוגמאות למבנה ה – json של ה – Request בחנות ספרים</u>

```
{"headers":
            {"action":"book/update"},
   "body":
            {"name":"Hobbit",
             "content" : "Some String Data"}
}
```

```
{ "headers":
      {"action":"book/get"},
   "body":
            {"name":"Hobbit" }
```

```
{   "headers":
              {"action":"book/delete"},
   "body":
            {"name":"Hobbit" }
```

המחלקה האחרונה בצד ה – server תיהיה ה –Driver וזה הקוד היחיד שאמור להיות מוכל בה:

```
public class ServerDriver {

      public static void main(String[] args) {

            Server server = new Server(34567);

            new Thread(server).start();

      }
}
```

מחלקה זו היא למעשה החלק שמחבר לנו את כל **רכיבי המערכת וגם מפעיל אותם**.

מספר דגשים חשובים לחלק זה:

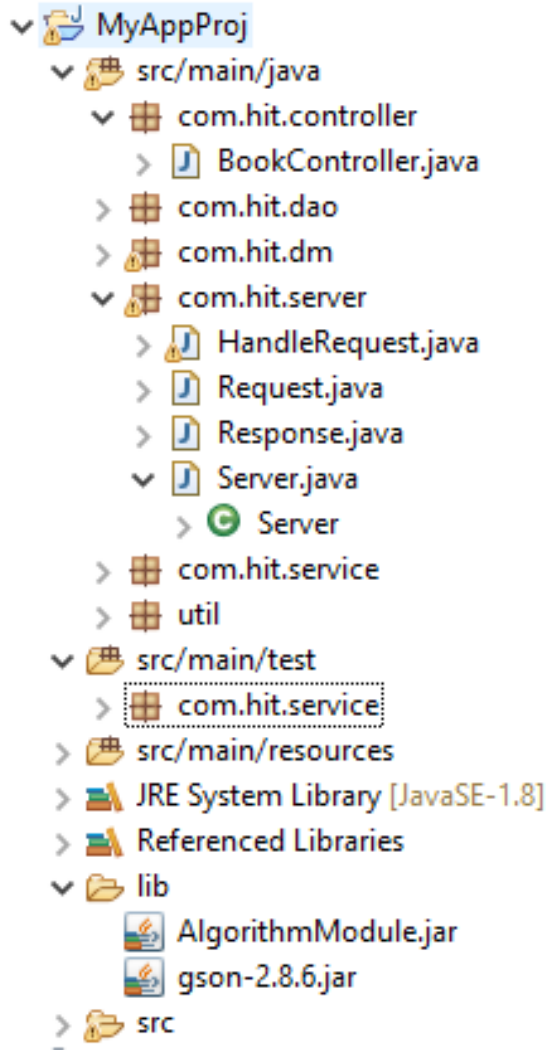
- השתמשו ב – Decorator מתאימים על מנת לעטוף את input/output streams של הבקשה.
  אני השתמשתי ב – Decorator אלו:

```
Scanner reader = new Scanner(new InputStreamReader(socket.getInputStream()))
PrintWriter writer = new PrintWriter(new OutputStreamWriter(socket.getOutputStream())
```

- כאשר מגיעה בקשה (Request) מה – client כ – json file, הבקשה מכילה content שקשור לאפליקציה שלכם (שוב בדוגמא זו, רשימת ספר):

```
Type ref = new TypeToken<Request<Book>[]>>(){}.getType();
Request<DataModel<T>[]> request = new Gson().fromJson(req, ref);
```

שימו לב למבנה התיקיות:



חלק ד' – ממשק משתמש

רקע (ויקיפדיה) - את המונח "חוויית המשתמש" טבע לראשונה דונלד נורמן(Donald Norman) באמצע שנות ה-90. בעבר התייחסו בעיקר לממשק משתמש UI, הממשק בין הטכנולוגיה לאדם לצורך השגת מטרה כלשהי, אך בשנים אלו חילחלה ההבנה כי ממשק המשתמש מהווה רק אספקט מסוים מתוך עולם חוויית המשתמש המתייחס גם לעיצוב, ארכיטקטורת המידע, אסטרטגיה, שיווק וטכנולוגיה וגם להיבטים רגשיים, אנושיים ותחושתיים. חוויית משתמש טובה יכולה לחסוך מאמצי הטמעה והשקעה בפעילות ניהול ידע לעידוד השימוש וניהול השינוי. בשנים האחרונות תחום חוויית המשתמש הפך למשמעותי בכל פתרון מחשובי בכלל, ופתרונות ניהול ידע בפרט, ונתפס כמרכיב קריטי והכרחי (Critical Success Factor) .

אחד האנשים שתרם רבות להכרת והבנת המושג "חוויית משתמש" ומשמעותו הוא סטיב ג'ובס, אשר התבסס על ממשק משתמש גרפי בעת פיתוח המחשב האישי" מקינטוש."

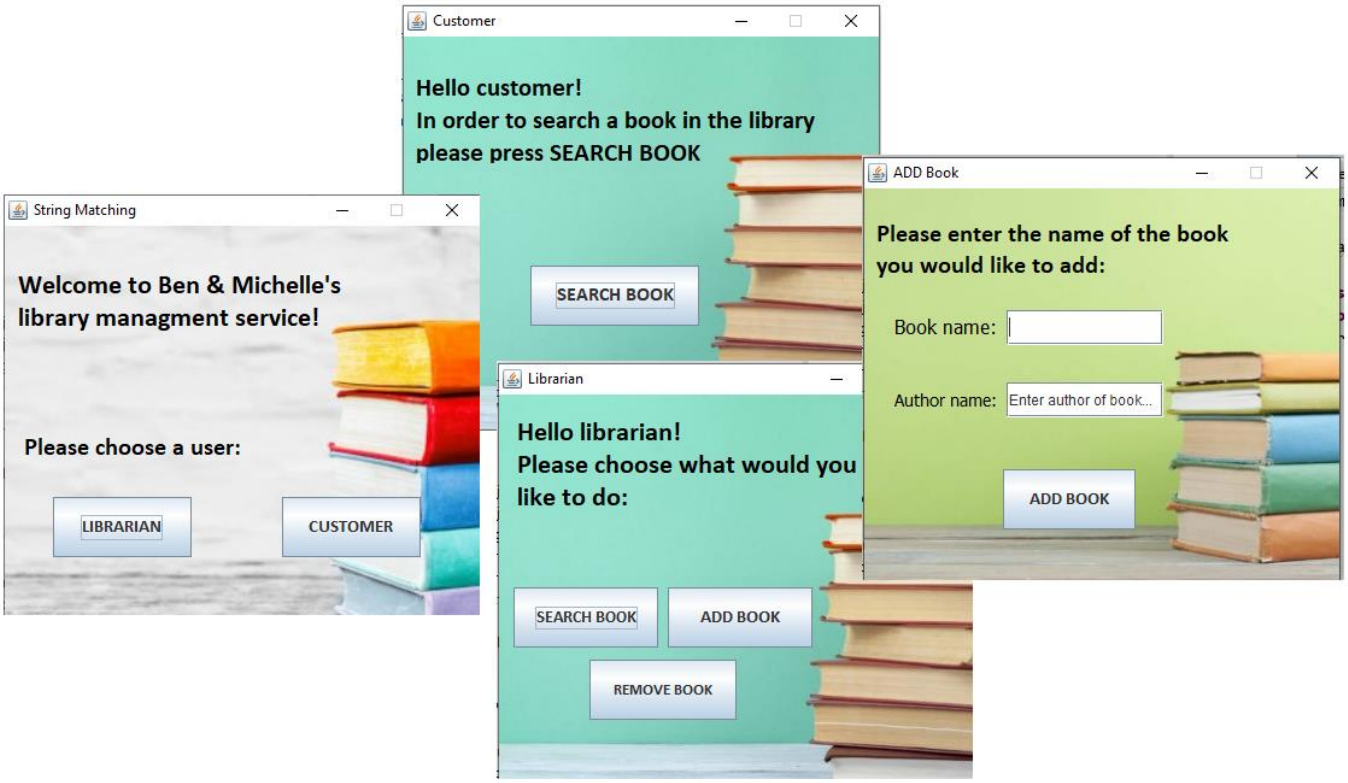
בחלק זה של הפרויקט אתם נדרשים לבנות את החלק הגרפי במערכת שלכם, חלק זה אמור **לשקף את כל הפונקציוליות** שמימשתם בצד השרת ולהנגיש אותה בצורה טובה וידידותית למשתמש.

חלק זה של הפרויקט צריך להיות **פרויקט נפרד לחלוטין** מהפרויקט מהקודם וזה בכדי לאפשר גמישות ובידוד של חלק אחד מהשני, על מנת לבנות את הממשק הגרפי השתמשו **אך ורק** בסיפריות הגרפיות של javaFx המיועדות לכך ושאותן למדנו בכיתה. דוגמאות רבות לבניית ממשקים ורכיבי UI תוכלו למצוא כאן.

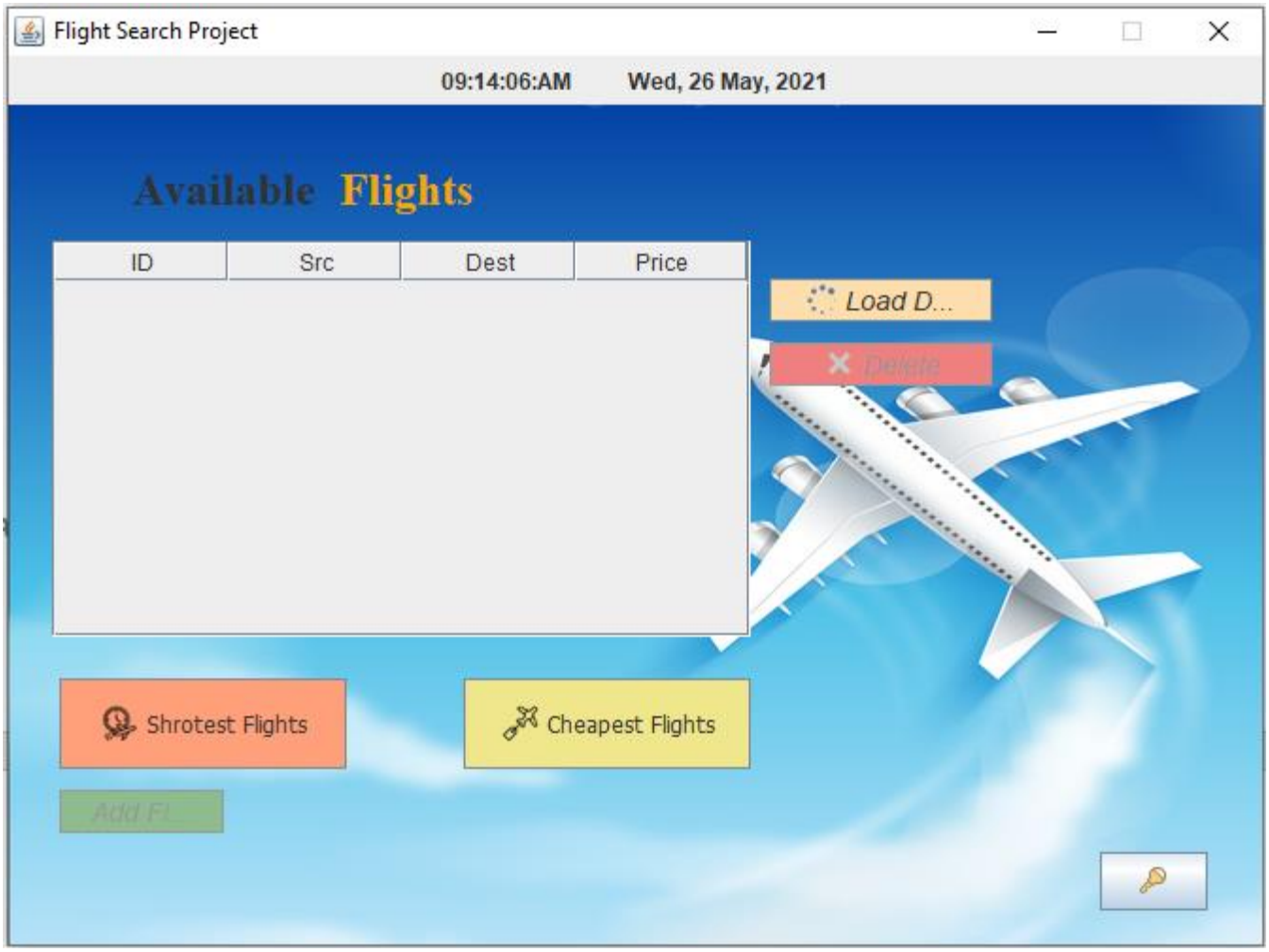
החלק הגרפי שלכם צריך לפנות לשרת על מנת לבצע את פעולות האפליקציה ולכן עליו להכיל מלבד מחלקות ה – UI את הצד שפונה לשרת (Client) ועושה זאת באמצעות האוביקט שנדרש לכך **ה – Socket.**

<u>להלן דוגמאות לממשקי משתמש משנים קודמות:</u>

אפליקציית ספריה המשתמשת באלגוריתם ה – String Matching לחיפוש ספר:



אפליקציית הזמנת טיסות המשתמשת באלגוריתם ה – Shortest Path



מערכות שכוללות שכבת UI ברוב המקרים מבוססות גם בצד ה - client על מודל ה – MVC כפי שנלמד בקורס בהנדסת תוכנה.

למודל ה – MVC יתרונות רבים אך העקרון ה – OOP החשוב ביותר במודל זה הוא עקרון ה - Loosely coupled .

עקרון זה ביסודו מניח כי כל רכיב במערכת או שכבה (שיכולה להיות מיוצגת ע"י מספר רכיבים) צריך להיות בלתי תלוי ברכיב אחר במערכת.

ארכיטקטורת מערכת שבנויה ע"פ עקרון זה צריכה לאפשר גמישות מירבית של **שינוי או החלפה** של כל אחת מהשכבות\רכיבים ללא השפעה כלל או השפעה מינימלית על השכבות האחרות במודל ה - MVC.

בנוסף כל שכבה מבודדת את הלוגיקה שלה ולכן במקרה שקיים באג במערכת השכבה שבה נמצא הבאג לא משפיע על השכבות האחרות,מה שמסייע באיתור וטיפול בבאגים במערכות מורכבות.

בונוס (עד 5 נקודות): ☺

- כיום כפי שהוסבר, ממשק המשתמש הוא חלק חשוב מאוד בכל מוצר תוכנה ועשוי לעיתים להכריע בבחירת הלקוח ולכן כל תוספת בעיצוב הממשק שתעזור להבין את פעולות המערכת בצורה טובה יותר תזכה בנקודות.
- סידור הרכיבים והעיצוב של ה – UI בידכם עצבו ותכננו כרצונכם.

דגשים לסרטון בפרויקט:

ראשית, אורך הסרטון **לא יעלה על 7 דקות**.
אתם רוצים להציג את הפרויקט ולסכם אותו ומעוניינים שאנשים גם יראו אותו אל תשעממו אותם, היו מדוייקים ועניינים ובנוסף הראו התלהבות.

מבנה הסרטון בחלקים:

- **(דקה ראשונה)** הציגו את עצמכם, שם הקורס והפרויקט לאחר מכן הראו את ארכיטקטורת המערכת וספרו מה יעודה. השתמשו בשנים\שלושה שקפים כדי לכלול את כל הנאמר.
- **(3 דקות)** יש לצלם ולהסביר את המחלקות החשובות בפרויקט (הקוד על גבי ה – eclipse/IntelliJ) הכוללות לוגיקה רלוונטית של פעולת המערכת, המחלקות הן:
אלגו' אחד לפחות מבין השניים שמימשתם, המחלקה server, המחלקה service והקישור מצד ה – client לשרת.
בנוסף יש להסביר בקצרה כיצד בדקתם את המערכת (המחלקות הרלוונטיות)
- **(3 דקות אחרונות)** להראות את פעולת המערכת דרך ממשק המשתמש בשני המצבים בחלק זה אינכם צריכים להראות את הקוד אלא את פונקציונליות המערכת ללקוח שישתמש בה

<u>לינקים רלוונטיים ליצירת הסרטון:</u>

לינק להורדת תוכנת ההקלטה - camtasia (Trial):
/http://www.techsmith.com/download/camtasia

סרטון קצר שמסביר איך ליצור סרטון עם תוכנת ה – camtasia ו - audio + אפשרות לצילום עצמי:
http://www.techsmith.com/tutorial-camtasia-8-02-record-your-screen.html

הסבר על שיתוף סרטונים ב – youtube:
http://www.youtube.com/watch?v=N9GKl4xehLM

הגשת הפרויקט

יש לארוז את הפרויקט בקובץ zip. אחד שכולל:

- פרויקט ה – client + פרויקט ה - server
- קובץ עם פרטים אישיים של מגיש\י הפרויקט (ת"ז ושמות פרטיים ומשפחה באנגלית)
- לינק לסרטון בקובץ txt.
- שנים\שלושה שקפים כפי שהוסבר לעיל

במידה ומגישים בזוג אין צורך להגיש פעמיים מספיקה הגשה אחת!!

בהצלחה