תרגיל בית 3 – בטיחות תהליכונים

אופן ההגשה:

- במידה ובתרגיל מופיעה דוגמא של פלט מסויים, הקפידו שהפלט שלכם יהיה זהה
- תשובות בכתב הקפידו להזין לתוך קובץ word, תוך ציון מספר השאלה ומספר הסעיף
- אם בתרגיל הוזכרו במפורש שמות מחלקות/משתנים וכד', או קבצים בהם יש להשתמש – יש לדייק בשמות, ולהשתמש בקבצים שהוגדו. יש לממש את השאלות ב java
- **יש לרשום את שם ות"ז המגישים בראש כל קובץ המהווה חלק מפיתרון התרגיל. (לדוגמא: אם בניתם 3 מחלקות, וקובץ וורד לתשובות, אז השם והת"ז יופיע בראש שלוש המחלקות וגם בראש קובץ הוורד). בקובץ הוורד יש לרשום את השם בעברית**
- יש להכניס כל חבילת קוד לתיקייה נפרדת
- במידה והמטלה מכילה יותר מקובץ אחד, יש לכווץ את כלל הקבצים לקובץ zip (ולא פורמטים אחרים כמו rar), כאשר שם הקובץ המכווץ כולל את ת"ז המגישים. למשל, אם הגישו את העבודה בעלי ת"ז 11111 ו- 22222, שם הקובץ יהיה 11111_22222.zip . לאחר הכיווץ, העלו את קובץ ה zip בלבד ללימוד.
- ההגשה בזוגות בלבד (אלא אם כן אושר אחרת ע"י המרצה)
- יש להגיש את הפתרון עד למועד שנקבע בלימוד
- חלק מהנבדק בתרגיל הוא עמידה בנקודות הנ"ל. הקפידו לעמוד בהן.

בהצלחה!

שאלה 1

פתחו חבילה בשם assig3_1 וממשו תוכנית המדמה משחק עץ או פלי, עם החוקים הבאים:

1. במשחק ישנם שני משתתפים, ומטבע אחד. לכל משתתף במשחק ישנו מונה, הסופר את כמות הפעמים שהתקבל עץ (מתקבל הערך true מהפוקנציה ()flipCoin).
2. המשחק נגמר, לאחר ששני המשתתפים הטילו את המטבע 10 פעמים
3. המנצח הוא השחקן עם מספר ההטלות הרב ביותר שבהן התקבל עץ

המחלקה GamePlay מייצגת את המשחק עצמו, את המטבע ואת מספר ההטלות

ממשו את המחלקה GamePlay באופן הבא:

1. שדה _coin_available המתאר האם המטבע זמין
2. שדה _rounds_counter המתאר את כמות ההטלות שבוצעו במשחק
3. פונקציה (makeCoinAvail(boolean val:
   a. הופכת את המטבע לזמין או לא זמין לפי הערך val
   b. במידה והמטבע הופך זמין, מודיעה על כך לשאר התהליכונים שממתינים למטבע
   c. מעדכנת את השדה _coin_available בהתאם לערך שקיבלה
4. פונקציה ()flipCoin:
   a. אם המטבע אינו זמין, התהליכון:
      i. ימתין עד שהמטבע יהפוך זמין (זכרו להשתמש ב guarded suspension)
      ii. ידפיס את שמו (באמצעות ()getName של המחלקה Thread), ואת העובדה שהוא נכנס להמתנה. למשל:
      Thread-0 is waiting for coin
   b. אם המטבע זמין, התהליכון:
      i. ידפיס שהוא מטיל מטבע. למשל:
      Thread-1 is flipping coin
      ii. יהפוך את המטבע ללא זמין במהלך ההטלה
      iii. יקדם את מספר ההטלות ב-1
      iv. יגריל ראנדומאלי 0 או 1
      v. יהפוך את המטבע שוב לזמין, ויודיע על כך לתהליכונים אחרים
   c. הפונקציה תחזיר את תוצאת ההטלה (0 = שקר/כשלון, 1 = אמת/הצלחה)
5. פונקציה ()getNumOfRounds :
   a. מחזירה את מספר ההטלות במשחק

המחלקה Gamer מייצגת תהליכון שחקן אשר מטיל את המטבע שוב ושוב

ממשו את המחלקה Gamer באופן הבא:

1. שדה goodFlipsCounter הסופר את כמות ההטלות שהשחקן ביצע והצליחו
2. שדה מסוג GamePlay המאותחל בבנאי (האובייקט מתקבל מבחוץ)
3. פונקציית ()play המתבצעת בלולאה, ובכל איטרציה:
   a. כל עוד שלא ביצעו לתהליכון INTERRUPT וגם מספר ההטלות <u>במשחק כולו</u> קטן או שווה ל 10:
      i. נסה להטיל מטבע, ואם הצלחת קדם את goodFlipsCounter ב-1
      ii. לך לישון למשך שנייה אחת
4. GETTER בשם getScore המחזיר את מספר ההטלות שהצליחו

המחלקה Judge מייצגת שופט במשחק, המאשר לשחקנים מתי הם רשאים להטיל את המטבע ומתי לא, ע״י הפיכת המטבע לזמין / לא זמין

ממשו את המחלקה Judge באופן הבא:

1. לולאה, בה כל עוד לא ביצעו לתהליכון interrupt:
   a. השופט הופך את המטבע ללא זמין למשך שנייה
   b. השופט הופך את המטבע לזמין למשך חצי שנייה

במחלקה הראשית של התוכנית:

1. צרו משחק חדש
2. צרו שני שחקנים
3. לאחר שהשחקנים <u>מסיימים</u> לשחק, מודפסת הודעה על השחקן שניצח (השחקן בעל הניקוד הגבוה ביותר). למשל:
   player 1 wins
   במידה ולשחקנים ניקוד זהה, יודפס תיקו כך:
   tie

ענו על השאלות הבאות:

1. מהם ה Invariant, Post and Pre condition של המתודה flipCoin במחלקה GamePlay?
2. נניח כי התווספה למשחק החוקיות הבאה:
   במידה ולשחקן יצא עץ, הוא בודק אם לשחקן היריב יצא פלי, ולא ממשיך לשחק עד שזה יקרה.
   איזה בעיית בטיחות עשויה להיגרם כתוצאה מתוספת זו? תנו דוגמא מוחשית של מהלכי משחק שיביאו לבעיה שציינתם.
3. מהו קטע הקוד הקריטי בתוכנית? איך זיהיתם אותו?
4. בפונקציה flipCoin – במידה ונשתמש בתנאי if רגיל בבדיקת הזמינות של המטבע במקום ב guarded suspension, איזה בעיה עלולה להתעורר בתוכנית?
5. במידה ואין שופט במשחק, והמשחק מתחיל שהמטבע זמין, האם ניתן לוותר על השדה coin_available_ במחלקה GamePlay, ועדיין להיות בטוחים שרק תהליכון **אחד** יטיל את המטבע בכל זמן נתון? מדוע?

שאלה 2

פתח חבילה בשם assig3_2 , והכנס לתוכה את קבצי הקוד שתחת התיקייה assig3_2 שצורפה למטלה.

בתוכנית קיים מימוש סמפור במחלקה MySemaphore, ומחלקה ראשית המייצרת תהליכונים ושולחת אותם לבצע עבודה במחלקה הנקראת HeavyWorker. התהליכונים שנוצרים במחלקה הראשית קוראים לפונקציות workA ו- workB במחלקה HeavyWorker

נדרש כי בתוכנית יתקיימו החוקים הבאים:

1. בכל רגע נתון יהיו עד שלושה תהליכונים בו-זמנית בתוך הפונקציה section1
2. מתוך אותם שלושה תהליכונים, רק אחד יכול להיות בתוך הפונקציה section2
3. תהליכונים שקוראים לפונקציה workB יצטרכו להמתין עד שהפונקציה workA תתבצע לפחות פעם אחת בשלמותה לפני שיוכלו לבצע את workB. במילים אחרות: workA תתבצע לפחות פעם אחת לפני ש workB תוכל להתבצע
4. יש לדאוג שהמחלקה HeavyWorker תהיה thread - safe

ניתן להניח כי הפונקציות section2 , section1 נקראות רק מתוך הפונקציה workA.

ערוך את הקוד בפונקציות workA, workB שבמחלקה HeavyWorker בלבד כך שהחוקים הנ"ל יתקיימו.
אין לערוך פונקציות אחרות, או מחלקות אחרות בקוד.

בשאלה זו מותר להגדיר מופעים של המחלקה MySemaphore כשדות של המחלקה ולהשתמש בהם **בלבד** למטרת הפתרון. אין להשתמש בכלים נוספים (כגון פונקציות synchronized, משתנים, פונקציות נוספות, תנאים, לולאות וכו')

ענו על השאלות הבאות:

1. הבט במימוש הסמפור במחלקה MySemaphore. מדוע אינו מקיים מניעת הרעבה? מה נחוץ לו ע"מ שיהפוך לכזה המקיים מניעת הרעבה?
2. האם נוכל להשתמש בסמפור במקום synchronized כדי להגן על קטע קוד קריטי? כיצד?

שאלה 3

בחן את קטע הקוד הנתון מטה.

איזה בעיית בטיחות עלולה להתעורר בקוד המדובר? תן דוגמא מוחשית של רצף פעולות שיביא לבעיה שתיארת

(הפקודה this.lock.tryLock() שקולה ל synchronized(this) {...} . היא תנסה לנעול את המנעול של המופע this, ותחזיר אמת אם התהליכון הצליח להשיג נעילה)

```
49      public static void main(String[] args) {
50          final BankAccount fooAccount = new BankAccount(1, 500d);
51          final BankAccount barAccount = new BankAccount(2, 500d);
52
53          new Thread(new Transaction(fooAccount, barAccount, 10d), "transaction-1").start();
54          new Thread(new Transaction(barAccount, fooAccount, 10d), "transaction-2").start();
55
56      }
57
58  }
```

```java
class Transaction implements Runnable {
    private BankAccount sourceAccount, destinationAccount;
    private double amount;

    Transaction(BankAccount sourceAccount, BankAccount destinationAccount, double amount) {
        this.sourceAccount = sourceAccount;
        this.destinationAccount = destinationAccount;
        this.amount = amount;
    }

    public void run() {
        while (!sourceAccount.tryTransfer(destinationAccount, amount))
            continue;
        System.out.printf("%s completed ", Thread.currentThread().getName());
    }

}
```

```java
public class BankAccount {
    double balance;
    int id;
    Lock lock = new ReentrantLock();

    BankAccount(int id, double balance) {
        this.id = id;
        this.balance = balance;
    }

    boolean withdraw(double amount) {
        if (this.lock.tryLock()) {
            // Wait to simulate io like database access ...
            try {Thread.sleep(10l);} catch (InterruptedException e) {}
            balance -= amount;
            return true;
        }
        return false;
    }

    boolean deposit(double amount) {
        if (this.lock.tryLock()) {
            // Wait to simulate io like database access ...
            try {Thread.sleep(10l);} catch (InterruptedException e) {}
            balance += amount;
            return true;
        }
        return false;
    }

    public boolean tryTransfer(BankAccount destinationAccount, double amount) {
        if (this.withdraw(amount)) {
            if (destinationAccount.deposit(amount)) {
                return true;
            } else {
                // destination account busy, refund source account.
                this.deposit(amount);
            }
        }

        return false;
    }
```