קורס פיתוח מתקדם ב- JAVA

פרויקט בנית שרת\לקוח

ויחידת אלגוריתמיקה

HIT – מכון טכנולוגי חולון

מרצה: ניסים ברמי

סמסטר ב 2024

תוכן עניינים

הסבר כללי 3

תיאור הפרויקט 3

חלק א' – מודול האלגוריתמים..... 5

חלק ב' – עקרונות OOP ו – Design Pattern..... 9

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

עליכם להגדיר את הצורך לאופטימיזציה **בפרויקט שלכם** בדומה לניהול ה – Cache למצוא משפחה של אלגוריתמים שממשים אופטימיזציה זו ולממש 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 .

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, 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 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 – **lAlgoNameOfTheAlgoFamilyTest** שכל תפקידה הוא לבדוק את שלושת האלגוריתמים שמימשתם ופעולתם. מחלקה זו תשתמש ב- Junit framework ותכיל 2 מטודות @test **לפחות** לכל אחד מהאלגוריתמים.

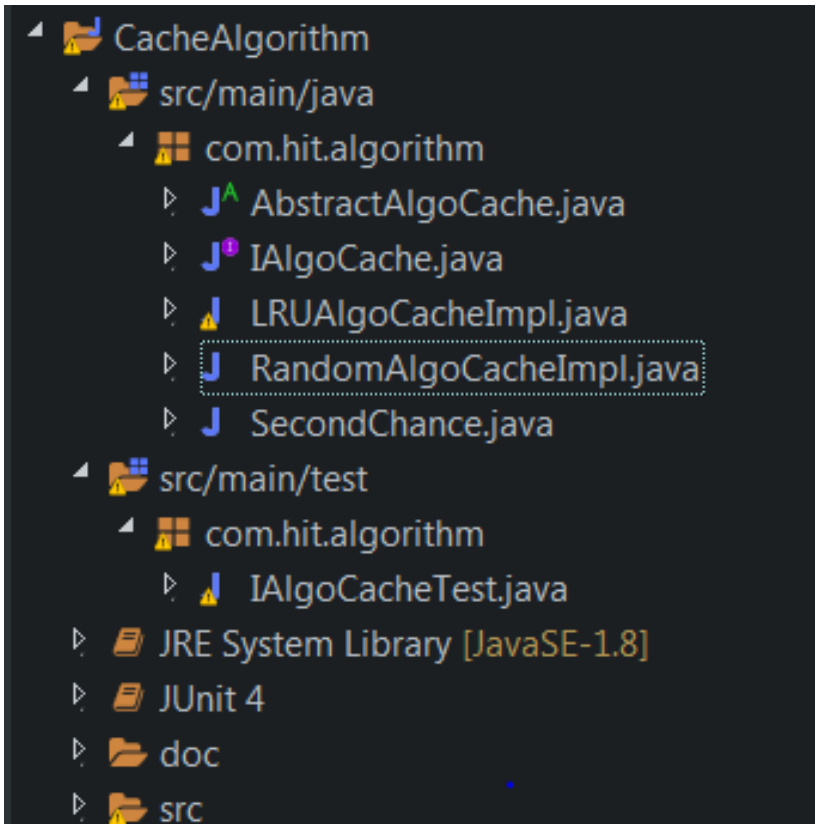
דמיינו שהמוצר הזה צריך להגיע ולרצות את הלקוח (במקרה זה הבודק)

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) :



חלק ב' – עקרונות OOP ו - Design Patterns

ארזו את הפרויקט שלכם **AlgorithmModule** הכולל את מחלקות האלגוריתמים שלכם וה – Test לתוכו

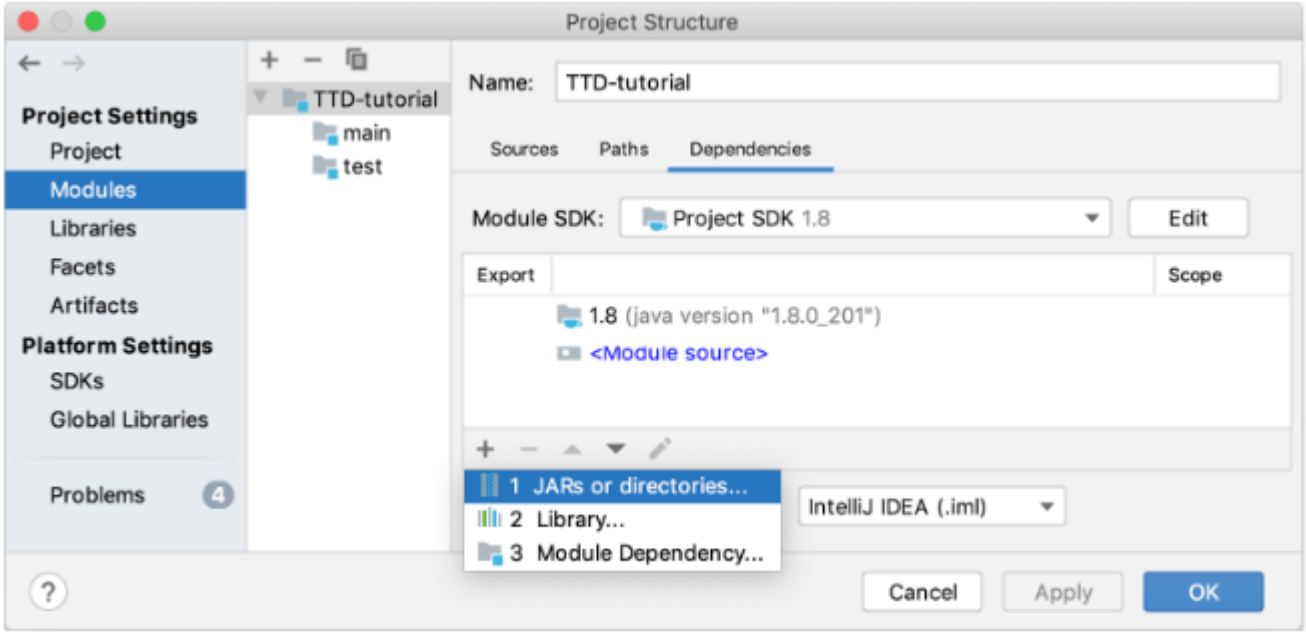
וארזו אותו כ – jar נפרד ע"פ המדריך הבא:

1. Create jar file from 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 בניהן. בנוסף אנו נעצב את המערכת תוך שימוש בעקרונות בסיסיים ב- OOP **<u>ושימוש בפתרונות סטנדרטיים</u>** 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

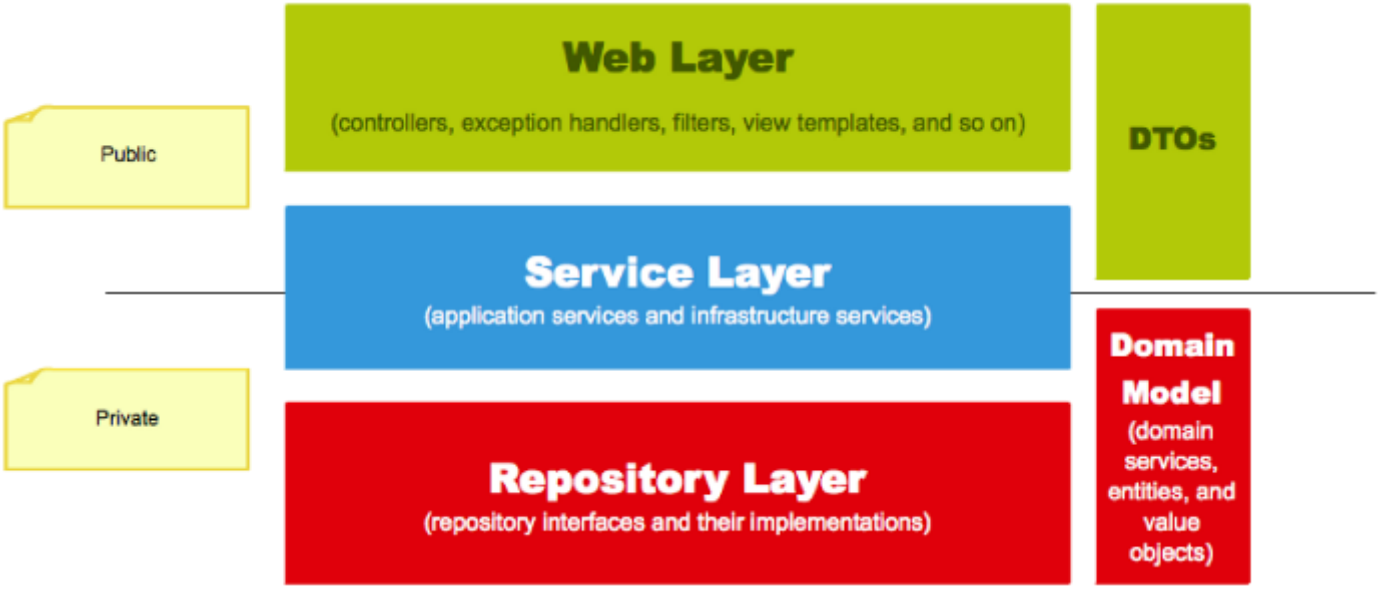
המחלקות הראשונות שנממש בחלק זה של הפרויקט הן ה – **YourServiceNameServices**.

שימו לב להחליף את שם ה – **YourServiceName**Service בשם הרלוונטי לפרויקט שלכם, מחלקות אלו אחראיות על ה – **<u>Business Logic</u>** 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: