תכנות מערכות – מטלה 3

יוליה מושן – 319565610

גיל פסי – 206500936

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

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?

.1

Pre-condition: מטבע ההטלה פנוי.

Post-condition: הודפסה הודעת הטלה או הודעת המתנה.

ערך ההטלה (1 עבור אמת - ניצחון או 0 עבור שקר - הפסד),

מטבע ההטלה פנוי.

Invariant: תהליכון אחד לכל היותר יכול להחזיק במטבע.

2. אנו עלולים להיתקל בבעיית Deadlock , כל אחד מהתהליכונים ממתין לתהליכון השני להפוך לפלי.

לדוגמה :

- תהליכון א' מטיל את המטבע ומקבל עץ. הוא בודק את תהליכון ב'. תהליכון ב' עדיין לא הטיל כלל את המטבע ולכן ודאי לא יכל לקבל את התוצאה – פלי . לכן תהליכון א' ממתין. נלקח זמן מעבד.

- תהליכון ב' מטיל את המטבע ומקבל גם הוא תוצאת עץ. הוא נתקל בבעיה זהה, גם מקבילו – תהליכון א' לא זכה בפלי. לפיכך גם תהליכון ב' נכנס למצב המתנה . כעת כל כלל השחקנים בתוכנית נמצאים במצב במתנה לצד השני ולא מתקדמים – Deadlock .

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 Correctness , יכול היווצר המצב הבא:

- השופט הופך את המטבע ללא זמין.נלקח זמן מעבד.
- תהליכון א' מנסה לגשת למטבע ללא הצלחה. נכנס למצב המתנה.
- השופט הופך את המטבע לזמין ומעיר את כלל התהליכונים. לפני שאחד מהם מספיק להתעורר הוא נועל שוב את המטבע. כלומר לשאר השחקנים אסור להשתמש במטבע.
- תהליכון א' מתעורר ולא בודק פעם נוספת את זמינות המטבע (אינו בתוך לולאה). הוא ממשיך לשחק למרות ש coin_available = false .

חשוב לציין שמצב זה לא ייפגע בנתונים תכנית או ישנה או את הקלט הסופי.

אף על פי כן פגענו **באינוריאנטה** – בכל רגע נתון רק תהליכון אחד אוחז במטבע.

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.

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?

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.

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)

בכל אחת מהפעולות שביצענו בדקנו שאכן היא התבצעה בהצלחה . למעט פעולה אחת בשורה 42 :

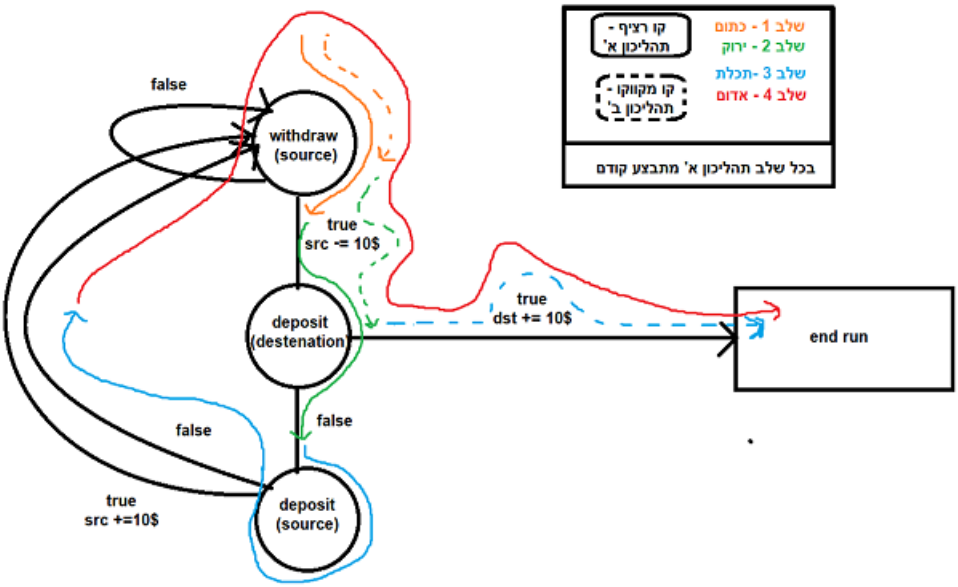
```
35
36      public boolean tryTransfer(BankAccount destinationAccount, double amount) {
37          if (this.withdraw(amount)) {
38              if (destinationAccount.deposit(amount)) {
39                  return true;
40              } else {
41                  // destination account busy, refund source account.
42                  this.deposit(amount);
43              }
44          }
45
46          return false;
47      }
```

המתודה אמנם תחזיר אמת או שקר לגבי הצלחת הפעולה אבל ייעשה דבר בנידון. בצורה כזאת עלול להיווצר מצב נכשלנו בהעברת הכסף לחשבון היעד ולכן ננסה להשיב את הכסף לחשבון המקור. אמנם פעולה זו תיכשל עדיין וחשבון המקור יותר בגירעון.

T1 מעביר מ – Foo ל- Bar

T2 מעביר מ- Bar ל- Foo

	תהליכון	אובייקטים נעולים	פעולות שהתבצעו	חשבון foo	חשבון bar	חשבון המקור	חשבון היעד
1	T1	-	מושך 10 דולר מהמקור	490	500	foo	bar
2	T2	bar	בתוך מתודת משיכה	490	500	bar	foo
3	T1	bar	מנסה להיכנס למתודת הפקדה ונכשל. ממשיך.	490	500	foo	bar
4	T2	foo	מסיים למשוך, עובר למתודת הפקדה	490	490	bar	foo
5	T1	foo	מנסה להפקיד מחדש בחשבון foo ללא הצלחה. ממשיך לאיטרציה הבא.	490	490	foo	Bar
6	T2	foo	מפקיד 10 $ בהצלחה בחשבון foo . מסיים את הריצה.	500	490	bar	Foo
7	T1	-	מבצע את כל ההעברה ללא הפרעה, מסיים ריצה.	490	500	foo	bar



ניתן לראות שבתום הריצה לאחד מהחשבונות היו רק 490 דולר. הדבר סותר את ה- Post condition שהוגדר ולכן קיימת בעיית בטיחות בקוד.