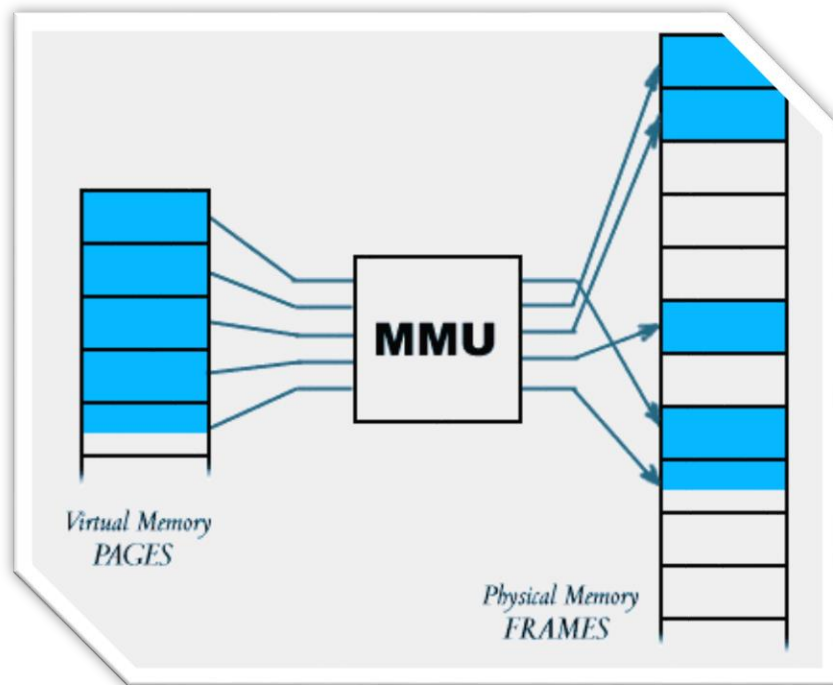


קורס פיתוח אלגוריתמי JAVA

פרויקט יחידת ניהול זכרון – MMU



HIT – מכון טכנולוגי חולון

מרצה: ניסים ברמי

סמסטר ב 2017

תוכן עניינים

3.....	פרויקט – יחידת ניהול הזיכרון (Memory management unit)
4.....	יחידת ניהול זיכרון (MMU) - הקדמה
6.....	חלק א' – אלגוריתמי Cache
10.....	חלק ב' – עקרונות OOP ו Design Patterns
15.....	חלק ג' – Multi-Threading

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 את מטלה 3 במועד שיקבע על מנת לקבל פידבק, מלבד המטלה הסופית (תרגיל 5).

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 שאת 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.

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 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.

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 חומרית ומכיוון שאנו בקורס שלנו מתעסקים בתוכנה בלבד, אנו נאלץ לדמה את המרכיבים החומריים בתוכנה.

חלק א' – אלגוריתמי Cache

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 מדויקת יותר אילו דפים יש להעביר מהזיכרון המשני לעקרי.

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 ובהם הוא משתמש, מרחב הכתובות הללו מוגדר **ככתובות וירטואליות** וסך כל הכתובות הוא **הזיכרון וירטואלי**. הזיכרון הוירטואלי מנוהל ע"י ה-CPU ומסתיר 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 בדרך"כ למספר תוכניות שונות) צריך לדאוג לנהל זאת בצורה יעילה ונכונה **ולמזער כמה שיותר את פעולות הדפדוף**.

יחידת ה-MMU צריכה להשתמש באלגוריתמים שייעודים לפתרון בעית ניהול המשאבים על מנת לבצע דפדוף, לכן היא צריכה להחזיק כ-reference ממשק לאלגוריתמי ההחלפה (אלגוריתמים אלה נקראים [Cache Algorithm](#)).

ממשק זה יקרא בפרויקט שלנו `IAlgoCache`, 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 Classes). בנוסף אנו נרצה לאפשר גמישות מוחלטת בשינוי מימוש האלגוריתם ולהוסיף מימושים נוספים ל - `IAlgoCache` 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.

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 תבחרו בהתאם לאלגוריתם (מה - containers המובנים של JAVA שלמדנו) שצריך להיות זהה מבחינת הגודל שלו ל - RAM ומבחינת התוכן שלו (הם לא 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 (חישוב למה לאחר שתכירו את האלגוריתמים).

שימו לב!! אלגוריתמים אלה מייעצים ל - MMU אותו נממש בהמשך ולא ל - RAM

ממשק ה - `IAlgoCache` וההסברים לגבי המטודות\תכונות **מצורפים כמו שאר המחלקות\ממשקים בתקייט ה - API's** וכפי שניתן לראות הוא מאפשר גמישות מירבית מבחינת ה - $\langle K, V \rangle$ Types איתם הוא יעבוד, אתם צריכים לשמור על גמישות זאת גם במימוש שלכם, ז"א החתימות של המטודות באלגוריתמים שלכם צריכות להישאר זהות ל - `IAlgoCache` ולקבל ולהחזיר טיפוסים גנריים **בדיוק** כפי שמחזיר ה - `IAlgoCache`, מי שירצה בהמשך להשתמש באלגוריתמים שלכם הוא זה שיקבע את הטיפוס איתו הוא רוצה לעבוד.

מחלקה נוספת (abstract class) שאתם צריכים לממש נקראת `AbstractAlgoCache<K, V>` והיא שכבה נוספת של הפשטה שמאפשרת שמירה של נתונים משותפים שקיימים בכל האלגוריתמים, למשל ה - `capacity`.

אלגוריתמי ההחלפה אותם אתם צריכים לממש בפרויקט שלנו הם:

NFU , LRU, Random - https://en.wikipedia.org/wiki/Page_replacement_algorithm

בלינק שצורף תוכלו לקבל הסברים 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 בדור"כ ה – HardDisk ועליו נדבר בהמשך), ה – API של היחידות הללו נמצאים בתיקיית ה – API, בנוסף אתם יכולים כמובן להוסיף מטודות private שלכם ע"פ הצורך.

שימו לב!! שני רכיבים אלא כולל המחלקות של אלגוריתמי ההחלפה הינם רק חלקים מה – MMU אותו נממש בהמשך.












המחלקה האחרונה אותה אתם צריכים לכתוב בחלק זה היא ה – IAlgoCacheTest שכל תפקידה הוא לבדוק את שלושת האלגוריתמים שמימשתם ופעולתם. מחלקה זו תשתמש ב- [JUnit framework](#) ותכיל 3 מטודות @test 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 את כל 6 המחלקות המתוארות ב – packages במבנה (שימו לב היטב למבנה התיקיות)

ובשמות הבאים (שימו לב לשמות האלגוריתמים):

- ▲  CacheAlgorithm
 - ▲  src/main/java
 - ▲  com.hit.algorithm
 - ▷  AbstractAlgoCache.java
 - ▷  IAlgoCache.java
 - ▷  LFUAlgoCacheImpl.java
 - ▷  LRUAlgoCacheImpl.java
 - ▷  SecondChanceAlgoCacheImpl.java
 - ▲  src/main/test
 - ▲  com.hit.algorithm
 - ▷  IAlgoCacheTest.java

חלק ב' – עקרונות OOP ו - Design Patterns

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

1. [Create jar file from eclipse](#)

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

2. [Adding Internal JARs \(Method 1\)](#)



עכשיו הגיע הזמן להתחיל ולפתח את מערכת ה – MMU שלנו

הוסיפו שתי מחלקות לפרויקט ה – MMU שלנו RAM, Page ע"פ ה – API המצורף בתיקיית ה – doc2:

RAM.html, Page.html

שימו לב, מחלקות אלו אינן דורשות לוגיקה לחלוטין אלא הן קונטיינרים של data, עשו שימוש במבני הנתונים הנכונים על מנת לממש אותן.

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

מחלקה נוספת שנמש בחלק זה של הפרויקט היא ה - `MemoryManagementUnit` (MMU). ה - MMU עצמו אשר יכול בתוכו (members - שני רכיבים **בלבד**: RAM, `AlgoCache`). 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 מצורף בתיקיית ה – API בקובץ:

MemoryManagementUnit.html

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:

```
pages [] getPages(pageIds[])

do If IAlgoCache  $\not\subset$  pageIds // touch the page

    If RAM is not full

        Retrieves page from HD// Page Faults

    Else

        Do logic of full RAM // page replacements

end;

return pages [] // from RAM

end func
```

כפי שניתן לראות ב - Pseudo Code עושה ה – MMU שימוש ב – IAlgoCache. ה – IAlgoCache מועבר כפרמטר
ב – CTOR ל – MMU ונשמר אצלו כ - reference – member (MMU API).

השימוש של ה – MMU ב- IAlgoCache 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, להוסיף בכל שלב אחרים ובאותה צורה
 להעביר גם אותם (open for extension). בכך השלמנו את ה – strategy pattern ושמרנו על עקרון ה – principal
 .open/close

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 (כ – members) שני constants עיקריים:

_SIZE - גודל ה – HD במספר דפים.

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

מעבר לכך ה – HD יאפשר את ה – API שמופיע בתיקית ה – API בקובץ:

HardDisk.html

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 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 ה – MMU (הפעם באמצעות [JUnit](#) כפי שלמדנו בכיתה) שם המחלקה תהיה – `MemoryManagementUnitTest` ותפקידה הוא לבדוק את תקינות פעולת המערכת.

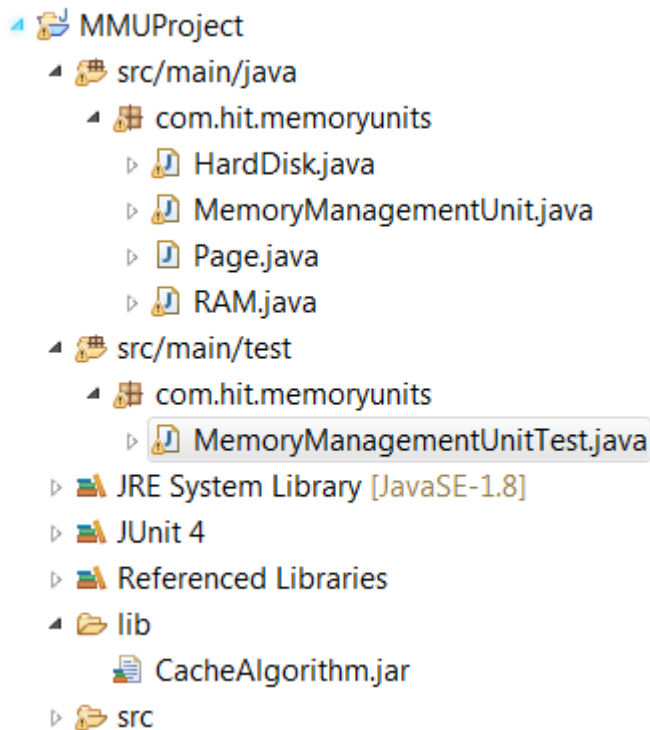
הטסטר צריך לבנות את כל האובייקטים הרלוונטיים (`IAlgoCache`, `MemoryManagementUnit`) השאר תיצור המערכת (בעצמה), ותחבר אותם על מנת שתוכלו להפעיל את המערכת.

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 5 המחלקות החדשות המתוארות ב – packages במבנה ובשמות הבאים
ובנוסף שימו לב ל include של ה – CacheAlgorithm.jar:



חלק ג' – Multi-Threading

מזל טוב !! בשלב זה של הפרויקט יש לנו מערכת MMU שעובדת ויודעת לנהל זיכרון וירטואלי כך שפעולות הדפדוף הן יעילות ושקופות למשתמש, אך כעת אנו רוצים גם לדמות את המערכת עובדת כפי שהיא במקומה הטבעי (חלק חומרת אליו ניגשת מערכת ההפעלה).

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

לא כך בעולם האמיתי, יחידת ניהול הזכרון פועלת אל מול מערכת ההפעלה ומהותה של מערכת ההפעלה להיות multi-tasking כלומר לדעת לנהל מספר תהליכים בו זמנית, לכן תפקידנו לדמות צורת פעולה זו ולהעריך 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 תת 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. אנו נממש את ה - Task שלנו הפעם ע"י **implements Callable**.

התהליך שלנו שהוא המחלקה הראשונה אותה נכתוב בחלק זה והיא תקרא **Process** (נמצאת בתיקיית ה - API בשם זה). מחלקה זו תבצע שתי פעולות בלבד אך **במחזורים** (ProcessCycles) ואותם נסביר בהמשך, הפעולות הן:

1. **לבקש דפים מה - MMU וכאשר הוא מקבל אותם לכתוב/לקרוא data שמתקבל ע"י ה - ProcessCycle**
2. **לישון (sleep) מספר שניות שהוגדרו לו ע"י ה - ProcessCycles**

שתי הפעולות הללו הן עיקר הפעולה של ה - Callable ולכן יתבצעו במטודת הפעולה של ה - Callable שהיא ה - **call()**.

המחלקה process מחזיקה שלושה members (ניתן לראות גם ב-API):

MemoryManagementUnit mmu – מוחזק כ- reference למערכת ה-MMU שלנו על מנת שיהיה ניתן לפנות אליו ולבקש את הדפים הרלוונטיים לו בכל מחזור.

int ProcessId – מזהה חח"ע של ה-process

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

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

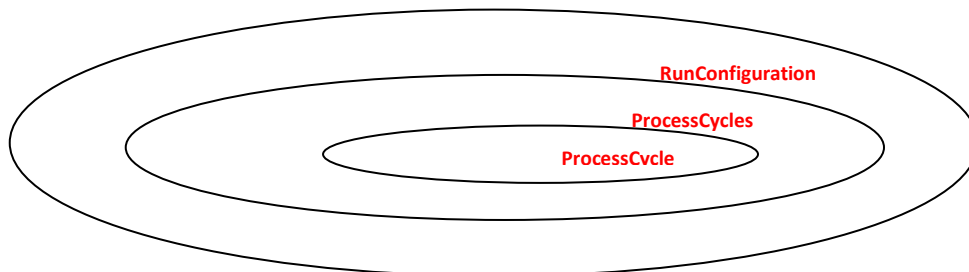
נתאר מהי מחזוריות של תהליך - כל תהליך כפי שנאמר יודע לבצע שתי פעולות בלבד לבקש דפים (לכתוב או לקרוא אליהם) ולישון מספר שניות.

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 את הזמן אותו הקצתה לו מערכת ההפעלה ([Time slice](#)), **מתזמנת** מערכת ההפעלה (ה-scheduler) תהליך נוסף (תהליך ב') מתוך תור התהליכים שמחכים, בזמן זה יכול תהליך א' לעבור למספר מצבים ("I/O bound", waiting etc..) ואנו נגדיר זאת כמצב שינה למספר שניות, **בקשת הדפים וקבלתם + שלב השינה מוגדר כמחזור אחד**.

בשלב מסויים תהליך א' מסיים את מצב השינה שלו ורוצה לבצע שוב קריאה וכתובה 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). ניתן למצוא את המחלקה בתיקיית ה-API תחת השם ProcessCycle.

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



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 לאובייקטים 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 אובייקטים אלו ללא צורך בכתיבה של מחלקה כלשהי ב – JAVA ואפילו ללא ידיעה כיצד אתם ממשותם את המחלקות שלכם.

הצורה שאנו נבצע את העברת המידע שלנו הוא ע"י הסכמה על מבנה האובייקטים 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 (MAP).

1. בכדי לייצג את האובייקט ProcessCycle נשתמש במבנה הבא:

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

שימו לב כי שמות ה – keys זהים לתכונות האובייקט 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 (שהוא רשימה של ProcessCycle) נשתמש במבנה הבא:

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

3. כדי לייצג את האובייקט RunConfiguration (שהוא רשימה של ProcessCycles) נשתמש במבנה הבא:

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

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 שתורמת רבות לעולם ה – open source והיא [Google](#)).

בכדי להשתמש בספריות אלו אנו נוריד תחילה את קובץ ה – JAR המכיל מחלקות אלו (רצוי את הגרסה [האחרונה](#)) 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 אובייקטים לקבצי JSON ובחזרה לאובייקטים כמובן במידה ותוכן ה – JSON תקין ומותאם לאובייקט הרלוונטי.

לדוגמה האובייקטים 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);
```

מחלקה נוספת אותה תצטרכו לכתוב בחלק זה תקרא CLI:

CLI - A command-line interface or command language interpreter (CLI), also known as command-line user interface is a means of interacting with a computer program where the user (or client) issues commands to the program in the form of successive lines of text (command lines).

מחלקה זו תהיה אחראית לממשק מול הלקוח על מנת להגדיר את התנהגות המערכת ובמקרה שלנו עם איזה אלגוריתם ירוץ ה - MMU ומה גודל ה - Capacity של ה - RAM.

ה - CLI יתמוך כעת רק בשתי פקודות בסיסיות:

<START/STOP> - פעולה זו תתחיל/תפסיק את פעולת המערכת.

<LRU/LFU/SS CAPACITY> - האלגו' איתו תרוץ מערכת ה - MMU + גודל ה - cache.

פעולת ה - CLI תתבצע כ - Thread נפרד (לכן עליה לממש את הממשק Runnable) ובצורה פשוטה: בכל פעם שיהיה לה פקודה מוכנה היא תפעיל את פקודת ה - start() במחלקה MMUDriver אותה נתאר בהמשך.

שימו לב בכל שלב אתם צריכים לטפל רק בפקודות שמוכרות ל - CLI כל פקודה אחרת יש להודיע ללקוח שהפקודה אינה תקינה ולבקש שוב קלט תקין.

דוגמאות לתקשורת CLI מול הלקוח:

No 1

```
start
Please enter required algorithm and RAM capacity
LRU 5
start
Please enter required algorithm and RAM capacity
LFU 5
stop
Thank you
```

No 2

```
start
Please enter required algorithm and RAM capacity
fff
Not a valid command
```

No 3

```
stop
Thank you
```

המחלקה האחרונה שאותה נכתוב בחלק זה של הפרויקט תקרא

ה – **MMUDriver**:



מחלקה זו היא למעשה החלק שמחבר לנו את כל רכיבי המערכת וגם מפעיל אותם, היא תשמש גם בין היתר כסטור 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 ע"י יצירת ה - CLI וקבלת הגדרת הלקוח ותדאג לעביר אותו כ - reference לכל אחד מה - processes כדי שיוכלו לעשות בו שימוש ולבקש את הדפים בהם יש להם צורך.

מתוארים להלן חלקים מהקוד של ה – MMUDriver (העתיקו והדביקו), שימו לב לבניית ה – CLI והפעלתו

ולמטודת ה Start:

```

public static void main(String[] args){
    CLI cli = new CLI(System.in, System.out);
    new Thread(cli).start();
}

public static void start(String[] command){

    IAlgoCache<Long, Long> algo = null;
    int capacity = 0;
    /**
     * Initialize capacity and algo
     */

    /**
     * Build MMU and initial RAM (content)
     */
    MemoryManagementUnit mmu = new MemoryManagementUnit(capacity, algo);

    RunConfiguration runConfig = readConfigurationFile();

    List<ProcessCycles> processCycles = runConfig.getProcessesCycles();
    List<Process> processes = createProcesses(processCycles, mmu);
    runProcesses(processes);
}

```

שימו לב!! מטודת ה - start היא סטטית מה שמחייב את שאר המטודות להן היא קוראת להיות סטטיות גם כן.

הסבר, לפעולת ה – MMUDriver והקוד שתואר, ע"פ שורות הקוד:

1. יצירת ה – CLI ואתחול מקורות ה – input/output שלו, והפעלתו כ – Thread נפרד.
2. יצירת ה – MMU ואתחולו (גודל ה – RAM או capacity) באמצעות קריאת ה - command שהתקבל מה – CLI
3. קריאת קובץ ה – JSON (יסופק בנפרד) ויצירת האובייקט – RunConfiguration כפי שהוסבר למעלה

4. יצירת התהליכים וביצוע החיבור בינם לבין ה – MMU. שימו לב שמספר התהליכים שיווצרו הוא כמספר ProcessCycles (חשבו מדוע...)
5. 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 האובייקט איתו עובדים התהליכים שהוא ה – MMU** ישנם מספר דרכים, כפי שלמדנו.

בונוס ראשון :

- בסיום פעולת ה – MMU (סיום העיבוד של כל ה – Processes) קיימים דפים ב – RAM שטרם נשמרו ל – HD דאגו לשמור אותם.

רמז: הוסיפו מטודה shutdown ל – MMU ודאגו לקרוא לה מה – MMUDriver

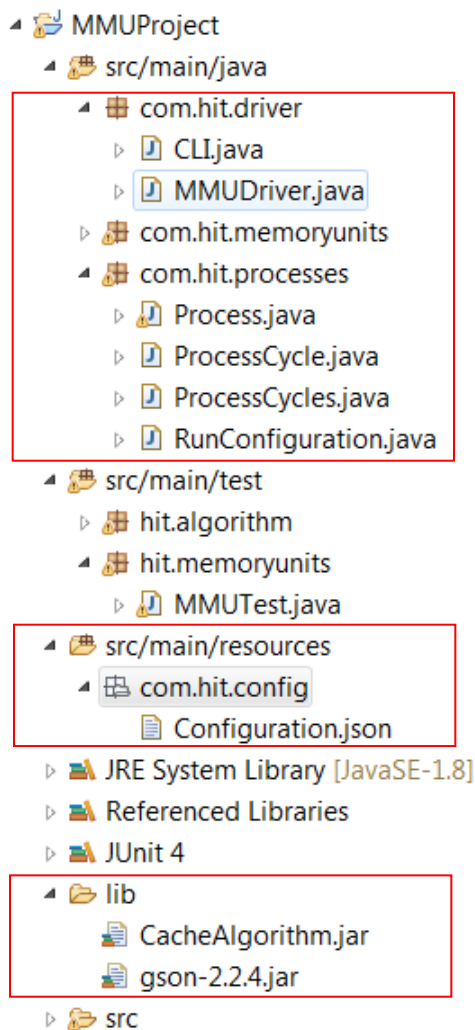
2 נקודות

- בצעו אופטימיזציה למערכת (ברכיב ה – MMU) בנושא של קריאת\כתיבת הדפים הדרכה: שנו את חתימת המטודה – getPages לחתימה הבאה:

```
getPages(Long[] pageIds, boolean[] writePges)
```

כעת תוכלו להבחין בין דפים לכתיבה ודפים לקריאה, חשבו כיצד ניתן לחסוך זמן בעיבוד דפים שבהם מבצעים קריאה בלבד. **2 נקודות**

לבסוף ארזו את כל 6 המחלקות החדשות המתוארות ב – packages במבנה ובשמות הבאים בנוסף הוסיפו מבנה תיקיות חדש ע"פ המתואר **src/main/resources** קובץ ה – Configuration.json מופיע לכם בתיקיה של התרגיל לבסוף את קובץ ה – gson.jar (גירסה חדשה) שהורדתם צרפו לפרויקט לתיקיית ה - lib :



רקע (ויקיפדיה) - את המונח "חויית 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 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 שתורם 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 לשקף את הפעולה של המערכת בצורה ויזואלית והספריות בהן אנו נשתמש לצורך כך הן ספריות ה – Swing (אופציונלי SWT) שלמדנו בכיתה. לפני שנוכל לשקף את פעולות המערכת ולבנות ממשק מתאים, יש צורך ראשית לתעד את פעולות המערכת בצורה מובנית וע"פ סדר התרחשותן וזאת נעשה בצורה סטנדרטית כפי שקיים בכל מערכות ה – Production באמצעות Logger שאותו נכתוב הפעם בעצמנו.

ראשית נסביר מהו Logger, מה תפקידו וכיצד הוא יסייע לנו בחלק זה של הפרויקט. ה – Logger הוא רכיב תוכנה (מחלקה) שתפקידו לספק API שמאפשר כתיבה ותיעוד של הודעות (messages) בזמן ריצת המערכת ל – resource חיצוני למשל File (בשלב זה אתם כמובן יודעים כיצד קורא תהליך הכתיבה ובאמתעות אלו מחלקות). ה – Logger הוא רכיב חשוב וקריטי למערכות מורכבות ולאו דווקא, 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 מכתובת 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) ותעשה בהם שימוש.

המופע (instance) של ה – Logger צריך להיות אחד והפניה אליו זהה וגלובאלית מכל רכיבי המערכת דאגו לכך (אתם כבר מכירים את ה – pattern).

להלן חלק מהקוד של המחלקה MMULogger השלימו את החלקים כפי שמתואר בהמשך

MyLogger

```
*MMULogger.java
1 package hit.util;
2
3 import java.util.logging.FileHandler;
4
5
6
7
8 public class MMULogger {
9
10     public final static String DEFAULT_FILE_NAME = "log.txt";
11     private FileHandler handler;
12
13     private MMULogger(){
14         // Complete the rest
15     }
16
17     public synchronized void write(String command, Level level){
18         // Complete the rest
19     }
20
21     public class OnlyMessageFormatter extends Formatter
22     {
23         public OnlyMessageFormatter() { super(); }
24
25         @Override
26         public String format(final LogRecord record)
27         {
28             return record.getMessage();
29         }
30     }
31 }
```

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 אובייקט זה ([FileHandler](#)), קראו והבינו כיצד להשתמש בו על מנת לכתוב את מטודת ה – write.

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

כל רכיבי המערכת (כל המחלקות) רשאיות לעשות שימוש ב – MMULogger ולגשת אליו אך **בשני מקרים בלבד:**

1. 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.

2. כאשר מתבצעת פעולה עדכון במערכת שהיא אחת מאלו:

- המערכת מתחילה לפעול נרשם גודל ה – RAM (מספר הדפים)
- 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 – 5 capacity

➤ Process Numbers – PN:70

פעולת ה – Process Numbers - מסומלת ע"י האותיות PN ובהמשך מספר התהליכים 70

➤ 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 - GP:P1 4 [-47, -96, 42, -62, -106]

פעולת ה - Get Pages - מסומלת ע"י האותיות GP מתחילה במספר התהליך P1 מספר הדף שנקרא 4

וה - data שנכתב לדף [-47, -96, 42, -62, -106] במקרה של קריאה המערך יהיה ריק []

להלן, דוגמה לחלק מקובץ Log תקני של המערכת:

```
RC 5
PN:70

PF 1
GP:P1 1 [50, -6, -16, 90, -45]

PF 213
PF 216
GP:P70 213 [-21, -18, -31, 70, 53]
GP:P70 216 [52, -13, 38, -123, 89]

PF 210
PF 209
GP:P69 210 [-21, -43, -54, 47, 20]
GP:P69 209 [-29, -38, -77, -6, -104]

PR MTH 1 MTR 205
PR MTH 213 MTR 206
GP:P68 205 [63, 41, -113, 121, -59]
GP:P68 206 [-52, 60, -67, -52, 111]

PR MTH 216 MTR 203
GP:P67 203 [-23, 90, 121, 6, -45]

PR MTH 210 MTR 202
GP:P66 202 [53, -25, 8, -105, 2]

PR MTH 209 MTR 199
PR MTH 205 MTR 200
GP:P65 199 [36, 19, 72, 104, 37]
GP:P65 200 [87, 40, -5, -8, -116]
```

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 בהתאם לקובץ 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 לא 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 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.

אנו נייצג כל שכבה באמצעות interface ומחלקה אחת לפחות (**כאן אתם ראשיים להוסיף מחלקות** נוספות):

❖ **השכבה הראשונה ה-Model** תכיל לפחות את המחלקה `MMUModel` וה-`Model` interface.

תפקידה של מחלקה זו הוא ליצור ולהפעיל את מחלקת ה-MemoryManagementUnit, כפי שנעשה בתרגיל הקודם

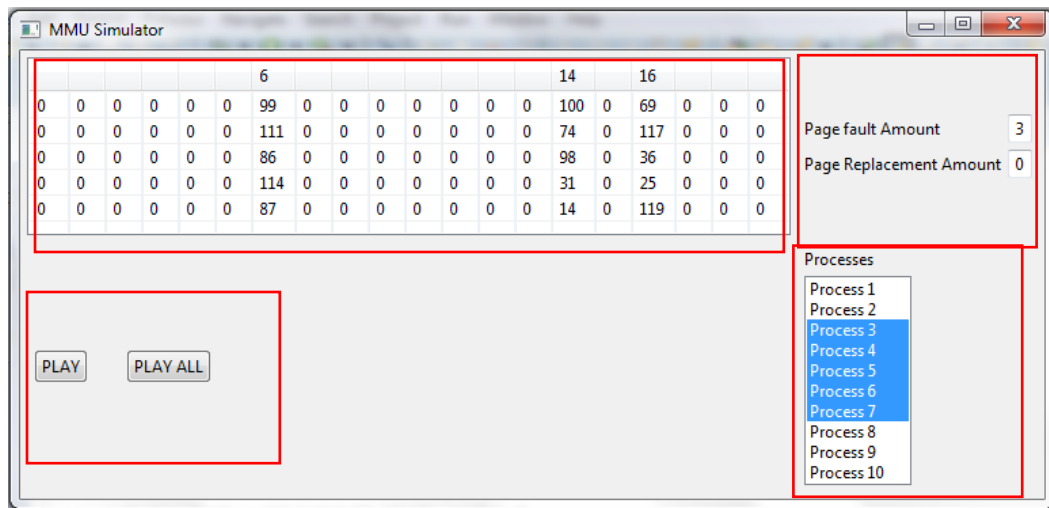
באמצעות `MMUDriver`, לכן עליכם לבצע כעת [Code refactoring](#) למערכת שלכם ולהעביר את הלוגיקה של הפעלת המערכת לשכבת ה-Model מהמחלקה `MMUDriver`, כעת בכדי שהמערכת שלכם תפעל ה-CLI יצטרך להיות חלק ממודל ה-MVC שלכם (חלק משכבת ה-View) ולהעביר את המידע דרך ל-Model דרך ה-Controller.

בנוסף ה - `MMUModel` תהיה אחראית לקרוא את קובץ ה - `log` שנוצר בתום תהליך הכתיבה ל - `log` ולדווח על היווצרותו ל - `controller` ע"פ ה - `observer pattern` שלמדנו בכיתה.
 לטובת קריאת ה - `Log` שתואר למעלה (השתמשו במחלקות המובנות הרלוונטיות ב - `JAVA` שלמדנו, המאפשרות את הקריאה הפשוטה והיעילה ביותר), הכילו את הפקודות הרשומות בקובץ במבנה נתונים פנימי של המחלקה.
 מצ"ב בתיקיית ה - `API`, ה - `API` של ה - `Model`, אתם רשאים להוסיף מטודות ע"פ בחירתכם.

❖ **השכבה השניה** ה - `Controller` תיוצג ע"י ה - `interface Controller` והמחלקה `MMUController` שתפקידה לקבל ב - `constructor` את ה - `Model` וה - `View` ולבצע את החיבור בניהם.

❖ **השכבה השלישית** ה - `View` תיוצג ע"י ה - `interface View` והמחלקה `MMUView` (החלק העקרי בתרגיל זה) והיא זו שתבנה את הממשק הגרפי של המערכת. שכבה זו מקבלת מה - `MMUController` את רשימת הפקודות (שתוארו למעלה), גודל ה - `RAM` וכמות התהליכים שרצו במערכת.
 ה - `MMUView`, בונה בהתאם לכך את ה - `UI` שיתואר להלן ומריצה את פקודות ה - `Log` ע"פ בקשת ה - `USER` שישתמש במערכת.

הממשק הגרפי יכול ללכוד את הרכיבים הבאים:



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

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).

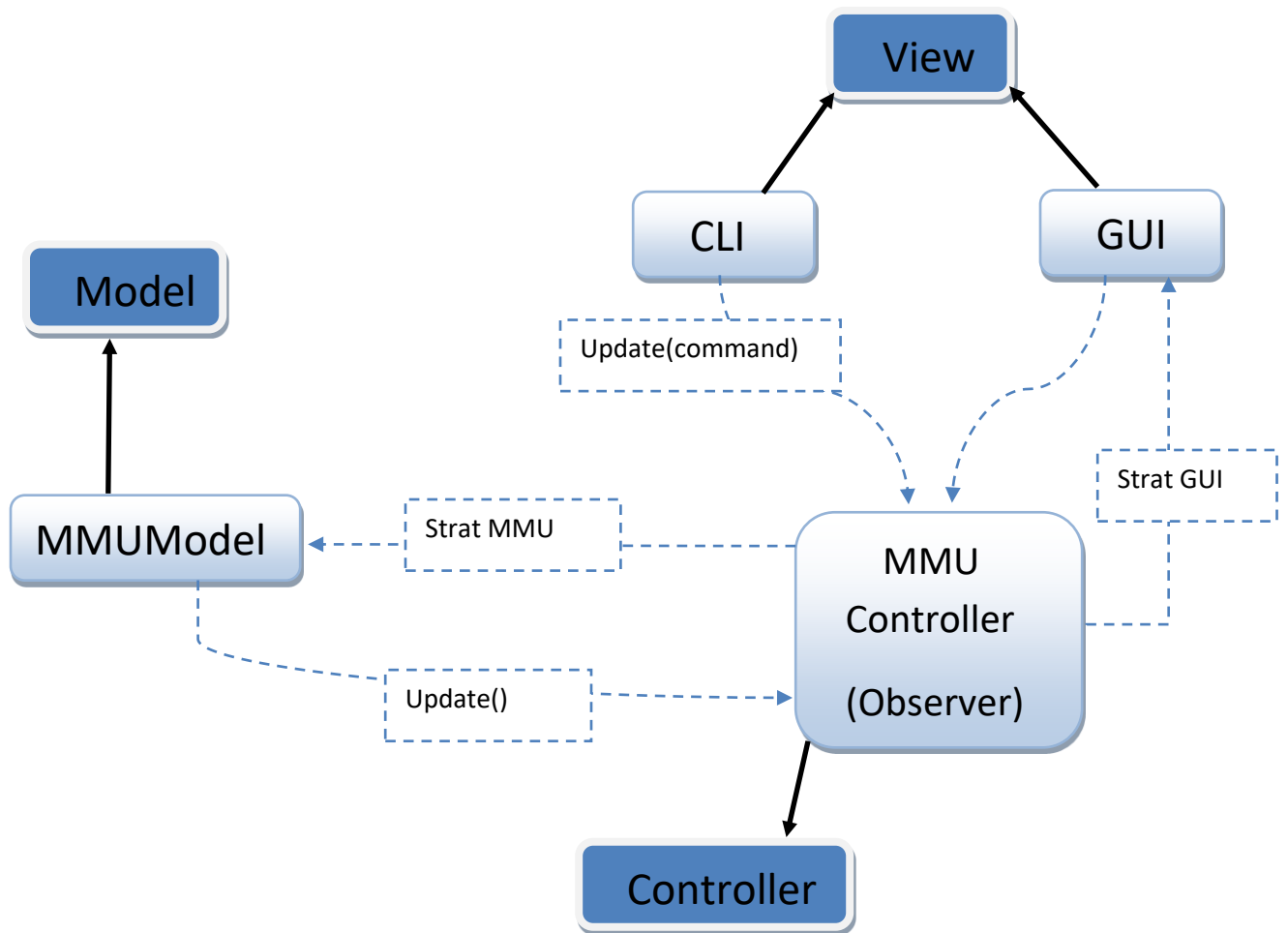
החלק השני מימין לטבלה הוא ה – `Counters` ותפקידם לשקף את מספר ה – `Page Page Fault` Replacement, הפקודות שרלוונטיות להם ברורות.

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 מעין 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**.

מצ"ב בתיקיית ה – API, ה – API של ה – **View** כפי שניתן לראות ה – `interface` מכיל מטודה אחת כל שאר המטודות יתווספו ל – `MMUView` ע"פ ה – **Design שלכם !!!**

מתואר להלן ה - Design + Flow control של המערכת:



מקרא:

ירשה -

הפעלת פקודה -

interface -

Concrete class -

יצירת ה - mvc תתבצע מה - MMUDriver בצורה הבאה:

```

public static void main(String[] args){
    /**
     * Build MVC model to demonstrate MMU system actions
     */
    CLI cli = new CLI(System.in, System.out);
    MMUModel model = new MMUModel(configuration);
    MMUView view = new MMUView();
    MMUController controller = new MMUController(model, view);
    model.addObserver(controller);
    cli.addObserver(controller);
    view.addObserver(controller);
    new Thread(cli).start();
}

```

**בונוס שני** (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 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.

ארוזו את כל המחלקות החדשות (לפחות 7) המתוארות ב - packages במבנה ובשמות הבאים:

- ▲ 📁 MMUProject
 - ▲ 📁 src/main/java
 - ▲ 📁 com.hit.controller
 - ▶ 📄 Controller.java
 - ▲ 📁 MMUController.java
 - ▶ 📄 MMUController
 - ▶ 📁 com.hit.driver
 - ▶ 📁 com.hit.memoryunits
 - ▲ 📁 com.hit.model
 - ▶ 📄 MMUModel.java
 - ▶ 📄 Model.java
 - ▶ 📁 com.hit.processes
 - ▲ 📁 com.hit.util
 - ▶ 📄 MMULogger.java
 - ▲ 📁 com.hit.view
 - ▶ 📄 MMUView.java
 - ▶ 📄 View.java
 - ▶ 📁 src/main/test
 - ▲ 📁 src/main/resources
 - ▲ 📁 com.hit.config
 - 📄 Configuration.json
 - ▶ 📁 JRE System Library [JavaSE-1.8]
 - ▶ 📁 Referenced Libraries
 - ▶ 📁 JUnit 4
 - ▶ 📁 doc
 - ▶ 📁 lib
 - ▲ 📁 logs
 - 📄 log.txt
 - ▶ 📁 src