

# קורס פיתוח מתקדם ב- JAVA

פרויקט בנית שרת\לקוח

ויחידת אלגוריתמיקה

HIT – מכון טכנולוגי חולון

מרצה: ניסים ברמי

סמסטר קיץ 2020

## תוכן עניינים

3..... הסבר כללי

3..... תיאור הפרויקט

5..... חלק א' – מודול האלגוריתמים

10..... חלק ב' – עקרונות OOP - Design Patterns

15..... חלק ג' – תקשורת ותכנות מקבילי

## 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 בשלוש 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 שלכם 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

להלן ההסבר לצורך בניהול זיכרון ההסבר כיצד מנהלים אותו ולבסוף הדוגמה ל"משפחה" של האלגוריתמים שעושים זאת, עליכם להגדיר את הצורך לאופטימיזציה בפרויקט שלכם בדומה לניהול ה- Cache (אך לא ניתן לבחור את הבעיה לניהול Cache) למצוא משפחה של אלגוריתמים שממשים אופטימיזציה זו ולממש אותם.

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](#).

בחלק הראשון של הפרויקט 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 - `IAlgoNameOfTheAlgoFamily` לדוגמה – `IAlgoCache`. לאחר מכן צרו את שתי מחלקות נוספות **לפחות** שיממשו את ה – interface שיצרתם כל אחת ע"פ הלוגיקה שלה למימוש האלגוריתם.

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

בנוסף אנו נרצה לאפשר גמישות מוחלטת בשינוי מימוש האלגוריתם ולהוסיף מימושים נוספים ל -

`IAlgoNameOfTheAlgoFamily` 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.

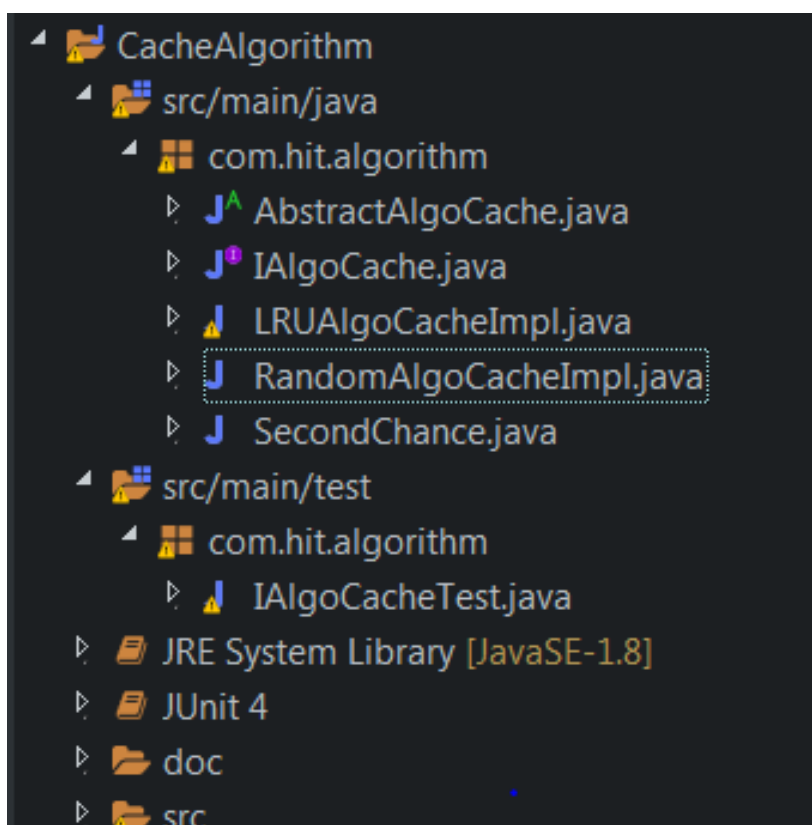
המחלקה האחרונה אותה אתם צריכים לכתוב בחלק זה היא ה – `AlgoNameOfTheAlgoFamilyTest` שכל תפקידה הוא לבדוק את שלושת האלגוריתמים שמימשתם ופעולתם. מחלקה זו תשתמש ב- [JUnit framework](#) ותכיל 2 מטרות @test **לפחות** לכל אחד מהאלגוריתמים.

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 המתוארות ב – packages **במבנה (שימו לב היטב למבנה התיקיות)**

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





## חלק ב' – עקרונות OOP ו - Design Patterns

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 eclipse](#)

#### Creating a New JAR File

To create a new JAR file in the workbench:

1. In the Package Explorer, you can optionally pre-select one or more Java elements to export. (These will be automatically selected in the [JAR Package Specification](#) wizard page, described in Step 4.)
2. Either from the context menu or from the menu bar's **File** menu, select **Export**.
3. Expand the **Java** node and select **JAR file**. Click **Next**.
4. In the **JAR File Specification** page, select the resources that you want to export in the **Select the resources to export** field.
5. Select the appropriate checkbox to specify whether you want to **Export generated class files and resources** or **Export Java source files and resources**. **Note:** Selected resources are exported in both cases.
6. In the **Select the export destination** field, either type or click **Browse** to select a location for the JAR file.
7. Select or clear the **Compress the contents of the JAR file** checkbox.
8. Select or clear the **Overwrite existing files without warning** checkbox. If you clear this checkbox, then you will be prompted to confirm the replacement of each file that will be overwritten.
9. **Note:** The overwrite option is applied when writing the JAR file, the JAR description, and the manifest file.
10. You have two options:
  - Click **Finish** to create the JAR file immediately.
  - Click **Next** to use the JAR Packaging Options page to [set advanced options](#), create a JAR description, or [change the default manifest](#).

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 (System architecture).

בחלק זה של נפתח את יחידות המערכת ונעמוד על הקשר בניהן. 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

המחלקה הראשונה שנממש בחלק זה של הפרויקט היא ה – `YourServiceNameService` :

ה – `YourServiceNameService` עצמו יכיל בתוכו (כ – members) שני רכיבים `IAlgoNameOfTheAlgoFamily`, `IDao` (יתואר בהמשך)

מחלקה זו צריכה לדעת לפנות לשני הרכיבים הללו בכדי לבצע את פעולתה ואת הלוגיקה של המערכת.

הפעולות בהן יתמוך ה- `YourServiceNameService` הן ע"פ צרכי האפליקציה שלכם אך בכל מקרה הוא יתמוך בפעולות הבאות שמירה, מחיקה והחזרה של ה- `DataModels` שלכם.

דבר נוסף שימו לב ש- `YourServiceNameService` עושה שימוש ב- `IAlgoNameOfTheAlgoFamily` שיועבר לו כפרמטר **ב- CTOR** ותישמר אצלו כ- `member`.

השימוש של ה- `YourServiceNameService` 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"וצקות" implementation לאותו API מועברות ל- `YourServiceNameService` 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 ואוחזרו לעודכנו למחקו 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` אליו נתממשק, בפרויקט זה הוא יהיה פשוט **file מקומי** בשם `DataSource.txt` (**ראו בסוף** התרגיל היכן אתם אמורים למקם קובץ זה).

### IDao.html

מחלקה נוספת שעליכם להוסיף היא ה- `DaoFileImpl`, מחלקה זו היא מימוש של ממשק ה- `IDao` והיא כאמור, אחראית לשמור את המידע ל- `File` שניתן לה ע"י המשתמש.

### DaoFileImpl.html

ה- `DaoFileImpl` יקבל את ה- `pathFile` (דרך ה- **CTOR**) אליו תיגש המחלקה בכדי לקרוא ולכתוב. על מנת לכתוב ולקרוא מ- `file` או זקוקים ל- `input/output stream` שמתאימים לקריאה/כתיבה של קבצים, בנוסף או גם זקוקים ל- `streams` שמתאימים לקריאה/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 - `YourServiceNameServiceTest` ותפקידה הוא לבדוק את תקינות פעולת המערכת.

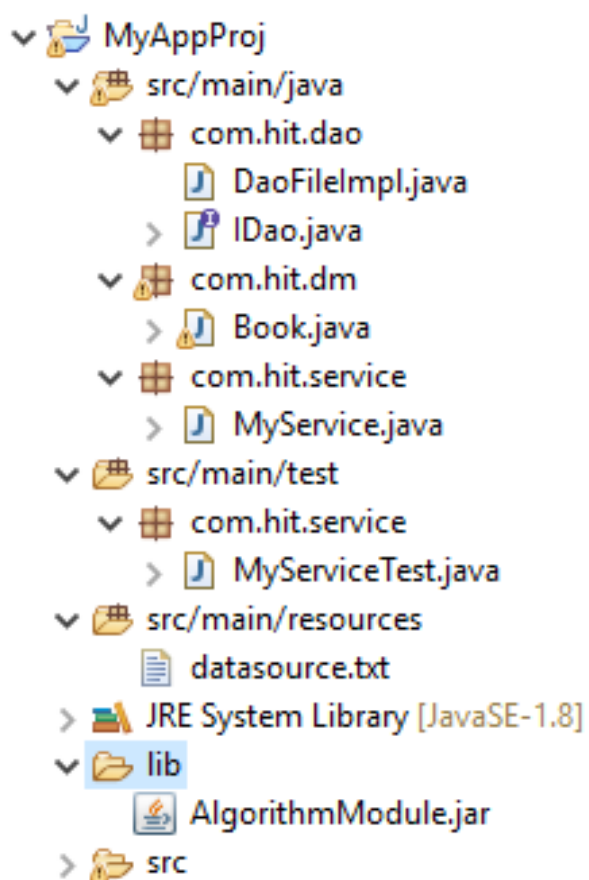
הטסטר צריך לבנות את כל האובייקטים הרלוונטיים ( `IAlgoNameOfTheAlgoFamily`, `YourServiceNameService`, `IDao` ), ויחבר אותם על מנת שיוכל להפעיל את המערכת.

### 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 ותכנות מקבילי



ארכיטקטורת [Client/Server](#) היא אחת מתצורות ההתקשרות 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 אובייקט 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 מהלקוחות ולדעת לנתב אותם כ- **thread נפרד** למחלקות שאחראיות לטיפול בבקשה.

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 במבנה `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](http://gson.jar) כפי שלמדנו בכיתה, לקרוא מה- `Header` של ה- `Request` את השדה `action` ולהעביר אותה למטודה הרלוונטית ב- `Controller`.

המחלקה השלישית ה- `Controller` מטרתה ליצור שכבת הפרדה בין ה- `Service` שלכם לשכבת ה- `Networking`, ממשו את ה- `API` שנתון ע"י קריאה למטודות הרלוונטיות ב- `Service`.

המחלקות רביעית\חמישית הן ה- `Response/Request` מחלקות אלו כפי שתואר לעיל יועברו מהלקוח דרך כל רכיבי השרת עד שמירת\איחזור\מחיקת ה- `Data (Request)` וחזרה `(Response)`.

### להלן דוגמאות למבנה ה- json של ה- Request בחנות ספרים

```
{ "headers":
    {
        {"action":"UPDATE"},
    }
    "body":
    {
        {"name":"Hobbit",
        "content":"Some String Data"}
    }
}
```

---

```
{ "headers":
    {
        {"action":"GET"},
    }
    "body":
    {
        {"name":"Hobbit" }
    }
}
```

```
{ "headers":

    {"action":"DELETE"},

    "body":

    {"name":"Hobbit" }
```

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

```
public class ServerDriver {

    public static void main(String[] args) {
        CLI cli = new CLI(System.in, System.out);
        Server server = new Server(34567);
        cli.addPropertyChangeListener(server);
        new Thread(cli).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):

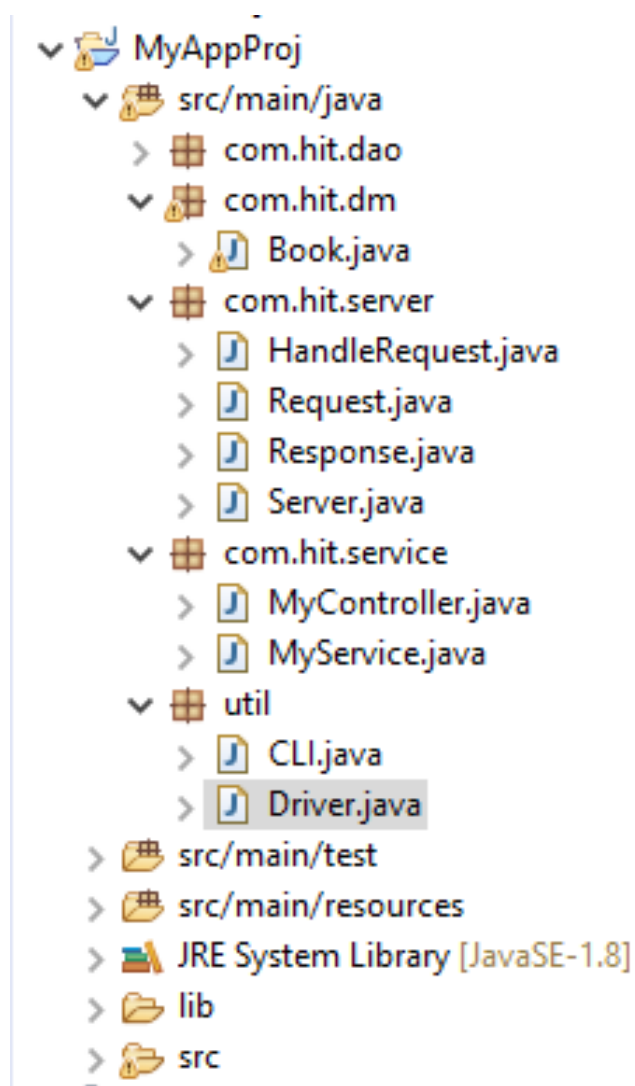
```
Type ref = new TypeToken<Request<Book>[]>>().getType();
Request<DataModel<T>[]> request = new Gson().fromJson(req, ref);
```

- קבעו את ה – Template בעת יצירת ה – HandleRequest, זה ניתן לשינוי כמובן ע"פ מה שתחליטו

```
new HandleRequest<String>(socket, new Controller<Book>())
```

- המערכת שלכם כעת היא מערכת multi threaded דאגו לסנכרן את המערכת היכן שצריך !!!

שימו לב למבנה התיקיות:



רקע (ויקיפדיה) - את המונח "חויית המשתמש" טבע לראשונה דונלד נורמן (Donald Norman) ( באמצע שנות ה-90. בעבר התייחסו בעיקר ל**ממשק משתמש** UI, הממשק בין הטכנולוגיה לאדם לצורך השגת מטרה כלשהי, אך בשנים אלו חילחלה ההבנה כי ממשק המשתמש מהווה רק אספקט מסוים מתוך עולם חויית המשתמש המתייחס גם לעיצוב, ארכיטקטורת המידע, אסטרטגיה, שיווק וטכנולוגיה וגם להיבטים רגשיים, אנושיים ותחושתיים. חויית משתמש טובה יכולה לחסוך מאמצי הטמעה והשקעה בפעילות ניהול ידע לעידוד השימוש וניהול השינוי. בשנים האחרונות תחום חויית 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 "חויית 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 הפונקציונליות** 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 java המיועדות לכך ושאותן למדנו בכיתה Swing. 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) ויוצר את האובייקט שנדרש לכך ה – Socket.

## המלצה !

מערכות שכוללות שכבת UI ברוב המקרים מבוססים 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.

במודל ה-MVC קיימות 3 שכבות עיקריות Model, View, Controller ולכל שכבה תפקיד משלה.

**ה-Model** הוא החלק שאחראי להכיל את ה-data לכן כל פעולה שהמערכת תבצע, נגזרת מהמידע או חלק מהמידע שנמצא ב-Model (ברוב המקרים בעולם האמיתי ישמרו המודלים במסדי הנתונים ה-DB ויקראו משם).

**ה-View** הוא החלק שאחראי על ממשק המשתמש ואינטרקציה עם המשתמש, חלק זה משקף בעיקר את הנתונים ששמורים ב-Model, לעיתים גם משנה ה-View את המודל מפעולות שיוזם המשתמש.

**ה-Controller** הוא החלק שמחבר את שני החלקים הנ"ל. ה-Model וה-View לעולם יהיו חלקים מופרדים ללא אפשרות ל"דבר" אחד עם השני, בכל זאת ללא העברת המידע בין ה-Model ל-View אין משמעות למודל ה-MVC ולכן תפקידו של ה-Controller הוא להעביר את המידע ובנוסף לבצע את לוגיקת החיבור בין שני החלקים במידה וקיימת.

במידה ותבחרו לממש את חלק זה ע"פ מודל ה-MVC כפי שתואר, הצמדו לכללים הבאים:

1. שכבת המודל, תכלול את המחלקות שיפנו ל-Server באמצעות אובייקט ה-Socket, הן גם ידעו לבצע את פעולת ה-parsing של הבקשות והתשובות שיהיו אך ורק במבנה json ולכן ישתמשו ב-jar של ה-Gson בדומה לצד השרת.
2. בודדו את המחלקות ה-model וה-view לחלוטין ואל תאפשרו גישה ישירה בין המחלקות הללו. כל פניה בין שתי מחלקות בשכבות הללו צריכה להיות דרך שכבת ה-Controller.
3. לוגיקת ה-UI כגון: לחיצה על כפתורים, קריאת מחרוזות מרכיבים כמו JTextFeild, JTextArea יש לטפל בשכבת ה-view בלבד.

**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.

**דוגמא משנים קודמות** למבנה התיקיות בפרויקט ה – client:

