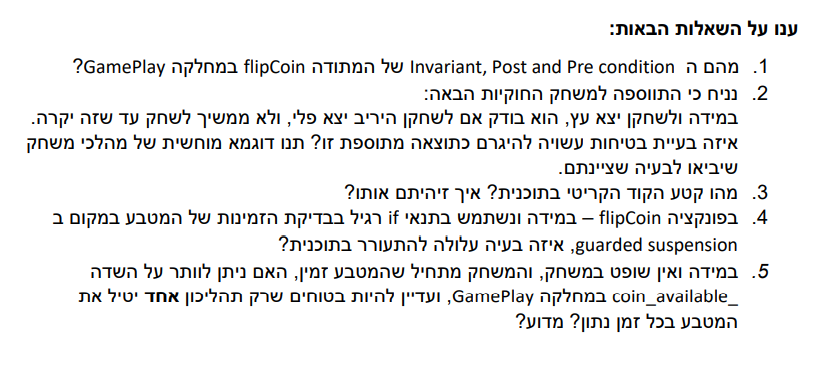
תכנות מערכות – מטלה 3

יוליה מושן – 319565610

גיל פסי – 206500936

קישור ל – GitHub : [GilPasi/SP-assignment-3 (github.com)](https://github.com/GilPasi/SP-assignment-3)



1.

Pre – condition : המטבע פנוי, הניקוד של המשתמש בו הוא X.

Post – condition : המטבע פנוי.

המטבע מצביע על הצלחה או כישלון .

במידה ומצביע על הצלחה הניקוד של המשתמש בו הוא X + 1 .

שמו של המשתמש בו הודפס.

Invariant : רק תהליכון אחד או אפס תהליכונים אוחזים במטבע.

2.אנו עלולים להיתקל בבעיית livelock התהליכון ממשיך לפעול שוב ושוב באופן סיזיפי.

לדוגמה:

* תהליכון 1 מתחיל לרוץ ומשתלט על המטבע ,הוא מטיל ומקבל עץ, מסיים את פעולתו.
* תהליכון 2 מתחיל לרוץ ומשתלט על המטבע, לאחר בדיקה מקבל כי תהליכון 2 לא קיבל פלי.

לכן הוא מנסה שוב להטיל את המטבע. אולם חשוב לזכור שמתודת הטלת המטבע אטומית ולכן המטבע לא ישתחרר . בדרך זו תהליכון 1 לא יקבל לעולם את הזכות להטיל שוב את המטבע ולשנות את גורלו של תהליכון 2 .

סיטואציה דומה יכולה לקרות גם באופן הפוך.

3. הקטעים הקריטיים בקוד הם הטלת המטבע, שינוי זמינות המטבע והשגת מספר הסיבובים.

זיהינו אותם בעקבות בעיות בטיחות \ נכונות שעלולות להיווצר במידה ולא יוגדרו באופן אטומי.

הטלת המטבע- במידה ולא יסונכרן ייתכן מצב בו שני השחקנים מטילים את המטבע באותו הזמן (האינווריאנטה לא מתקיימת). כמו כן מצב יותר חמור הוא שינוי המטבע במהלך התוכנית.

למשל שחקן 1 מטיל את המטבע ומקבל תוצאה שאינה מזכה בניקוד . לפני שהתוכנית מספיקה לבדוק זאת נלקח זמן מעבד עבור השחקן השני המבצע את אותה הפעולה . הוא דווקא יותר בר מזל ומקבל ערך המזכה בנקודה. שוב נלקח זמן מעבד לטובת שחקן 1 והוא בודקת את ערך המטבע – מזכה בנקודה למרות ששחקן 1 בכלל לא ראוי לקבל נקודה .

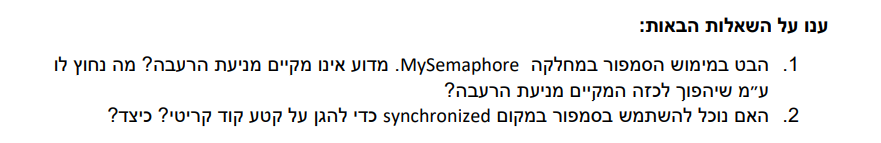
שינוי זמינות המטבע- גם כאן קיימת בעיית בטיחות ייתכן שתהליכון תפס מטבע פנוי. לפני שהספיק לשנות את ערכו ללא פנוי גם התהליכון השני הספיק להשתלט על המטבע.

השגת מספר הסיבובים – ייתכן שניכנס למתודה ונבדוק מספר סיבובים שגוי – כיוון שעד שהבדיקה התבצעה שונה מספר הסיבובים – בעיית נכונות.

4. במידה ונשתמש ב – if רגיל ולא בלולאת while אנו מסתכנים בבעיית liveness , יכול היווצר המצב הבא:

* תהליכון 1 מתחיל ריצה ומשתלט על המטבע. נלקח זמן מעבד.
* תהליכון 2 מתחיל ריצה. מנסה להשתלט על המטבע ללא הצלחה ולכן חוזר לישון. זמן מעבד עובר שוב לתהליכון 1.
* תהליכון 1 מסיים את תורו ומעיר את כלל התהליכונים.
* תהליכון 2 מתעורר וממשיך בתוכנית אבל לא חוזר לבדוק שוב האם המטבע פנוי .כך יצא שתהלכון 2 לא שיחק בכלל.

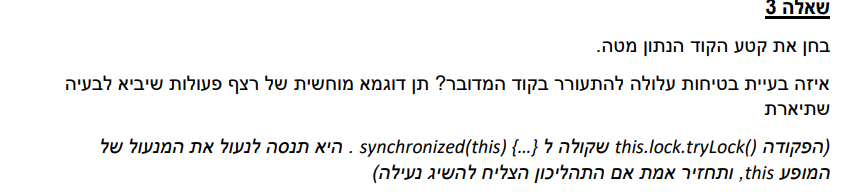
5.אכן ניתן לשחק את המשחק ללא שופט וללא coin\_available\_ . במקום המטבע נשתמש באובייקט במשחק כולו GamePlay כמנעול . אם נקפיד על כללי ה סנכרון הראויים מובטח לנו שרק תהליכון אחד משחק בכל עת.



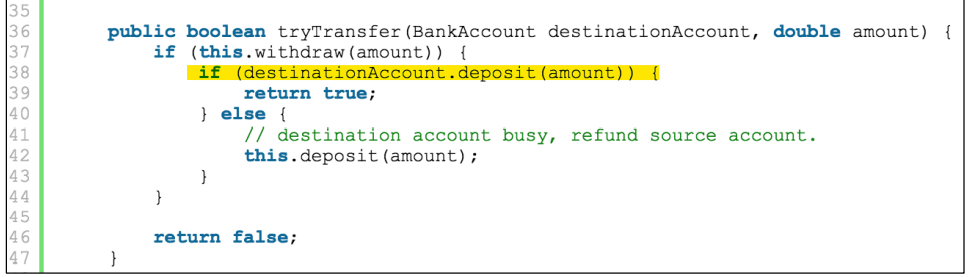
1.הסמפור במחלקת MySemaphore עלול לגרום למצב של הרעבה כיוון שאינו מקפיד על תכונת fairness , לא מתחייב שהתהליכון הראשון שביקש permit הוא גם הראשון לקבל אותו בתור.

ע"מ לממש מניעת הרעבה יש ליצור מנגנון המקפיד על הסדר. למשל להכניס את כלל התהליכונים הממתינים למבנה נתונים מסוג תור ולהעניק הרשאה בכל פעם רק לראשון בתור.

2. סמפור עם permit בודד הוא למעשה סנכרון. שכן הוא מאפשר רק לתהליכון אחד לבצע עבודה בקטע קוד נתון . לפיכך אכן מספור יכול להחליף synchronized.



המנעול נועל רק את החשבון הנוכחי. מכאן שהבעיה המרכזית נעוצה בשורה 38 :



שורה זו משנה את המאזן של חשבון היעד מבלי לנעול אותו. באופן הזה הוא חשוף לבעיות בטיחות.

דוגמה מוחשית:

1. תהליכון א' מפעיל את מתודת tryTransfer ומגיע עד לשורה 38.
2. תהליכון א' נכנס למתודת ההפקדה ונועל את חשבון foo . הוא מנסה לבצע את ההעברה לחשבון bar . כעת חשבון foo מכיל 490 וחשבון bar 500.
3. תהליכון אחד שולף את הערך ה- balance של bar (500) אך טרם מספיק להוסיף לו את ערך הטרנסאקציה נילקח זמן מעבד.
4. תהליכון ב' מבצע פעולה הליך דומה עד לשורה 37 .מאחר המנעול מופעל רק על חשבון foo הוא חופשי לבצע פעולות הכרוכות בנעילת bar . כלומר כעת לאחר מתודת withdraw הסכום בחשבון bar הוא 490 .

נלקח זמן מעבד לטובת תהליכון א'.

1. תהליכון א' ממשיך בפועלו מבלי לדעת על השינוי בחשבון bar. כלומר הוא מבצע חישוב פשוט:

הערך המקורי : 500 ועוד הערך החדש : 10 = 510. את ערך זה הוא מעדכן בחשבון bar.

ידוע לנו שזה לא הערך הנכון כיוון שתהליכון ב' החל בתהליך ההעברה. לכן הערך אמור היה להיות 490 + 10 = 500 .