פיתוח תוכנה מתקדם 2 – סמסטר א' מועד ב' תשפ"ב

תזכורת: כתובת מערכת הבדיקות: https://cktest.cs.colman.ac.il/

שם הקורס, PTM2 מועד ב'. לאחר הורדת המבחן ממערכת הבדיקות. העתיקו את כל קבצי ה Java לתוך הפרויקט ב package בשם test.
במבחן זה 3 שאלות, חובה לענות על כל 3 השאלות ולהגיש למערכת הבדיקות במוד הגשה סופית לפני סוף המבחן.

הערה:

אנחנו מזהירים מראש, בנוסף למערכת הבדיקות, הבחינה תיבדק בצורה ידנית.
כל ניסיון להטעיית מערכת הבדיקות ייחשב כרמאות, פגיעה בערכים ובעקרונות היסוד של האקדמיה ובהם טוהר הבחינות.

שאלה 1 – תכנות מקבילי באמצעות ת'רדים (35 נק')

בשאלה הראשונה, תצטרכו להביא לידי ביטוי את הידע שצברתם במהלך הקורס בנושא ת'רדים ומשתנים אטומיים.

בקובץ ParallelCounter.java עליכם לממש את המתודה parallelCountIf.
המתודה מקבלת רשימה של אלמנטים מסוג list כלשהו, פרדיקט p, ומספר ת'רדים.
על המתודה לפתוח את מספר הת'רדים הרצוי ובכל ת'רד לעבור על תת-רשימה של list.
ניתן להניח כי מספר האיברים ברשימה list מתחלק ללא שארית במספר הת'רדים.
המתודה תחזיק משתנה מסוים, היא תעלה ותוריד את הערך של אותו משתנה בהתאם לתשובה שהפרדיקט p מחזיר על כל אחד מהאיברים של list. אם p מחזיר אמת, נעלה את הערך של המשתנה ב-1, אם p מחזיר שקר נוריד את הערך ב-1.
המתודה **תחכה לסיום פעולתם של הת'רדים שפתחה** ואז תחזיר את המספר שצברה.
הביטו ב MainTrain1 כדי להבין את אופן הפעולה הרצוי:

- יצרנו רשימה המכילה 10 ערכים שונים.
- ThreadCounter שייך לבדיקה. מטרתו לספור ברקע כמה ת'רדים נפתחו. אין לערוך או להגיש קובץ זה.
- parallelCountIf מקבלת רשימה, פרדיקט שבודק תנאי מסוים על איבר ברשימה (לדוגמא, האם האיבר הוא אי זוגי) ומס' ת'רדים של 2.
  - ע"פ פרמטרים אלו נפתחו שני ת'רדים, בכל אחד מעובדים 5 איברים של list ומחזיר ערך בהתאם למספר האיברים שעמדו בתנאי.
  - מספר הת'רדים שנפתחו צריך להיות 3 : ה main + שני הת'רדים שבקשנו לפתוח.

לדוגמא, עבור התנאי "האם האיבר הוא אי זוגי" בהינתן הרשימה [1,2,3,5]:

עבור האיברים האי זוגיים 1, 3, 5 שעומדים בתנאי, נעלה את הערך של המשתנה 3+, עבור האיבר 2 שלא עומד בתנאי נוריד את הערך (1-). לכן, הערך שיחזור הוא כמובן 2.

עליכם לערוך ולהגיש רק את ParallelCounter.java.
הבדיקה במוד ההגשה דומה, אך היא כוללת אלמנטים נוספים שבודקים עמידה בהגדרות לעיל כגון רשימה אחרת או תנאי אחר.

<u>הערה</u>: **חל איסור לעשות שימוש ב synchronized.**

שאלה 2 – Fork Join Pool (30 נק')

מספרי לוקאס הם סדרה של מספרים טבעיים. הגדרתה דומה לזו של סדרת פיבונאצ'י:
כל איבר בסדרה הוא סכום של שני קודמיו.
היא נבדלת מסדרת פיבונאצ'י בתנאי ההתחלה: האיבר האפס והאיבר הראשון הם 2 ו-1 בהתאמה.

תחילת הסדרה היא: 2, 1, 3, 4, 7, 11, 18, 29, 47, 56 , ...

ההגדרה הרקורסיבית של הסדרה היא:

$$L_n := \begin{cases} 2 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ L_{n-1} + L_{n-2} & \text{if } n > 1. \end{cases}$$

לדוגמא, סכום האיבר הרביעי בסדרה הוא 4 (סכום האיבר השלישי (3) + סכום האיבר השני (1)).

עליכם לממש את המחלקה LucasNumbers.java כסוג של RecursiveTask.

בהינתן איבר מסוים בסדרה עליכם לחשב אותו באופן רקורסיבי. כדי ליעל את החישוב עליכם להשתמש ב Fork Join pool.

ב MainTrain2 נבצע מספר בדיקות המאמתות כי אתם באמת עשיתם שימוש נכון ב Fork Join Pool.
כמו כן, נבדוק כי הערך שחזר הוא באמת הערך הנכון בסדרה.

יש לציין כי ב"מוד הגשה" יהיו בדיקות עמוקות יותר עם פרמטרים שונים.

שאלה 3 – אופטימיזציות קוד (35 נק')

Simple Moving Average / ממוצע נע הוא כלי סטטיסטי המחשב ממוצע המתקבל מנתונים על פי חלון זמן מסוים והוא מחושב באופן הבא:

$$SMA_w = \frac{1}{w} \sum_{i=n-w+1}^{n} x_i$$

בקובץ BadCode.java מצויה המתודה SMA שלוקחת כקלט ווקטור ומשתנה בשם window המייצג את גודל ה"חלון" ומה שהיא מחזירה זה ווקטור עם הדאטה לאחר שעבר "החלקה".

המתודה SMA עושה שימוש במבנה נתונים מסוג תור (Queue) כך שבכל איטרציה היא תכניס לתוכו איבר אחר מתוך הווקטור ובנוסף, שתי מתודות עזר.

מתודת העזר הראשונה נקראת fixSum, היא מקבלת תור ו-window, היא בודקת האם מספר האיברים בתור גדול מגודל החלון, אם כן, היא תסיר אותו מהתור.
לאחר מכן, היא תרוץ על כל האיברים בתור ותחזיר את הסכום שלהם.

מתודת העזר השנייה נקראת calcMean, מה שהיא עושה זה פשוט לקבל את הסכום שחושב בfixSum, את window ומה שהיא עושה זה לחשב את הממוצע על פי גודל החלון.

דוגמא לממוצע נע:

בהינתן הקלט: {1, 3, 5, 6, 8} נרצה לחשב את הממוצע הנע של הווקטור על פי חלון בגודל 3.
נקבל את הערכים הבאים (החלון מסומן בצהוב):

{1, 3, 5, 6, 8}	0.33333…
{1, 3, 5, 6, 8}	1.33333…
{1, 3, 5, 6, 8}	3.0
{1, 3, 5, 6, 8}	4.66666…
{1, 3, 5, 6, 8}	6.33333…

לכן הווקטור שיחזור הוא: {0.333.., 1.333.., 3, 4.666.., 6.333..}

בקובץ BadCode.java תמצאו את SMA, המימוש הלא יעיל של המתודה הנ"ל.
עליכם לכתוב את המתודה SMA_Opt ב GoodCode.java כך שתהיה יעילה פי 5 מ SMA.

<u>הערה</u>: **אין להשתמש בפונקציות של BadCode.java.**

יש לציין כי ב"מוד הגשה" יהיו בדיקות עמוקות יותר עם פרמטרים שונים.

הגשה

עליכם להיכנס למערכת הבדיקות בכתובת: https://cktest.cs.colman.ac.il/ ולהגיש ל PTM2 מועד ב' את הקבצים LucasNumbers.java, ParallelCounter.java, GoodCode.java.

בכל הגשה יש להגיש את כל הקבצים (להתייחס לפלט רק של השאלות שעניתם עליהן).

ניתן להגיש במוד אימון ובמוד הגשה כמה פעמים שתרצו עד לסוף המבחן.

בסוף המבחן יש להגיש במוד הגשה ואז במוד הגשה סופית. אחריה תקבלו מס' אסמכתא בין ארבע ספרות.
לאחר הגשה במוד הגשה סופית לא תוכלו להגיש יותר.

בהצלחה!