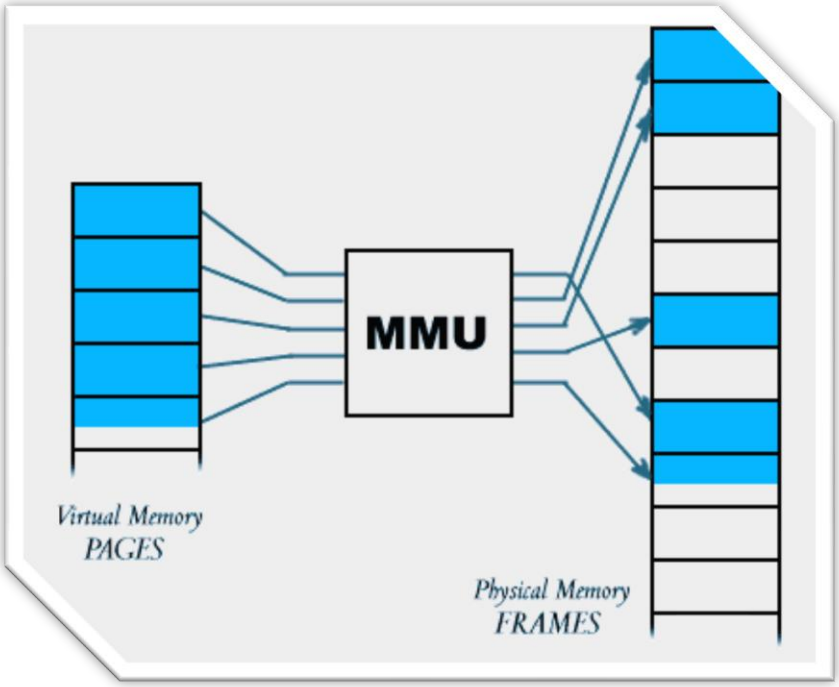
קורס פיתוח אלגוריתמי JAVA

פרויקט יחידת ניהול זכרון – MMU



המסלול האקדמי המכללה למנהל

מרצה: ניסים ברמי

קיץ 2014

תוכן עניינים

פרויקט – יחידת ניהול הזיכרון (Memory management unit) 3

יחידת ניהול זיכרון (MMU) - הקדמה 4

חלק א' – אלגוריתמי החלפה 6

חלק ב' – עקרונות OOP ו - Design Patterns 10

חלק ג' – Multi-Threading 14

חלק ד' – ממשק משתמש 20

חלק אחרון – תקשורת 27

פרויקט – יחידת ניהול הזיכרון (Memory management unit)

פרויקט זה יכלול 5 אבני דרך שכל אחת מהן תהווה מטלה בפני עצמה. לכל מטלה יהיה מועד הגשה ואתם **מחויבים** להגיש 3 מהן (במועדיהן). ניתן להגיש את כולן ובמקרה זה יחושבו ה – 3 בעלות הציונים הגבוהים ביותר. שימו לב שבכל מקרה 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.

שתי המטלות הראשונות נועדו להגשה בבודדים ושלושת המטלות האחרונות כולל הגנת הפרויקט בזוגות (לא ניתן להגיש בשלשות וכן ניתן להגיש בבודדים). בשתי המטלות הראשונות יוצגו 3 אלגוריתמים ואתם תצטרכו לבחור 2 מהם למימוש, הפרויקט הסופי יצטרך לכלול את כל ה – 3 לכן **רצוי** שתחלקו את המימוש עם בן הזוג העתידי, אך זו רק המלצה.

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.

יחידת ניהול זיכרון (MMU) - הקדמה

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, שתפקידה הוא תרגום מרחב הכתובות הווירטואלי אותו "מכיר" המשתמש ומרחב הכתובות הפיסי (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.

מטרת הפרויקט שלנו הוא לממש את יחידת ניהול הזכרון (או משהו דומה*) במערכת ההפעלה בתוכנה בלבד, תוך שימוש בעקרונות תכנות מונחה עצמים, design Pattern, ספריות ומבני נתונים מובנים בשפת JAVA.

במסגרת הפרויקט אנו ניגע, מן הסתם, במספר נושאים הקשורים למערכת ההפעלה, אני אסביר את הנושאים לטובת ביצוע הפרויקט בלבד, ז"א מההסברים כנראה לא תוכלו להבין את כל הנושא ולשם כך נועד הקורס במערכות ההפעלה או במילים אחרות אין לראות בקורס זה כתחליף לקורס מערכות הפעלה ☺ .

* מערכת ההפעלה כחלק מביצוע תהליכים פונה לזיכרון הפיזי של המחשב על מנת לכתוב ולקרוא מידע. הזיכרון הפיזי של המחשב הוא בוודאי יחידה חומרתית ומכיוון שאנו בקורס שלנו מתעסקים בתוכנה בלבד, אנו נאלץ לדמה את המרכיבים החומרתיים בתוכנה.

חלק א' – אלגוריתמי החלפה

בחלק הראשון של הפרויקט אנו נבנה את החלקים הבסיסים של יחידת ניהול הזיכרון (MMU) ונממש מספר אלגוריתמים שתפקידם יהיה ל"ייעץ" ל - MMU כיצד לבצע את תהליך הדפדוף או בצורה מדוייקת יותר אילו דפים יש להעביר מהזיכרון העקרי למשני.

בכל שלב, כאשר המחשב עובד ואיתו מערכת ההפעלה, קיימות מספר תוכניות שרצות במקביל ויש לנהל את הגישה שלהם לזיכרון, בפרויקט שלנו אנו נתייחס לכל תוכנית כתהליך - process (כמובן שתיתכן תוכנית שמריצה מספר processes, אך אנו נפשט זאת) ונממש זאת כאבן דרך נוספת בהמשך הפרויקט.
כל process דורש ממערכת ההפעלה שימוש בזיכרון, ה - process לא יודע להבחין בין זכרון עקרי למשני מבחינתו הדפים אליהם הוא יכתוב או שמהם הוא יקרא, שייכים למרחב הכתובות שמוגדרות לו מראש ואיתם הוא רץ, מרחב הכתובות הללו מוגדר **ככתובות וירטואליות** וסך כל הכתובות הוא **הזיכרון וירטואלי**. הזיכרון הוירטואלי מסתיר בפני ה - process את הזיכרון הפיזי (עקרי ומשני) של המחשב וכך מאפשר לתוכנית לרוץ בצורה פשוטה ורציפה תחת מרחב הכתובות שהוגדר לה (להרחבה בנושא http://en.wikipedia.org/wiki/Virtual_memory).

על מנת שיתבצע תהליך של קריאה או כתיבה של page כלשהו ע"י process כלשהו, יש צורך שהדף הרלוונטי **ימצא בזיכרון העקרי** של המחשב שנקרא Random Access Memory (RAM). הזיכרון העקרי הוא יחידה קטנה ו"יקרה" מבחינת התכולה שלה ולכן בכל זמן נתון, כיוון שקיימים מספר **מוגבל** של דפים ב – RAM (ששייכים בדר"כ למספר תוכניות שונות) צריך לדאוג לנהל זאת בצורה יעילה ונכונה **ולמזער כמה שיותר את פעולות הדפדוף.**

יחידת ה - MMU צריכה להיוועץ או להשתמש באלגוריתם כלשהו על מנת לבצע דפדוף, לכן היא צריכה להחזיק כ- reference יחידה שכל תפקידה הוא מימוש של אלגוריתם החלפה. יחידה זו תיקרא בפרויקט שלנו
IAlgo (Interface). ה – MMU 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 Class). בנוסף אנו נרצה לאפשר גמישות מוחלטת בשינוי מימוש האלגוריתם ולהוסיף מימושים נוספים ל - IAlgo 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 הכרנו ולמדנו בכיתה.
על מנת לייעץ ל – MMU האלגוריתמים שתממשו (Concrete Classes) צריכים להיות מסונכרנים **לגבי הפעולות שמתבצעות על ה – RAM** (בקשת דף, הוספת דף ומחיקת דף), הם צריכים להכיל containers שאותם אתם תבחרו בהתאם לאלגוריתם (מה – containers המובנים של JAVA שלמדנו) שצריך להיות זהה מבחינת הגודל שלו ל – RAM ומבחינת התוכן שלו (**הם לאו דווקא צריכים להכיל אובייקטים זהים**) שכן במידה וה – RAM מגיע לתכולה המלאה שלו צריך להתבצע דפדוף. מעבר לכך במידה ויש דרישה לדף כלשהו שנמצא ב – RAM האלגוריתם שלנו ירצה לדעת מזה (חישבו למה לאחר שתכירו את האלגוריתמים).
שימו לב!! אלגוריתמים אלה מייעצים ל – MMU אותו נממש בהמשך ולא ל - RAM

IAlgo API

```
public interface IAlgo <T>{

    public T get(T t);

    public T add(T t);

    public void remove(T t);
}
```

כפי שניתן לראות הוא מאפשר גמישות מירבית מבחינת ה – Type <T> איתו הוא יעבוד, אתם צריכים לשמור על גמישות זאת גם במימוש שלכם, ז"א החתימות של המטודות באלגוריתמים שלכם צריכות להישאר זהות ל – IAlgo ולקבל ולהחזיר Type <T> גנרי כפי שמחזיר ה - IAlgo, מי שירצה בהמשך להשתמש באלגוריתמים שלכם הוא זה שיקבע את הטיפוס איתו הוא רוצה לעבוד.

הסבר לגבי המטודות:

- המטודת get – יכולה להיקרא גם מטודת touch ובהפעלה שלה ניתן לבדוק האם האלמנט הרלוונטי קיים ב – RAM ובכל מקרה לעדכן את האלגוריתם שדף זה התבקש ע"י ה - MMU.
- המטודת add – מוסיפה את האלמנט שמתקבל ומסירה אלמנט רק במידה וה – RAM מלא.
- המטודת remove – מסירה אלמנט.

אלגוריתמי ההחלפה אותם אתם צריכים לממש בפרויקט שלנו הם:

FIFO - https://www.youtube.com/watch?v=KejyTiATz18

LRU - https://www.youtube.com/watch?v=I9_BpSXBodU

MFU – http://courses.teresco.org/cs432_f02/lectures/14-memory/14-memory.html (בסוף קיים קלט לדוגמא)

צרפתי לינקים לסרטונים והסברים לגבי האלגוריתמים אך אתם כמובן מתבקשים לחפש ולהעמיק ככל הנדרש בנושא.

אנו כאמור, מספקים ל – MMU שלנו אלגוריתמי החלפה של PAGES בין ה - RAM שהוא הזיכרון העקרי לזכרון המשני (שהוא בדר"כ ה – HardDisk ועליו נדבר בהמשך) נגדיר כעת את היחידות הללו ונציג את ה – API שלהם.

PAGE API:

```
public class Page <T>{

    private int pageId;
    private T content;

    public Page(int id, T content){}
    public int getPageId() {}
    public void setPageId(int pageId) {}
    public T getContent() {}
    public void setContent(T content) {}
    @Override
    public int hashCode() {}
    @Override
    public boolean equals(Object obj) {}
    @Override
    public String toString(){}
}
```

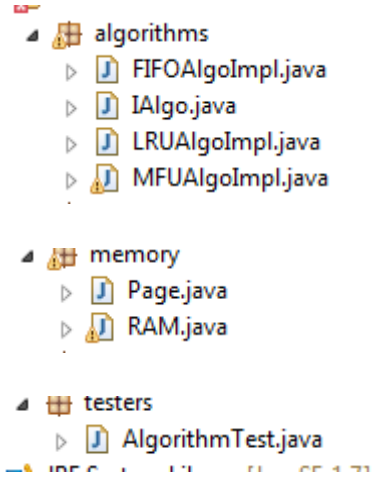
RAM API

```
import java.util.Map;

public class RAM {

    private Map<Integer, Page<byte[]>> pages;


    public RAM(int initialCapacity){}

    public Page<byte[]> getPage(int pageId){}

    public void addPage(Page<byte[]> addPage){}

    public void removePage(Page<byte[]> removePage){}

    public Page<byte[]>[] getPages(Integer[] pageIds){}

    public void addPages(Page<byte[]>[] addPages){}

    public void removePages(Page<byte[]>[] removePages){}
}
```

כפי שניתן לראות קיימות שגיאות קומפילציה שצריכות להסתדר ברגע שתוסיפו את הקוד הרלוונטי, בנוסף אתם יכולים להוסיף מטודות private ע"פ הצורך. שימו לב שני רכיבים אלא כולל המחלקות של אלגוריתמי ההחלפה הינם רק חלקים מה – MMU אותו נממש בהמשך.

מחלקה האחרונה אותה אתם צריכים לכתוב בחלק זה היא ה – AlgorithmTest שכל תפקידה הוא לבדוק את שלושת האלגוריתמים שמימשתם ופעולתם. מחלקה זו תכיל מטודת **main** שתיצור אובייקטים לכל אחד מהאלגוריתמים תבנה קלט שיהיה רשימה של String (גדולה מה – Capacity של ה- container הפנימי של האלגוריתם) ותייצר תסריטים לבדיקת האלגוריתמים. חשבו שהמוצר הזה צריך להגיע ולרצות את הלקוח (במקרה זה הבודק) לכן ככל שתבדקו יותר תגיעו לרמה טובה יותר ומוצר טוב יותר.

לבסוף ארזו את כל 6 המחלקות (לאחר שבחרתם 2 מ – 3 האלגוריתמים) המתוארות ב – packages במבנה ובשמות הבאים (שימו לב לשמות האלגוריתמים):



חלק ב' – עקרונות OOP ו - Design Patterns

חלק חשוב ובלתי נפרד בפיתוח תוכנה הוא ה – design. לאחר שמובנות הדרישות\צרכים אותן צריכה לספק המערכת **חייבים** לעבור לשלב שבו מעצבים את המערכת (System architecture) לפני שמעבירים את הדרישות לקוד.
מבחינת הפרויקט שלנו ,דרישות המערכת הוצגו בחלק של ההקדמה ,אך כיוון שלא ניתן להגדיר את שלב הדרישות כתרגיל, שילבנו גם חלק של פיתוח יחידות בסיס.

בחלק זה של התרגיל אנו נוסיף את שאר יחידות המערכת ונעמוד על הקשר בניהן. בנוסף אנו נעצב את המערכת תוך שימוש בעקרונות בסיסיים ב- OOP ושימוש בפתרונות סטנדרטיים לבעיות מוכרות שהם ה – design patterns. העקרון הראשון והבסיסי אליו נצמד הוא היכולת לתת גמישות להרחבה (מבחינת הוספת יכולות) בצורה פשוטה וקלה אך באותו אופן גם לסגור את המערכת לשינויים (מבחינת ה – API) עקרון זה ב - OOP נקרא:

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

המחלקה הראשונה שנממש בחלק זה של הפרויקט הוא ה – MemoryManagementUnit(MMU). ה – MMU עצמו אמור להכיל בתוכו (כ – members) שני רכיבים בלבד: RAM, IAlgo. בנוסף הוא צריך לדעת לפנות לדיסק הקשיח (hard disk) במטרה לקרוא ולכתוב דפים. ה – MMU כיחידה מייצגת לכאורה מערכת מאוד חכמה ומורכבת שיודעת לנהל זיכרון (למצוא, להחליף ולהסיר דפים) אך הכל דרך API של **מטודה אחת בלבד**
(מדהים !!!).

ה – API של ה – MMU :

```
public class MemoryManagementUnit {

    private IAlgo<Integer> algo;
    private RAM ram;

    public MemoryManagementUnit(int ramCapacity, IAlgo<Integer> algo){
        ...
    }

    public Page<byte[]>[] getPages(Integer[] pageIds){

        ...
    }
```

כאמור, הפעולה היחידה שה – MMU יודע לעשות הוא להחזיר דפים ע"פ pageIds שניתנים לו כמערך.
בכדי להחזיר את אותם דפים עליו להפעיל את הלוגיקה שתיארנו בהרחבה בהקדמה, נתאר אותה
ב – Pseudo code הבא:

```
Array pages getPages(Array pageIds)

    For i := 0 to pageIds.length do

        If RAM not contains pageId

            If RAM is not full

                (pageFault)

            Else

                Do logic of full RAM (pageReplacement)

        Else

            Ialgo get pageId

    End Loop;

    Return Array pages from RAM;

End Func
```

כפי שניתן לראות ב - Pseudo Code עושה ה – MMU שימוש ב – IAlgo. ה – IAlgo מועבר כפרמטר ב – CTOR ל – MMU ונשמר אצלו כ - reference בתור – member (MMU API). השימוש של ה – MMU ב- IAlgo הוא שימוש ב – API בלבד (get, add, remove), ללא הבנה כיצד מומש אותו API וללא אפשרות לשנות אותו (closed for modification). אותן מחלקות ש"יוצקות" implementation לאותו API מועברות ל – MMU מבחוץ ולכן ניתן בקלות להעביר סוגים שונים של מימושים ולהוסיף בכל שלב אחרים ובאותה צורה להעביר גם אותם (open for extension). בכך השלמנו את ה – **strategy pattern** ושמרנו על עקרון ה – open/close principal.

הרכיב הנוסף שבו ה – MMU עושה שימוש הוא ה – Hard Disk(HD) וגם לו נכתוב מחלקה נפרדת.
את ה – HD איננו יכולים לדמות בחומרה (לפחות לא במסגרת קורס זה) לכן כתחליף לכך נשתמש ב - file
ונכתוב לשם את המידע שלנו (דפים).

ה – HD הוא רכיב הנגיש לרכיבים נוספים במחשב מלבד ה – MMU, לדוגמה בעת התקנת תוכנות נוספות משתמש בו ה – OS לטובת הקצאת זכרון (טווח כתובות פיזי) נוסף שפנוי. מצב מעיין זה שבו קיים במערכת resource שאליו רוצים **גישה אחת בלבד**, היא בעיה נפוצה בעולם התוכנה והדרך להתמודד איתה היא באמצעות שימוש ב – **singleton pattern**.

ה - singleton pattern כפי שלמדנו בכיתה, מאפשר לנו להגביל את מספר המופעים (instances) של אובייקט בזכרון לאחד ופנייה גלובאלית של הרכיבים המשותפים במערכת למופע זה.

ה – HD **חייב** להחזיק בתוכו שני constants עיקריים:

_SIZE - גודל ה – HD במספר דפים, למשל בדוגמה שלהלן מספר הדפים שאותן יכיל ה – HD בעת **היצירה** הוא 1000 דפים, עם ids בסדר עולה מ - 1 עד - 1000

DEFAULT_FILE_NAME- שם הקובץ אליו ה – HD יכתוב\יקרא דפים.

מעבר לכך ה - HD יאפשר את ה – API הבא:

```
public class HardDisk {
    final static String DEFAULT_FILE_NAME = "hdPages.txt";
    final static int _SIZE = 1000;

    public static HardDisk getInstance() {
    }

    public Page<byte[]> pageFault(int pageId){
    }


    public Page<byte[]> pageReplacement(Page<byte[]> moveToHdPage, Integer moveToRamId){
    }
```

חישבו כיצד להשלים את יצירת ה – HD על מנת להפוך אותו להיות singleton וכיצד לטפל ב – exceptions רלוונטיים בצורה נכונה .

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.

הצורה שבה הוסיפו בכל שלב יכולות נוספות של קריאה וכתיבה מבלי **לפגוע במה שקיים** ויותר מכך ע"י **הרחבה של מה שקיים** נעשה באמצעות ה – design pattern ← **decorator**.

אנו נכתוב מחלקה שתסייע בצורה דומה לקריאה של ה – HD **בלבד** ע"י שימוש ב – decorator pattern.

מחלקה זו תקרא HardDiskInputStream והיא תספק את ה – API הבא:

```
public Map<Integer, Page<byte[]>> readAllPages(){
    ....
}

public Page<byte[]> readSinglePage(Integer pageId){
    ....
}
```

חשבו מאיזה stream תרצו לרשת\להרחיב (extends) ובמקרה זה גם טפלו ב – exception רלוונטיים.

גם בחלק זה המחלקה האחרונה אותה אתם צריכים לכתוב היא טסטר שיבדוק את יחידת ה – MMU.

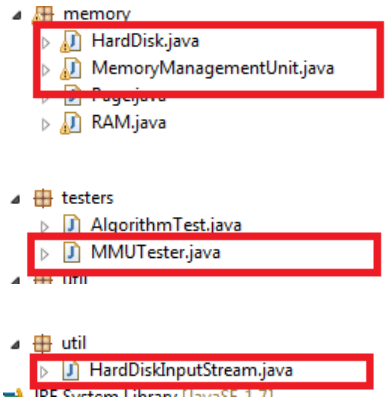
שם המחלקה תהיה – MMUTest ותפקידה הוא לבדוק את תקינות פעולת המערכת. טסטר זה יכיל מטודת **main** שתיצור את כל האובייקטים הרלוונטיים (IAlgo, MemoryManagementUnit השאר תיצור המערכת בעצמה), ותחבר אותם על מנת שתוכלו להפעיל את המערכת.

נקודות חשובות בעת בדיקת המערכת:

- קבעו את גודל ה – HD להיות גדול מה – RAM על מנת שתהיה משמעות לעבודת ה – MMU.
- אין צורך בטסטר זה לבדוק שוב את כל האלגוריתמים, מספיק להשתמש באחד (מהסיבה שכבר בדקתם אותם בתרגיל הקודם).
- בתחילת עבודת המערכת ה – RAM אינו מכיל דפים והוא מתמלא תוך כדי פעולת המערכת.
- **פונקציית עזר:** על מנת להדפיס מערך של byte, לאחר שאתחלתם אותו במספרים השתמשו

```
System.out.println(Arrays.toString(new byte[]{2,3,77}))  ->  [2,3,77]
```

לבסוף ארזו את כל 4 המחלקות **החדשות** (באדום) המתוארות ב – packages במבנה ובשמות הבאים:



חלק ג' – Multi-Threading

מזל טוב !! בשלב זה של הפרויקט יש לנו מערכת MMU שעובדת ויודעת לנהל את הזיכרון בצורה טובה תוך ביצוע פעולת **דפדוף** יעילה ושקופה למשתמש, אך כעת אנו רוצים גם להפעיל את המערכת בצורה דומה לאיך שהיא עובדת במקומה הטבעי (כחלק ממערכת ההפעלה).

בחלק הקודם של הפרויקט (חלק ב') הדרך שהפעלנו את ה - MMU היה באמצעות ה - MMUTester שזוהי דרך לגיטימית להפעלת המערכת שמדמה לנו **תהליך אחד בלבד** שמבקש דפים מהמערכת ובצורה טורית, לכן בעיות מהסוג של Race Condition לא קיימות בצורת הפעלה זו.

לא כך בעולם האמיתי, יחידת ניהול הזכרון היא חלק אינטגרלי ממערכת ההפעלה ומהותה של מערכת ההפעלה להיות multi-tasking כלומר לדעת לנהל מספר תהליכים בו זמנית, לכן תפקידנו לדמות צורת פעולה זו ולהערך לכך בהתאם.

על מנת לרוץ בסביבה בה קיימים מספר תהליכים שרצים בו זמנית, יש צורך להגדיר מהו תהליך (Process).

*תהליך (ויקיפדיה) - הוא מופע של תוכנית מחשב שמופעל על ידי מערכת מחשב שיש לה היכולת להפעיל מספר תהליכים בו זמנית. תוכנית מחשב היא בעצמה רק אוסף פקודות, בעוד שתהליך הוא **ההפעלה של אותן פקודות**.*

בכדי לדמות תהליך שפונה למערכת ה – MMU, אנו נשתמש בפרויקט שלנו במיני תהליך הנקרא Thread. למדנו כי ב - JAVA השימוש ב – Threads הוא חלק מובנה בשפה מה שיאפשר לנו בצורה פשוטה ונוחה לבנות תהליך ולהריץ אותו. ישנן שתי שיטות ליצור thread ב – JAVA ואנו נבחר בשיטה המועדפת של – implements Runnable.

התהליך שלנו שהוא המחלקה הראשונה אותה נכתוב בחלק זה והיא תקרא Process. מחלקה זו תבצע שתי פעולות בלבד אך **במחזורים** (ProcessCycles אותם נסביר בהמשך):

1. **לבקש דפים מה – MMU וכאשר הוא מקבל אותם לכתוב אליהם data שמתקבל ע"י ה - ProcessCycle**
2. **לישון מספר שניות שניתן לו שוב ע"י ה - ProcessCycles**

שתי הפעולות הללו נעשות כחלק מפעולת ה – Thread ולכן מתבצעות ממטודת הפעולה של ה – Thread שהיא **ה – run()**.
המחלקה process מחזיקה שלושה members:
MemoryManagementUnit mmu – מוחזק כ - referece למערכת ה – MMU שלנו על מנת לקבל דפים.

int ProcessId– מזהה חח"ע של ה – process

ProcessCycles processCycles – מה **<u>שמזין</u>** את התהליך בדפים שאותו עליו לבקש, במידע אותו עליו לכתוב לדפים אלו ובזמן שינה בכל מחזור.

Process API:

```
public class Process implements Runnable{

    private int id;
    private ProcessCycles processCycles;
    private MemoryManagementUnit mmu;

    public Process(int id, MemoryManagementUnit mmu, ProcessCycles processCycles) {
    }

    @Override
    public void run() {
    }
```

שלוש מחלקות נוספות **ופשוטות** שאנו נממש יהיו אחראיות על מחזורי ה – processes.

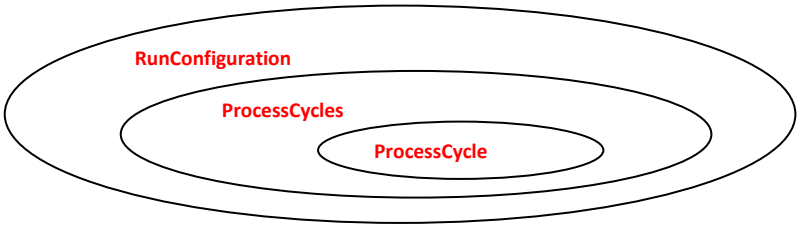
נתאר מהי מחזוריות של תהליך - כל תהליך כפי שנאמר יודע לבצע שתי פעולות בלבד לבקש דפים (ולכתוב אליהם) ולישון מספר שניות. בדומה לתהליך אמיתי במערכת ההפעלה הפעולות הללו חוזרות על עצמן, ז"א כל תהליך (נניח, תהליך א') מבקש ממערכת ההפעלה דפים בכדי לכתוב או לקרוא מהן, לאחר שהוא מקבל אותם ועובד איתם **מתזמנת** מערכת ההפעלה תהליך נוסף (תהליך ב') **מתור** של תהליכים שמחכים ובזמן זה נכנס תהליך א' למצב שינה למספר שניות **בקשת הדפים וקבלתם + שלב השינה מוגדר כמחזור אחד**. בשלב מסויים תהליך א' מסיים את מצב השינה שלו ורוצה לבצע שוב קריאה וכתיבה לדפים (לא בהכרח לאותן דפים) ואז הוא נכנס לתור של שאר התהליכים עד שמערכת ההפעלה תתזמן אותו שוב ואז הוא יתחיל את המחזור הבא שלו.

המחלקה הבסיסית שלנו שתדמה מחזור (**אחד**) של תהליך תקרא ProcessCycle. מחלקה זו **לא** תכיל לוגיקה אלא פשוט תחזיק שלוש תכונות (members) ותדע להחזיר ולאתחל אותן (getter/setter).

ProcessCycle API

```
public class ProcessCycle {

    private List<Integer> pages;
    private int sleepMs;
    private List<byte[]> data;

    public ProcessCycle(List<Integer> pages, int sleepMs, List<byte[]> data) {
    }

    /**
     * getter/setter of pages and sleepMs
     */

    @Override
    public String toString() {
    }
```

שתי המחלקות הנוספות הקשורות למחזוריות (ProcessCycles, RunConfiguration), פשוט מכילות רשימה אחת של השניה במבנה המתואר ויודעות להחזיר ולאתחל את אותן רשימות:



ProcessCycles API

```
public class ProcessCycles {

    private List<ProcessCycle> processCycles;

    public ProcessCycles(List<ProcessCycle> processCycles) {
    }

    /**
     * get/set of processCycles
     */

    @Override
    public String toString() {
    }
```

RunConfiguration API

```
public class RunConfiguration {

    private List<ProcessCycles> processesCycles;

    /**
     * @return the processesCycles
     */
    public List<ProcessCycles> getProcessesCycles() {
    }

    /**
     * get/set of processesCycles
     */

    @Override
    public String toString() {
    }
```

כפי שניתן להבין מה – APIs של מחלקות אלו, המטרה היחידה של המחלקות הללו היא **להחזיק מידע בלבד** ללא שימוש בלוגיקה כלל, אך על מנת שנוכל להפעיל את המערכת שלנו (שזוהי מטרת התרגיל) אנו זקוקים לאוביקטים אלו כשהם מאותחלים ורצוי שניתן יהיה לשנות את המידע בקלות כדי שידמה מספר תסריטים כמה שיותר רחב.
בתרגיל זה אני אספק לכם את **התוכן** של אוביקטים אלו ללא צורך בכתיבה של מחלקה כלשהי ב – JAVA ואפילו ללא ידיעה כיצד אתם ממשתם את המחלקות שלכם.

הצורה שאנו נבצע את העברת המידע שלנו הוא ע"י הסכמה על מבנה האוביקטים שיועברו בנינו ובאמצעות **קבצי טקסט בלבד**.
חשבו על קבצי הטקסט הללו כבסיס משותף שמחבר בין שתי מערכות ללא צורך בידיעה של כיצד כל מערכת משתמשת בהם.
קבצי הטקסט האלו יהיו בפורמט JSON שהוא הפורמט המקובל והשכיח ביותר כיום להעברת מידע בצורה טקסטואלית. פורמט זה הוא פשוט וקל שמורכב **תמיד** מ – key, value.

1. בכדי לייצג את האובייקט ProcessCycle נשתמש במבנה הבא:

```
"pages": [],
"sleepMs": int,
"data": [[],[],[]]
```

שימו לב כי שמות ה – keys זהים לתכונות האוביקט שלכם וכי ה – values מתאימים לערכים שהתכונות שלכם יכולות לקבל:

- Pages – מייצג מערך של int שהם בעצם ה – ids שהתהליך רוצה לקבל ב – cycle הנוכחי.
- sleepMs – מספר השניות שהתהליך יישן ב – cycle הנוכחי.
- Data - מערך של מערכים של byte. לכל דף במערך ה – pages ירשם המערך שקיים ב – מערך ה – data בהתאמה.

2. כדי לייצג את האובייקט ProcessCycles נשתמש במבנה הבא:

```
"processCycles": [{
                    "pages": [],
                    "sleepMs": int,
                    "data": [[],[],[]
                    ]
                  },{
                    "pages": [],
                    "sleepMs": int,
                    "data": []
                  }
                  ]
```

רשימה של ProcessCycle

3. כדי לייצג את האובייקט RunConfiguration נשתמש במבנה הבא:

```
{ "processesCycles": [
    {
        "processCycles": [
            {
                "pages": [],
                "sleepMs": int,
                "data": []
            }
        ]
    },
    {
        "processCycles": [
            {
                "pages": [],
                "sleepMs": int,
                "data": [
                    []
                ]
            }
        ]
    }
  ]
}
```

רשימה של ProcessCycles

בגלל שכיחותו של פורמט ה – JSON נכתבו מספר readers/writers לטובת עבודה עם קבצי ה – JSON על מנת לאפשר עבודה קלה ופשוטה בין היתר גם ב - JAVA. ספריות ה - open source שאנו נשתמש בהן לטובת העבודה עם קבצי ה – JSON הם Gson (נכתבו ע"י חברה קטנה שנמצאת בשלבים הראשונים שלה ותורמת רבות לעולם ה – open source והיא Google).

בכדי להשתמש בספריות אלו אנו נוריד תחילה את קובץ ה – JAR המכיל מחלקות אלו ולאחר מכן נוסיף אותו לספריות הפרויקט שלנו בצורה הבאה:

Right click on MMUProject → Build Path → Configure Build Path… → Libraries → Add External JARs → Select gson.jar

לאחר שצרפתם את **ספריות** ה – Gson ניתן ליצור כעת את **האובייקט** Gson ולהשתמש בו בקלות לקריאה וכתיבה של אוביקטים לקבצי JSON ובחזרה לאוביקטים כמובן במידה ותוכן ה – JSON תקין ומותאם לאוביקט הרלוונטי.

לדוגמה האוביקטים שלנו (שימו לב במה משתמש ה – JsonReader וע"פ איזה Design Pattern):

```
ProcessCycle processCycle = new Gson().fromJson(new JsonReader(new FileReader(CONFIG_FILE_NAME)), ProcessCycle.class);
ProcessCycles ProcessCycles = new Gson().fromJson(new JsonReader(new FileReader(CONFIG_FILE_NAME)), ProcessCycles.class);
RunConfiguration runConfiguration = new Gson().fromJson(new JsonReader(new FileReader(CONFIG_FILE_NAME)), RunConfiguration.class);
```

המחלקה האחרונה שאנו נכתוב בחלק זה של הפרויקט תקרא ה – MMUDriver. מחלקה זו היא למעשה החלק שמחבר לנו את כל **רכיבי המערכת וגם מפעיל אותם**. מחלקה זו תשמש בין היתר גם כטסטר לחלק זה של הפרויקט בכך שהיא תיצור לנו את כל התהליכים תאתחל אותן ותדאג להפעיל אותם אך לפני כן היא גם תיצור את ה - MMU ותדאג לעביר אותו כ – reference לכל אחד מה – processes כדי שיוכלו לעשות בו שימוש ולבקש את הדפים בהם יש להם צורך.

פעולת ה – MMUDriver מתבצעת כולה במטודת **ה – main()** והקוד שלה מתואר להלן:

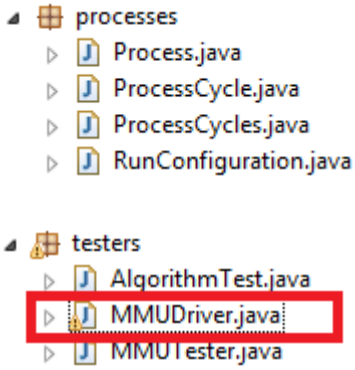
```
MemoryManagementUnit mmu = new MemoryManagementUnit(5, new LRUAlgoImpl<Integer>(5));
RunConfiguration runConfig = readConfigurationFile();
List<ProcessCycles> processCycles = runConfig.getProcessesCycles();
List<Process> processes = createProcesses(processCycles, mmu);
runProcesses(processes);
```

הסבר, לפעולת ה – MMUDriver והקוד שתואר:

- יצירת ה – MMU בשורת קוד הראשונה היא configurable ז"א את גודל ה – RAM ניתן לשנות אך לקבוע גם את גודל ה – IAlgo בהתאם. כמו כן ניתן לשנות את האלגוריתם איתו תעבוד ה – MMU בדוגמה הנוכחית זהו אלגו' ה - `LRUAlgoImpl`
- בשורת קוד השניה מתבצעת קריאת קובץ ה – JSON (יסופק בנפרד) ויצירת האוביקט – `RunConfiguration` כפי שהוסבר למעלה
- בשורת הקוד הרביעית יש ליצור את התהליכים ולבצע את החיבור בינם לבין ה – MMU. שימו לב שמספר התהליכים שיווצרו הוא כמספר `ProcessCycles` (חשבו מדוע...)
- שורת הקוד האחרונה היא המטודה שאחראית על הפעלת התהליכים והמערכת כולה.

התהליכים חייבים **לעבוד במקביל** (שהרי לשם כך התכנסנו בתרגיל זה) חשבו איך לנהל אותם (רמז: בכלים מובנים כפי שלמדנו) ובנוסף **דאגו לסנכרן בצורה נכונה את האוביקט איתו עובדים התהליכים שהוא ה – MMU** ישנם מספר דרכים שוב כפי שלמדנו.

לבסוף ארזו את כל 5 המחלקות **החדשות** המתוארות ב – packages במבנה ובשמות הבאים:



חלק ד' – ממשק משתמש

(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 הפלך למשמעותי בכל פתרון מחשובי בכלל, ופתרונות ניהול ידע בפרט, ונתפס כמרכיב קריטי והכרחי (Critical Success Factor).

אחד האנשים שתרם 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 לשקף את **הפעולה של המערכת** בצורה ויזואלית והספריות בהן אנו נשתמש לצורך כך הן ספריות ה - SWT שלמדנו בכיתה.
אבל לפני שנוכל לשקף את פעולות המערכת ולבנות ממשק מתאים, יש צורך ראשית לרשום את הפעולות בצורה מובנית בתוך המערכת וע"פ סדר התרחשותן וזה יעשה כפי שנעשה בצורה סטדנרטית בכל המערכות באמצעות **Logger**.

ראשית נסביר מהו Logger, מה תפקידו וכיצד הוא יסייע לנו בחלק זה של בפרויקט. ה – Logger הוא רכיב תוכנה (מחלקה) שתפקידו לספק API שמאפשר כתיבה ותיעוד של הודעות (messages) מהמערכת או מרכיבים מסויימים במערכת. ה – Logger הוא רכיב חשוב וקריטי למערכות מורכבות ולאו דווקה, בכך שהוא מאפשר להבין את פעולת המערכת בצורה תמציתית וסלקטיבית ע"י קריאה וניתוח של קבצי ה – Log אותם הוא מייצר.
השימוש ב – Logger הוא מובנה וחלק אינטגרלי מכתיבת הקוד של המערכת ובאחריותם של המפתחים לעשות בו שימוש ובצורה נכונה.
לדוגמה: שגיאות אותן ניתן לצפות ולטפל (כחלק מ try/catch) רצוי שיכתבו ל – Log לטובת טיפול עתידי.

המחלקה הראשונה אותה נכתוב בחלק זה של הפרויקט היא ה – MMULogger ואנו נשתמש במחלקה זו על מנת לתעד את פעולות מערכת ה - MMU שלנו. מחלקה זו בדומה למחלקות אחרות שכתבנו ובהתאם לעקרונות שלמדנו ב – OOP תתבסס על מחלקות מובנות ב – JAVA תחזיק אותם (composite), תעשה בהם שימוש
ו - Customization.
המופע (instance) של ה – Logger צריך להיות **אחד והפניה אליו זהה וגלובאלית מכל רכיבי המערכת** ולכן יש לדאוג לכך.

MyLogger API

```java
public class MMULogger {

    final static String DEFAULT_FILE_NAME = "log.txt";
    private FileHandler handler;

    private MMULogger(){
        try {
            handler = new FileHandler(DEFAULT_FILE_NAME);
            handler.setFormatter(new OnlyMessageFormatter());
        } catch (IOException e) {
            System.err.println("Cannot create handler" + e.getMessage());
        }
    }

    public void write(String command, Level level){
    }

    public class OnlyMessageFormatter extends Formatter
    {
        public OnlyMessageFormatter() { super(); }

        @Override
        public String format(final LogRecord record)
        {
            return record.getMessage();
        }
    }
}
```

כפי שניתן לראות ה – MMULogger מספק מטודה אחת של כתיבה ומחזיק שני members:

- DEFAULT_FILE_NAME – קבוע שהוא מיקום ה – file אליו הוא יכתוב
- Handler – מצ"ב קישור ל – API של אוביקט זה (FileHandler), קראו והבינו כיצד להשתמש בו על מנת לכתוב (לממש את מטודת ה – write).

אוביקט נוסף שתעשו בו שימוש במסגרת ה – Logger הוא ה – LogRecord קראו והבינו גם את ה – API שלו. אנו נעשה שימוש בשני Levels בלבד (שיוזכרו בהמשך) על מנת ליצור אותו והם **INFO, SERVE.**

כל רכיבי המערכת (כל המחלקות) רשאיות לעשות שימוש ב – MMULogger ולגשת אליו אך **בשני מקרים בלבד**, זהו היכן מקרים אלו מתרחשים בקוד שלכם והשתמשו ב – Logger בהתאם, **המקרים הם**:

כאשר אירעה שגיאה בפעולת ה – MMU, כלומר בכל המקומות בהם טיפלתם עד עכשיו בשגיאות try/catch באמצעות כתיבה ל – System.out/err, עליכם לכתוב כעת ל – MMULogger.

במקרה זה ה – LogRecord יקבל את ה – Level – SERVE.

כאשר מתבצעת פעולה עדכון במערכת שהיא אחת מאלו:

המערכת מתחילה לפעול נרשמים גודל ה – RAM ומספר התהליכים שירוצו במערכת, PageFault, Page Replacment, getPages.

במקרה זה ה – LogRecord יקבל את ה – Level – INFO.

נתאר כעת מהו הפורמט (המבנה) שעליכם להיצמד אליו על מנת לכתוב כל אחת מהפקודות הללו בקובץ:

- RAM capacity - `RC 5`

פעולת ה – RAM capacity - מסומלת ע"י האותיות RC ובהמשך ה – capacity 5

- Processes Number – `PN 5`

פעולת ה – Processes Number - מסומלת ע"י האותיות PN ובהמשך כמות התהליכים 5

- Page Fault - `PF 2`

פעולת ה – Page Fault - מסומלת ע"י האותיות PF ובהמשך ה – ID של הדף 2

- `Page Replacement - PR MTH 2 MTR 4`

פעולת ה - Page Replacement - מסומלת ע"י האותיות PR ובהמשך MTH (move to HD) ומספר ה – ID של הדף 2 ו – MTR (move to RAM) ומספר ה – ID של הדף 4

- Get Pages - `P1 R 4 W [-47, -96, 42, -62, -106]`

פעולת ה – Get Pages - מתחילה במספר התהליך P1 מספר הדף שנקרא R 4 וה – data שנכתב לדף W [-47, -96, 42, -62, -106]

<u>להלן, דוגמה לקובץ Log תקני של המערכת:</u>

```
RC 5
PN 5

PF 7
PF 6
P2 R 7 W [-23, -55, 2, -52, -124]

P2 R 6 W [-45, 23, -112, -78, 101]

PF 2
PF 1
P0 R 2 W [98, 95, 41, -121, -22]

P0 R 1 W [-94, -45, 125, -69, -90]

PF 3
PR MTH 7  MTR 4
P1 R 3 W [54, -102, -100, 117, 108]

P1 R 4 W [-69, -94, -74, 113, 114]

P0 R 1 W [-91, 72, -5, -28, -114]

P0 R 2 W [44, 85, -113, -96, -2]

P1 R 4 W [-47, -96, 42, -62, -106]

P1 R 3 W [-116, 77, 65, -113, 119]

P0 R 1 W [103, -81, -58, -93, -27]

P0 R 2 W [-71, -49, 103, -110, -99]
```

עד עתה תארנו כיצד יש לתעד את פעולות המערכת ולבנות את קובץ ה – Log, כעת נתאר את עיצוב ובניית הממשק הגרפי וכיצד הוא אמור לפעול ע"פ קובץ זה.

העיצוב של החלק הגרפי במערכת שלנו יבנה ע"פ מודל ה – MVC שהוא ה – Design pattern בו משתמשים בכל המערכות המבוססות UI. למודל ה – MVC יתרונות רבים אך העקרון ה – OOP החשוב ביותר במודל זה הוא Loosely coupled .

עקרון זה ביסודו מניח כי כל רכיב במערכת או שכבה (שיכולה להיות מיוצגת ע"י מספר רכיבים) צריך להיות תלוי או מחובר כמה **שפחות** 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.

בנוסף כל שכבה מבודדת את הלוגיקה שלה ולכן גם את הבאגים שלה, באג בשכבה אחת צריך להיות מטופל בשכבה זו בלבד (מה שמסייע באיתור הבאגים 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 ישמר\ו המודל\ים 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.

בפרויקט שלנו אנו ניצור שלוש מחלקות שייצגו כל שכבה במודל ה - MVC.

המחלקה הראשונה תקרא `MMUModel` ותפקידה **לקרוא את קובץ** ה – Log שתואר למעלה (השתמשו במחלקות המובנות הרלוונטיות ב – JAVA שלמדנו, המאפשרות את הקריאה הפשוטה והיעילה ביותר), להכיל את הפקודות הרשומות בקובץ במבנה נתונים פנימי שלה וכמובן לדעת להחזיר אותו.

מחלקה זו גם אחראית לטפל בכל השגיאות (Exceptions) הרלוונטיות בקריאת הקובץ לדוגמה: קובץ לא קיים, פקודה לא קיימת וכדומה.

MMUModel API

```
package mvc;

import java.util.List;

public class MMUModel {
    private List<String> commands;

    public MMUModel(String filePath){
    }


    public List<String> getModel() {
        return commands;
    }

    private void readLog(){
        //Log reading and exceptions Handling...
    }

}
```

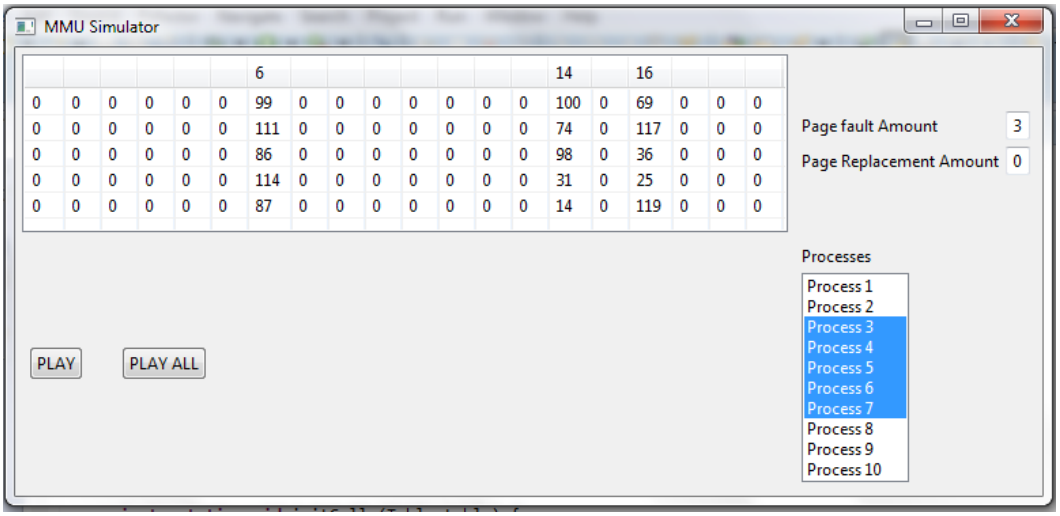
המחלקה השניה תקרא `MMUController` ותפקידה ליצור את ה – `Model` ליצור את ה – `View` ולחבר בניהם (**Straightforward...**) והיא תכיל מטודת `main` אחת.

המחלקה השלישית תקרא `MMUView` והיא זו שתבנה את הממשק הגרפי של המערכת. מחלקה זו מקבלת ב – CTOR שלה את **רשימת** הפקודות (שתוארו למעלה) שכוללות בין היתר גם את גודל ה – RAM וכמות התהליכים שרצו במערכת ה – MMU, בונה בהתאם לכך את ה – UI שיתואר להלן ומריצה את פקודות ה – Log ע"פ בקשת ה – USER שישתמש במערכת.

MMUView API

```
public class MMUView {


    private List<String> commands;

    public MMUView(List<String> commands){
        this.commands = commands;
    }

    public void open() {
        //Create mmu UI
    }
```

הממשק הגרפי יכלול לפחות את הרכיבים הבאים:



כפי שניתן לראות קיימים שלושה חלקים עקריים וכל חלק אחראי על שיקוף מידע אחר במערכת, נסביר את תפקידי הרכיבים ע"פ הדוגמה (בכיוון השעון):

החלק הראשון הוא הטבלה שצריכה לשקף את מצב ה – RAM. הפקודה שרלוונטית לשינוי התצוגה ברכיב זה היא ה- `getPages` בלבד. ע"פ הדוגמה המתוארת הדפים שנמצאים כעת ב – RAM הם 6,14,16 כאשר ה – `data` שבכל דף מוצג תחתיו (גודל המערך הוא 5).

החלק השני מימין לטבלה הוא ה – `Counters` ותפקידם לשקף את מספר ה - Page Replacement, Page Fault. הפקודות שרלוונטיות להם ברורות.

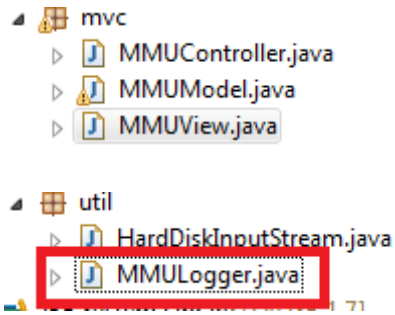
החלק השלישי הוא רשימה של כל התהליכים שרצו במערכת. רשימה זו מאפשרת למשתמש לבחור\לבודד את התהליכים שלהם **הוא רוצה** לשקף את הפקודות, בדוגמה שלנו התהליכים שנבחרו הם 3-7.
שימו לב!! שחלק זה מהווה מעיין פילטר לתהליכים שאינם נבחרו ולכן פקודות של תהליכים שלא נבחרו לא צריכות להיות משוקפות.

החלק הרביעי והאחרון הוא כפתורי ההפעלה, תפקידם בעצם לקרוא את הפקודות מרשימת הפקודות ולהפעיל את החלקים הראשון והשני בהתאם תחת האילוץ של החלק השלישי. הקריאה (החילוץ) של הפקודות מהרשימה יכול להתבצע בשתי צורות: האחת **שורה אחת** בכל לחיצה כאשר לוחצים על כפתור ה – **PLAY** והשניה את **כל השורות** בלחיצה על ה – **PLAYALL**.

דגשים לחלק זה:

- כיום כפי שהוסבר, ממשק המשתמש הוא חלק חשוב מאוד בכל מוצר תוכנה ועשוי לעיתים להכריע בבחירת הלקוח ולכן זהו החלק הראשון בו ניתן לקבל בונוס בפרויקט זה, כל תוספת בעיצוב הממשק שתעזור להבין את פעולות המערכת בצורה טובה יותר תזכה בנקודות (לדוגמה: אנימציות, צביעה משתנה של חלקים וכו')
- סידור הרכיבים ועיצוב ה – shell איננו חייב להיות ע"פ מה שהוצג אתם רשאים לעצב כרצונכם.
- חלק זה עשוי לסייע לכם בהבנה טובה יותר של פעולת ה – MMU ומהווה בסיס טוב להצגת המערכת גם בסרטון וגם בהגנה על הפרויקט לכן בנו לכם מספר תסריטים רלוונטיים.

ארזו את כל 4 המחלקות **החדשות** המתוארות ב – packages במבנה ובשמות הבאים:



חלק אחרון – תקשורת

המוצר שלכם נמכר במליוני דולרים ועשיתם את האקזיט המיוחל !! אך זוהי רק תחילת הדרך, שכן מחקרים מראים שתחזוקת מוצר תוכנה היא החלק העקרי, הארוך והיקר ביותר בכל חיי מוצר תוכנה.

בחלק הקודם נדרשתם להוסיף תיעוד למערכת ה – MMU (כתיבה לקבצי Log) ואף להוסיף ממשק משתמש שיודע לשקף את פעולת המערכת ע"פ קבצי ה – Log. יכולת זו, בין היתר, מסייעת להבין טוב יותר את פעולת המערכת ובמידת הצורך אף לאתר בעיות.

בחלק זה אנו נוסיף את היכולת לקבל\למשוך את קבצי ה – Log משרת מרוחק ולאחר מכן להפעיל את ממשק המשתמש שלנו ע"ג הקובץ שהתקבל.
פעולה זו תדמה מצב שללקוחות שנתקלים בבעיות עם המוצר שלנו יש מנגנון פשוט ונוח להעברת קבצי ה – Logשלהם ואנו יכולים להגיב בצורה מהירה ויעילה על מנת לאתר את התקלה ולטפל בהתאם.

ארכיטקטורת Client/Server היא אחת מתצורות ההתקשרות הנפוצות ביותר ברשתות מחשבים. תצורה זו מאוד פשוטה ומבוססת בעיקרה על רעיון שבו קיימים שני צדדים בארכיטקטורה ולכל צד תפקיד משלו:
צד אחד - הוא צד השרת ותפקידו לספק **שירותים** כלשהם עליהם הוא מצהיר שהוא יודע לספק.
צד שני – הוא צד לקוח ותפקידו **להפעיל** את אותם שירותים **לקבל את המידע** ולעשות איתו את מה **שהוגדר לו**.

המחלקה הראשונה אותה אנו נכתוב בחלק זה של הפרויקט תהווה את **צד השרת**. היא תחזיק את אוביקט מסוג `ServerSocket` **שיאזין ל – port 12345** (כפי שלמדנו בכיתה) ותקרא `MMULogFileServer`.
מחלקה זו היא יחידה נפרדת שצריכה לפעול בשרת מרוחק ולכן אנו ניצור אותה בפרויקט נפרד שיקרא **MMUServer** וב – Package שיקרא **mmunetworking**.
היא תכיל שתי מטודות מטודת main ומטודה סטטית אחת `checkPassword()`.
מטודת ה – main תכיל את **הפסאודו קוד** הבא:

```
Get passwordAndUsername from client
String fileName = checkPasswordAndUserName (passwordAndUsername)
If (fileName != null) {
  return through clientSocket OutputStream file(with name == fileName)
} else {
  return through clientSocket OutputStream "Wrong password"
}
```

הקובץ ששמו `fileName` יהיה ע"פ הקונבנציה הבאה – RemoteLog_XX.txt. קובץ זה צריך להיות נגיש למחלקה בדיסק המקומי שלה (באותה צורה שהשתמשנו בקבצים מקומיים בחלקים הקודמים) ועם פורמט זהה כמובן לפורמט קבצי ה – Log מהחלק הקודם.

הסיומת XX מייצגת את **מספר הקובץ** והיא שייכת ללקוח ספציפי **אחד** ז"א לא יוחזק קובץ זהה ללקוחות שונים, לדוגמה:

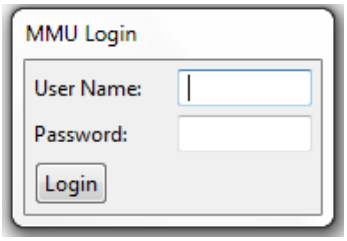
User name: Nissim Password: 1234 → RemoteLog_7.txt
User name: Dudu Password: 4567 → RemoteLog_18.txt

על מנת **למפות** משתמשים לקבצים תחזיק מחלקה זו **Map** שישמר\יקרא גם הוא **מקובץ מקומי** נוסף שיקרא users.txt **באותה דרך ששמרנו ל – HardDisk בחלק 2** (input/output streams). סיסמאות ושמות משתמשים בעולם האמיתי נשמרים ב – data bases חיצוניים אך מסיבות ידועות אנו מדמים זאת באמצעות קובץ. ה – Map יהיה במבנה הבא:

Key	Value
Nissim1234	RemoteLog_7.txt
Dudu4567	RemoteLog_18.txt
. . .	. . .

כפי שניתן לראות בפסאודו קוד תפקיד המטודה `checkPassword` לבצע את הבדיקה במבנה הנתונים של המשתמשים\סיסמאות ולהחזיר את שם הקובץ הרלוונטי במידה ונמצא אחרת `null` .

<u>צד הלקוח</u> לא יהיה מחלקה נוספת אלא תוספת למחלקה `MMUController` שכתבנו בחלק הקודם.
ה - `MMUController` כזכור, יוצר את ה – `MMUModel` ע"י הקובץ **המקומי** Log.txt שנוצר מפעולת ה – MMU. התוספת ל – `MMUController` תיתן את האפשרות להתחבר לשרת חיצוני (`MMULogFileServer`)
להוריד ממנו את קובץ ה - Log.txt (שיקרא RemoteLog.txt) ולשקף **באותה צורה** באמצעות ה – `MMUView` את פעולות המערכת.
על מנת להתחבר לשרת המרוחק (`MMULogFileServer`) נדרשים שם משתמש וסיסמה, אותם יזין המשתמש דרך ממשק נוסף **ופשוט** שיקרא `MMULoginDialog` והוא יראה כך(אתם רשאים לעצב אחרת):



ה – API של `MMULoginDialog` צריך להיות פשוט ולחשוף ל – `MMUController` (לאחר היצירה) שתי מטודות: מטודת open() – פותחת את ה – Dialog ומציגה את הממשק

מטודת getUserNameAndPassword() – מחזירה String לאחר שהמשתמש לוחץ על כפתור ה – Login שהוא שרשור של הסיסמה ושם המשתמש.

לאחר שה - `MMUController` מפעיל את שתי המטודות וקיימים לו שני הנתונים הוא ניגש לשרת ה - `MMULogFileServer` ומנסה לקבל את הקובץ. לבסוף צריך קוד ה – `MMUController` להראות כך (גם כאן אתם רשאים לממש אחרת, **כמובן ללא פגיעה בפונקציונליות**):

```
public class MMUController {

    private static final boolean _REMOTE = true; // Or false

    private static final String LOCAL_PATH = "log.txt";
    private static final String REMOTE_PATH = "RemoteLog.txt";

    public static void main(String[] args) throws FileNotFoundException, IOException {

        String fileName = LOCAL_PATH;//Set local path - default

        if(_REMOTE){
            MMULoginDialog dialog = new MMULoginDialog(new Shell());
            dialog.open();
            String unAndPass = dialog.getUserNameAndPassword();
            fileName = REMOTE_PATH;

            /**
             * go to server with unAndPass ...
             */

        }
        MMUModel model = new MMUModel(fileName);
        MMUView view = new MMUView(model.getModel());
        view.open();
    }
}
```

מספר דגשים וטיפים לחלק זה:

- חלק זה עוסק בתקשורת לכן התמקדו במחלקות הרלוונטיות, מסך ה – Login וקבלת הנתונים יכול להדחות לסוף.
- שימו לב לסגור input/output streams בשני הצדדים client/server ניתן להשתמש (> גרסה 7) ב – try

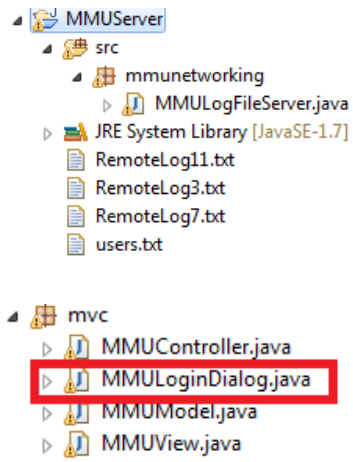
```
try (Socket socket = new Socket(hostName, portNumber);
      PrintWriter out = new PrintWriter(socket.getOutputStream(), true)
```

זה כבר יסגור את שני המקורות המופיעים בסוגריים { … }

- צרו יותר מלקוח וקובץ אחד ששייך לו על מנת להראות את תקינות המערכת.
- **אינכם מחוייבים לטפל ביותר מבקשה אחת** (למרות שזהו מצב לא סביר בכלל במערכת רגילה) לכן תדאגו שבקשה אחת תעבוד תקין. מעבר לטיפול ביותר מבקשה אחת הוא פשוט.

- **שימו לב,** אילו קבצים שמורים בצד לקוח ואילו בצד שרת (תוכלו להבין זאת גם ממבנה הקבצים, למטה)

ארזו את המחלקות **החדשות** המתוארות בפרויקטים וב – packages במבנה ובשמות הבאים:



נספח אופן הגשת מטלות:

יש לארוז את הפרויקט לקובץ zip. מתוך האקליפס כולל כל הקבצים.

לחיצה ימנית על שם הפרויקט ← export ← general ← archive file ← תנו את השם MemoryManagmentProject לפרויקט

ותוודאו שמסומן Create directory structure for files