מסמך דרישות – פרויקט מסכם בקורס תכנות בסביבת האינטרנט

סטודנטים יקרים,

- מסמך זה מכיל את הדרישות לפרויקט המסכם.
- כל קבצי המקור וחומרי העזר זמינים לרשותכם ב-moodle.
- סטודנטים שצפו בשיעורים ויעשו חזרה על הקוד שכתבנו יכולים לסיים את הפרויקט תוך פרק זמן קצר.
- באפשרותכם לבחור באחד משני מועדי ההגנה הבאים
  1. מועד 1 – 17.8 בשעה 16:00
  2. מועד 2 – 19.8 בשעה 16:00
- המסמך כולל שני חלקים- החלק הראשון עוסק בביצוע התאמות לקוד מוכר מהשיעורים.
  החלק השני עוסק במימוש של משימות אלגוריתמיות, גם הן על-בסיס פרויקט שכתבנו יחד הסמסטר.
- שני החלקים במסמך זה הם חובה. בנוסף, אפרסם בנפרד אתגר שיכול לזכות אתכם ב-10 נקודות בונוס.
- **שימו לב – במהלך הסמסטר כתבנו פרויקטים עם design ופונקציונאליות דומים מאוד לאלו שנדרשים מכם בפרויקט המסכם ובחלק מהדרישות, עד כדי התאמה של קוד קיים.**
- ניתן להגיש את הפרויקט באופן יחידני/בקבוצות של 2 עד 3 סטודנטים.
- סטודנטים שמגישים את הפרויקט בשלשה חייבים לממש את סעיף 15 בסעיף ההנחיות בחלק 2 (עוסק ב-`CompletableFuture<T>`).

תמצית: שרת אשר יכול לבצע פעולות אלגוריתמיות שונות בהתאם לסוג של ה-task שהתקבל. הפעולות השונות והטיפול בלקוחות בשרת עשויים להתבצע במקביל (concurrently).
עליכם להשתמש בעקרונות OOD (Object Oriented Design) ו-Multithreading שלמדנו במהלך הסמסטר, בפרט בפרויקטים שכתבנו בשיעורים 11 ו-12.

חלק 1

בחלק הראשון עליכם לממש טיפוס מניה (enum) בשם `TaskType` ומחלקה בשם `TaskWrapper`.

`TaskType` (enum)

- על טיפוס זה לייצג הן **סוג של פעולה לביצוע ואת העדיפות שלה.**
- שימו לב, על אף שהשדות של enum (קבועים) מוגדרים כ-final, ניתן להכיל משתנה שיקבל ערכים שונים בין שדה אחד לאחר.
- לצורך פונקציונאליות זו, ראשית יש להגדיר משתנה ב-enum בשם priority (מסוג int או Integer) וכן לקבל ב-constructor את העדיפות.
- השדות של `TaskType` יהיו:
  1. IO (בעל עדיפות 1)
  2. COMPUTATIONAL (בעל עדיפות 2)
  3. UNKNOWN (בעל עדיפות 3)
- **בהגדרה של כל שדה יש להעביר את העדיפות שלו וכן לממש את ()toString כך שתוחזר מחרוזת עם שם השדה.**
- על הטיפוס לממש את המתודות: ()getPriority אשר מחזירה את הערך של משתנה ה-priority ו-(setPriority(int oPriority אשר ישנה את העדיפות של הקבוע בהתאם למספר שהועבר.

המחלקה `TaskWrapper`

- המחלקה `TaskWrapper` אשר תממשו **דומה מאוד ל-`PriorityRunnable`** שכתבנו בשיעורים 3 ו-7.
- אתם רשאים לעשות שימוש במחלקות קונקרטיות ובממשקים בהם השתמשנו בפרויקט ה-`GenericService`.
- מחלקה זו תייצג פעולה שתרוץ במסגרת worker thread כך שניתן יהיה להחזיר תוצאה של חישוב כלשהו או לא להחזיר ערך בכלל.
- על כן, **`TaskWrapper<V>`** צריכה לממש את הממשק **`RunnableFuture<V>`**
  הכרנו ממשק זה את והשתמשנו במחלקה **`FutureTask<V>`** אשר מממשת את **`RunnableFuture<V>`**.

*** תזכורת: `RunnableFuture<V>` יורש משני ממשקים: `Runnable` ו-`Future<V>`.

1. ממשק זה מייצג תוצאה עתידית של פעולה שניתן לבצע על-ידי קריאה למתודה ()run.

2. את התוצאה ניתן להשיג ע"י קריאה למתודה ()get (ראו מתודות convert בפרויקט GenericService).

- המחלקה `TaskWrapper<V>` תספק פרטים אודות ה-task לביצוע ע"י הכלה של TaskType.
- אובייקטים מטיפוס `TaskWrapper` יהיו ברי-השוואה באמצעות שדה ה-priority של ה-data member מסוג TaskType. חשבו אילו מתודות עזר עליכם להגדיר.
- עליכם להגדיר שני constructorים כך שניתן יהיה לקבל אובייקטים הן מסוג Runnable והן מסוג `Callable<V>` וכן אובייקט TaskType שייתן את הפרטים על ה-task לביצוע.
- שימו לב אילו פרמטרים יש להכריז בכל אחד מה-constructorים בהתייחס לטיפוס `RunnableFuture`.
- **היזכרו** ב-design pattern בו השתמשנו ב-`PriorityRunnable`, הוא בהחלט יסייע לכם גם כאן. ה-design יכתיב data members שיש לכלול במחלקה.
- חשבו האם data memebers שה-design דורש מאתנו צריכים להיות מהטיפוס הקונקרטי או מהטיפוס האבסטרקטי – (עקרון Liskov Substition)
- בצעו override למתודה ()toString כך שאם נבצע ()println לאובייקט מסוג `TaskWrapper` נקבל את סוג ה-task ואת העדיפות שלו.
- בנוסף עליכם לממש את המתודות הבאות:
  - `public boolean cancel(boolean mayInterruptIfRunning)`
    מתודה זו תנסה לבטל את ביצוע הפעולה. הפרמטר מסמן האם לבצע interrupt ל-thread שמבצע את הפעולה.
  - `public boolean isCancelled()`
    מחזיר true אם הפעולה בוטלה לפני שהושלמה באופן תקין
  - `public boolean isDone()`
    מחזיר true אם הפעולה הושלמה
  - `public V get() throws InterruptedException, ExecutionException`
    מחזיר את תוצאת החישוב
  - `public V get(long timeout, TimeUnit unit) throws InterruptedException, ExecutionException, TimeoutException`
    מחזיר את תוצאת החישוב במידה ולא עבר הזמן שהוגדר כ-timeout

- חשבו כיצד ה-design pattern שתבחרו יסייע לכם לממש את המתודות לעיל בקלות.
- אתם רשאים להוסיף מתודות ושדות בהתאם לצורך – קיימים data members ומתודות נוספות שעליכם לממש על-מנת לספק את הפונקציונאליות המלאה של מחלקה זו.

חלק 2

- בחלק השני נעסוק במתן מענה לבעיות אלגוריתמיות, בדומה לפרויקטים שכתבנו יחד במהלך הסמסטר.
- כל בעיה היא משימה שהמערכת צריכה לבצע ולהחזיר את תוצאת החישוב ללקוח.
- המשימות מופיעות בסיפא של המסמך לאחר רשימת ההנחיות המלאה.
- הבעיות הספציפיות יועברו על-גבי Socket הלקוח אל השרת וכל בקשה תטופל ב-thread נפרד באמצעות Handler קונקרטי (ראו שיעור 7).
- **בשיעור 7 עסקנו ב-networking, בפרט Multi-threaded Server Architecture, TCP & Sockets.**
  אתם רשאים לעשות שימוש במחלקות שמימשנו בשיעור זה (כולל הרחבה/שינוי).
- אני מאפשר לכם לקבוע את ה-API ברכיבים החדשים שתוסיפו (בפרט חתימות מתודות, שמות ממשקים/מחלקות חדשות).

דגשים להרצת הפרויקט משיעורים 10 ו-11

- במתודת ה-()main במחלקה MatrixAsGraph עשינו שימוש ב-GenericService עם API מעט שונה מזה שכתבנו בכיתה (ראו מתודות convert)
- ה-Callable שכתבנו אשר מגדיר פעולת חישוב מקבילי של רכיבים קשירים הועבר למתודה apply ב-GenericService. חתימת המתודה:

```
public<V> Future<V> apply(final Callable<V> callable,
Function<Callable<V>,RunnableFuture<V>> runnableTFunction)
throws InterruptedException
```

- על כן, **<u>השינוי היחיד</u>** שעליכם לבצע על מנת להריץ את הפרוויקט בשלמותו הוא אך ורק במתודת ה-main() במחלקה `MatrixAsGraph`:
  1. באפשרותכם לבצע שינויים במחלקה `GenericService`
  2. להוסיף למחלקה `MatrixAsGraph` ThreadPool לבחירתכם ולהזין את ה-Callable (**מומלץ**).
  המימוש למחלקה `MatrixAsGraph` מופיע ב-moodle תחת שיעור 11 ודוגמה לשימוש ב-Executor תוכלו למצוא בקוד משיעורים 1 ו-7.
  אם תבחרו באופציה השניה, ערך ההחזרה מהמתודה submit() ל-ThreadPool יהיה במקרה זה: Future<List<HashSet<Index>>> בדומה לקוד הנוכחי.

<u>דגשים להרחבת המחלקות Server ו-Handler (ראו קוד משיעור 7)</u>

1. אתם רשאים להשתמש בקוד משיעור 7, להרחיבו ולשנותו.
2. אתם רשאים להרחיב את ה-abstract Handler ולשנותו בהתאם ל-API שתיצרו.
3. תזכורת: פעולת accept() של ה-ServerSocket תחזיר לנו במקרה של הצלחה socket תפעולי.
אנו נעביר את InputStream וה-OutputStream ל-Handler רלוונטי אשר יבצע את הלוגיקה שהוגדרה לו ב-thread נפרד. **הפרוויקט שכתבנו בשיעור זה יסייע לכם לממש את הדרישות.**
דוגמאות לשימוש ב-Handlers וב-Socket/ServerSocket תפעולי מופיע בפרוויקט משיעור 7 כולל שימוש ב-Decorators, לדוגמה:

```
Runnable handleLogic = () -> {
    try {
        requestConcreteHandler.handle(request.getInputStream(),
                request.getOutputStream());
        // Close all streams
        request.getInputStream().close();
        request.getOutputStream().close();
        request.close();
    } catch (IOException ie) {
    }
};
...
   var objectInputStream = new ObjectInputStream(
           (client.getInputStream()));
   inTransaction transaction = (inTransaction)
   objectInputStream.readObject();
```

4. אינני מגביל אתכם ל-OutputStream/InputStream ספציפי וכן לייצוג המידע שיועבר ע"ג ה-Socketים, כל עוד תאפשרו את הפונקציונאליות של המשימות שהוגדרו להלן.
5. הטיפול בכל בקשה (כלומר קריאה למתודה handle של ה-Handler הקונקרטי) תעשה במסגרת Thread ב-ThreadPoolExecutor: כלומר עלינו לבצע execute ל-runnable שעוטף את המתודה handle כפי שמימשנו בפרוויקט.
6. חשבו האם פעולת accept() של ה-ServerSocket צריכה להתבצע ב-thread הראשי של השרת או לעטוף אותה ב-thread נפרד.
7. וודאו כי יש אפשרות להפסיק את פעולת השרת ושפעולה זו נעשית באופן שהוא Thread-Safe באמצעות המנגנונים השונים שלמדנו במהלך הסמסטר.
8. יש לוודא שאתם סוגרים את ה-resources השונים עם סיום שימוש מוצלח או במקרה של כישלון באמצעות: try-with-resources
9. היררכיית ה-Handlerים והחתימות שלהם הם לשיקולכם (כדאי לחשוב כיצד ניתן לטפל ברמה האבסטרקטית ולהבין כיצד הבדלים בין משימה אחת לאחרת יכול להשפיע על בחירתכם).

<u>הנחיות</u>

1. עליכם לכתוב מימוש למערכת שתספק את השירותים הנדרשים לפתרון המשימות (ברשימת המשימות למימוש) תוך שימוש בעקרונות ה-OOD ו-Multithreading שלמדנו במהלך הסמסטר.
בפרט מהפרוויקט שכתבנו בשיעור 11 כולל שימוש מתקדם ב-Generics וטיפוסים פרמטרים.

2. להזכירכם, בפרויקט 11 מימשנו מערכת שמספקת מגוון שירותים וכוללת בין היתר אלגוריתם Multi-Source Connected Components בו השתמשנו למציאת כל קבוצות ה-1ים במטריצה בינארית.
3. בפרויקט המסכם בדומה לפרויקט משיעור 11 ישנן משימות אשר דורשות מכם לבצע מעבר על המטריצה מכמה מקורות במקביל (concurrently), באופן יעיל ונכון כפי שעשינו במחלקה TraverseLogic<T>.
4. כל חיפוש מאינדקס מקור עשוי להתבצע ב-Thread משלו. על כן, יש לוודא שאין התנגשות במידע שנשמר ב-Threadים באמצעות הצהרה על מבני נתונים מסוג ThreadLocal<T>.
5. את אתחול הטיפוסים המוצהרים מסוג ThreadLocal<T> יש לבצע באמצעות המתודה:
```
public static <S> ThreadLocal<S> withInitial
(Supplier<? extends S> supplier)
```
כפי שביצענו בפרויקט 11.
6. להזכירכם טרם ביצוע מעבר על גרף ממקור נתון יש לגשת גישה למידע מסוג `ThreadLocal` באמצעות המתודה: `public T get()`
7. ערך ההחזרה מחיפוש מקומי צריך להיות טיפוס קונקרטי ולא `ThreadLocal`.
8. שימו לב שאם תממשו אלגוריתם לחיפוש מקבילי, אך בסופו של דבר תריצו אותו במסגרת אותו Thread (לא יעיל), מבני נתונים שהוגדרו כ-`ThreadLocal` עדיין יכילו מידע שנשמר במסגרת חיפוש קודם - וודאו שאין בהם מידע לפני שמתבצע חיפוש חדש. כתבנו יחד מימוש שנותן מענה חלקי לבעיה.
9. **בניגוד לשימוש במבני נתונים שהם מקומיים לחיפוש**, איחוד של המידע (כלומר הוספתו למבנה נתונים אחד) צריך להתבצע באמצעות מבני נתונים Thread-Safe או באמצעות עטיפה של מבנה נתונים קונקרטי באמצעות אחת מהמתודות הסטטיות של **`Collections.Synchronized`**.
10. כמו כן, במידה ומדובר באלגוריתם שמתבצע באופן מקבילי, קריאה בסגנון algorithm.traverse(graph) צריכה להתבצע ב-Thread נפרד.
11. השתמשו במידת הצורך ב-`ReentrantReadWriteLock` באמצעות אחד או שני המנעולים שמכיל (ראו פרויקט משיעור 12)
12. השתמשו ב-Streams ו-method references בהתאם לצורך. לדוגמה:
```
s.getData()).stream().filter(index -> matrix.getValue(index) ==1)
    .map(adjacentIndex ->
     new GraphNode<>(adjacentIndex,s))
    .collect(Collectors.toList());
....
hashSets.sort((Comparator.comparingInt(HashSet::size)));
```
13. ניתנת חשיבות גם ל-Exceptions עליהם תכריזו ותטפלו, ב-Annotations עבור ה-Compiler ולמטרות תיעוד – אלו רק חלק מ הפרקטיקות שהשתמשנו בהן בפרויקטים במהלך הסמסטר.
14. יש לתעד את הקוד באופן תמציתי כולל פרמטרים
15. בונוס 5 נקודות: באחת מהמשימות (לפחות), יש להשתמש באובייקט מסוג `CompletableFuture<T>` על-מנת להגדיר Callback על תוצאה של פעולה חישובית. שימו לב, הפעולה החישובית צריכה להתבצע במסגרת איזשהו `ThreadPoolExecutor` ואילו את ה-Callback ב-Thread ב-`ForkJoinPool` . דוגמאות וקוד לשימושכם נמצא בשיעור 13 בקובץ `MyCompletableFuture.java`.

רשימת המשימות

- בסעיף זה נציג את המשימות שהמערכת שלכם תפתור.
- בדומה לפרויקט משיעור 11, אנו מעוניינים לממש אלגוריתמים באופן בלתי תלוי בבעיות שאותן אנו נפתור. כלומר, נרצה לכתוב קוד שיפתור את המקרה הכללי ובאמצעות התאמות נפתור מגוון use cases.
- בבדיקה של הפרויקט אתן חשיבות לנכונות הקוד כמו גם על Thread-safety ו-design יעיל.
- **קיים קשר הדוק בין משימות 1,2 ו-4**

משימה 1- מציאת כל קבוצות ה-1ים (אינדקסים ישיגים כוללים אלכסונים)

שימו לב –כתבנו פרויקט שעונה (בין היתר) על משימה דומה מאוד למשימה זו, עד כדי התאמה.

קלט: מערך 2D של int או Integer

פלט: רשימה של כל קבוצות ה-1ים ללא כפילויות בהתאם לאינדקס שלהם (רשימה של HashSet<Index>)

- עליכם להוסיף פונקציונאליות לאלגוריתם הקיים. מעתה אינדקס ישיג (reachable) יכלול גם את האלכסונים. כיוון שה-design שלנו הוא loosely-coupled ניתן לבצע הרחבה למחלקה אחת בלבד, זאת מבלי לשבור את הקוד הקיים ולשנות את האלגוריתם עצמו.
- להזכירכם, אנו הפרדנו בין ייצוג של מטריצה בינארית, לייצוג של מטריצה שניתן לרוץ עליה כגרף בהתייחס ל-Index. שוחחנו היכן כדאי לבצע את ההרחבה/תוספת.

** החל ממשימה זו ניתן לנוע מאינדקס שערכו 1 לאינדקס אחר שערכו 1 בכל הכיוונים (8 כיוונים) בהתאם למגבלות הקלט.

<u>משימה 2- מציאת כל המסלולים מאינדקס מקור לאינדקס יעד</u>

- קלט: מערך 2D של int או Integer, אינדקס מקור ואינדקס יעד
- פלט: רשימה שכוללת את. כל המסלולים התקינים מאינדקס המקור לאינדקס היעד (ללא חזרה על אינדקסים). הרשימה צריכה להיות ממוינת מהקטנה לגדולה מבחינת כמות האינדקסים בכל רשימה (כפי שביצענו במתודת ה-()main במחלקה `MatrixAsGraph`
- לדוגמה בהינתן המערך הבא, אינדקס מקור (0,0) ואינדקס יעד (4,4):

  [1, 0, 0, 1, 0]
  [0, 1, 0, 0, 1]
  [1, 0, 1, 1, 1]
- הפלט יהיה:

  [(0,0), (1,1), (2,2), (2,3), (2,4)]
  [(0,0), (1,1), (2,2), (2,3), (1,4),(2,4)]

<u>משימה 3- מציאת מסלולים קצרים ביותר מאינדקס מקור לאינדקס יעד</u>

- קלט: מערך 2D **עד גודל 50 X 50** של int או Integer, אינדקס מקור ואינדקס יעד
- פלט: רשימה עם המסלולים הקצרים ביותר מאינדקס המקור לאינדקס היעד (קבוצות ה-1ים שכוללות את המספר הקטן ביותר של אינדקסים בין המקור ליעד.
- כידוע יכולים להיות כמה מסלולים קצרים ביותר בין אינדקס מקור לאינדקס יעד
- לא תמיד כדאי לבצע סריקה של מטריצה מכמה אינדקסים במקביל.למדנו שקיים Overhead למימוש כזה והוא כדאי כאשר הקלט מספיק גדול.
- **במשימה 3 (ובמשימה זו בלבד) עליכם לבצע את החישובים במסגרת thread אחד בלבד.**

<u>משימה 4 - משחק צוללות</u>

קלט: מערך דו-מימדי של int או Integer

פלט: מספר הצוללות על לוח המשחק

חוקים:

1. צוללת יכולה להיות שני 1ים (לפחות) במאונך
2. צוללת יכולה להיות שני 1ים (לפחות) במאוזן
3. לא יכולים להיות שני 1דים באלכסון אלא אם כן עבור שניהם מתקיימים סעיפים 1 ו-2.
4. המרחק המינימלי בין שתי צוללות (ללא קשר לאוריינטציה) הוא משבצת אחת

דוגמה 1 לקלט לא תקין:

[1, 1, 0, 1, 1]
[1, 0, 0, 1, 1]
[1, 0, 0, 1, 1]

דוגמה 2 לקלט לא תקין

[1, 0, 0, 1, 1]
[1, 0, 0, 1, 1]
[0, 1, 0, 1, 1]

דוגמה 3 לקלט תקין- 2 צוללות

[1, 0, 0, 1, 1]

[1, 0, 0, 1, 1]
[1, 0, 0, 1, 1]

דוגמה 4 לקלט תקין- 3 צוללות
[1, 1, 0, 1, 1]
[0, 0, 0, 1, 1]
[1, 1, 0, 1, 1]

שימו לב - שימוש באלגוריתם שכתבנו יחד בשיעור 11 עשוי לפתור את הבעיה אך בצורה לא יעילה.
חשבו אילו שינויים יש לבצע באלגוריתם לגילוי רכיבים קשירים מכמה מקורות שונים, תוך התחשבות בחוקי משחק הצוללות.

הערות
1. אתם רשאים לבצע את הפרויקט ביחידים או בזוגות.
2. ההגנה תיערך באופן מקוון.
3. במעמד ההגנה, כל אחד מהמגישים יציג את הקוד ב-IDE לבחירתו באמצעות שיתוף מסך ב-zoom.
4. אם אין ברשותכם מיקרופון שמחובר למחשב, ניתן יהיה לשוחח איתי בטלפון (במקביל לשיתוף המסך**).** אין צורך במצלמה.
5. עליכם לבחור באחד משני המועדים האפשריים להגנה.
6. בתחילת אוגוסט יעלו שני גיליונות excel משותפים עם חלונות זמן – בחרו את השיבוץ המועדף עליכם.
5. כל אחד מהסטודנטים חייב להכיר את הפרויקט במלואו.
6. **אנא מכם - אל תעתיקו**. לרשותכם נמצאים ב-moodle כל הקבצים שכתבנו במהלך הסמסטר, הקלטות שיעורים וחומרי עזר נוספים.
7. ניתן ליצור איתי קשר במייל nathand@hit.ac.il