תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

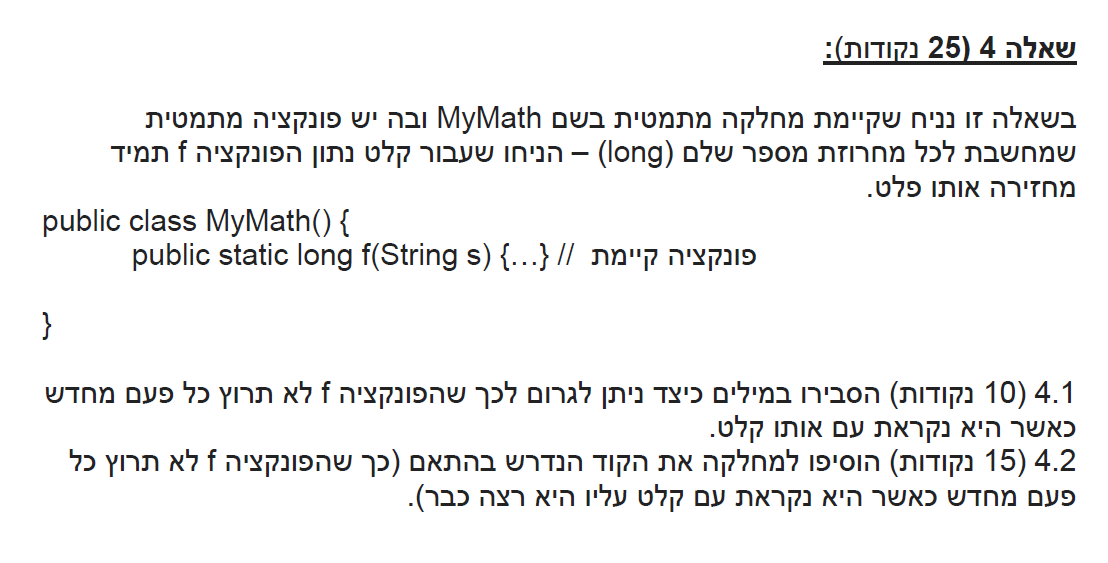
נניח ונוכל להשתמש ב- תהליכונים(לפי מספר הליבות), נחלק את הרשימה ל רשימות בערך באותו גודל.

ניצור תהליכונים ונמפה לכל תהליכון שיפעיל merge sort על תת רשימה כלשהי.

נחכה שכלל התהליכונים יסיימו.

לאחר שכלל התהליכונים יסיימו, נאחד כל שתי רשימות לרשימה אחת ונמיין אותה, נמשיך כך עד שברשימה המאוחדת יהיו כל האיברים מהרשימה המקורית.

**הערה:** בחרתי לפתור את זה עם merge sort כדי שאוכל להשתמש בפונקציה merge כדי להמשיך ולמזג את תתי הרשימות, אך אפשר לבחור כל מיון אחר ולהתאים אותו לצורך.



**סעיף 4.1**

נוסיף למחלקה שדה של HashMap<String, Long> ובכל הפעלה של הפונקציה נבדוק אם הקלט s הוא מפתח בהאשמאפ, אם כן, נחזיר את הערך שלו, אחרת נבצע את החישוב ונשמור בהאשמאפ את s כמפתח ואת התוצאה של החישוב כפלט.

**סעיף 4.2**

Private static HashMap<String, Long> map = new HashMap<>();

בתחילת הפונקציה f

If(MyMath.map.containsKey(s){

Return MyMath.map.get(s);

}

//rest of the function

בסוף הפונקציה

MyMath.map.put(s, out);

return out;

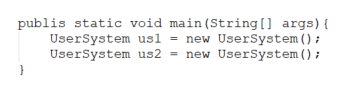
כאשר out הוא התוצאה של החישוב.

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

**סעיף 1:**

הבעיה התכנונית נובעת בתיעוד בראש המחלקה ובאופן המימוש. במימוש נדרש שיהיה אך ורק משתנה אחד אך מאחר והconstructor הוא public, ניתן ליצור כמה אובייקטים כאלה שרוצים.



**סעיף 2:**

נתאים את המחלקה להיות מחלקת סינגלטון.



תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

1. מטרת הקוד היא לשמור בתוך סט של מחרוזות את כל המחרוזות המייצגות מספרים מ-0 עד מיליון.  
   בסוף אמור להיות מודפס Thread map size=1000000
2. מה שבפועל יקרה הוא מאחר ושני התרדים משתמשים בset, ולכן מה שיקרה זה שהם בכל פעם יפריעו אחד לשני בפונקציה add, יווצר race condition בין התרדים וכתוצאה מכך ככל הנראה לא יכנסו לסט כל המחרוזות וגודל הסט לא יהיה 1000000
3. יש צורך לבצע סנכרון בין התרדים בעזרת בלוק synchronized על set.  
   ניצור פונקציית עזר סנכרונית שמקבלת set ומחרוזת ומכניסה את המחרוזת בסט, בכך נהפוך נוכל להשתמש בset.add בצורה סנכרונית.

**הערה כללית למישהו שינסה להריץ את הקוד:**אם תשימו לב, הפלט אף פעם לא יהיה 1,000,000 ולכן אפשר להסיק שמשהו מונע ממנו להגיע לזה, מה שאומר שיש פה racing על אחד המשאבים ויש חוסר סנכרון בקטע קוד.

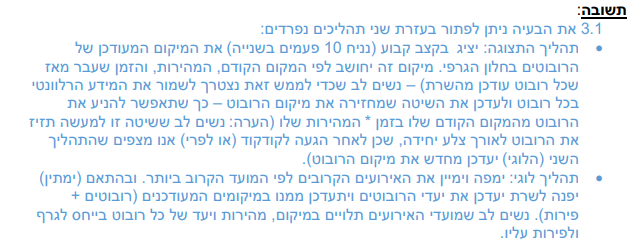


1. הבעיה העקרונית היא שבמידה ונרצה להוסיף סוג חדש של צורה נצטרך להוסיף התנייה נוספת על מנת לטפל בה כמו שצריך, מצב זה סותר את עקרון Open/Close שמ-SOLID.
2. תמונה שמכילה טקסט

   התיאור נוצר באופן אוטומטינשאיר במחלקה רק את המשתנה size ונהפוך את print לפונקציה אבסטרקטית, ניצור 3 מחלקות חדשות שכולן יורשות מShape כאשר כל אחת מהן תדרוס את print.  
   נהפוך את כל המחלקה Shape לאבסטרקטית ובכך נמנע ליצור צורה כללית(שאינה מהסוגים שהוגדרו, וגם נוכל לדרוש שהמחלקות היורשות יממשו את print().
3. תמונה שמכילה טקסט

   התיאור נוצר באופן אוטומטי

**סעיף 3.1**

נפריד את העבודה לשני תהליכים נפרדים:  
תהליך תצוגה: יציג בקצב קבוע(נניח 10 פעמים בשנייה) את המיקום המעודכן של הרובוטים, מיקום זה יחושב לפי המקום הקודם, הזמן והזמן שעבר מאז העדכון האחרון.  
תהליך לוגי: יתעסק בפאן הלוגי ויבצע מה שצריך לעשות ברמה הלוגית, לא רלוונטי לפרט עליו כי לא עשינו את המטלה.  
**תשובה רשמית:**

**סעיף 3.2**

Jsonקיצור של JavaScript Object Notation, המהווה ייצוג טקסטואלי של הערכים השמורים בו, ניתן לשמור כל אובייקט שנרצה בקובץ JSON תוך התאמה לפורמט המתאים מבחינת הסוגים שניתן לשמור בו ופירוק אובייקט מורכב לאובייקטים המרכיבים אותו.  
ל-JSON מבנה ברור המבוסס על סוגריים מאוזנים ומיפוי מפתחות לערכים.  
בJSON ניתן לשמור את סוגי המשתנים הבאים:  
מחרוזות, מספרים, משתנים בוליאניים, מערכים, רשימות, אובייקטים פשוטים ומורכבים(אובייקטים מורכבים בנויים מאובייקטים פשוטים), null.  
**יתרונות:**קריא לעין אנושית – ניתן לקרוא ולהבין אותו בקלות  
תומך במספר שפות תכנות – ניתן להשתמש בו בהרבה שפות תכנות כך שאין תלות בשפה בה יוצרים את הJSON לבין השפה בה קוראים את הJSON

**חסרונות:**

כבד מבחינת משקל – מאחר וJSON הוא ייצוג טקסטואלי של המידע השמור בו, המידע השמור בו מיוצג כמחרוזת, וסך כל המשקל של המחרוזת יהווה את משקל הJSON, כך שככל שהמחרוזת יותר ארוכה, הקובץ יהיה יותר כבד.

Serializable  
ממשק בjava המאפשר המרה של אובייקט לזרם ביטים בינארי אותו ניתן לשמור בקובץ ולקרוא ממנו.

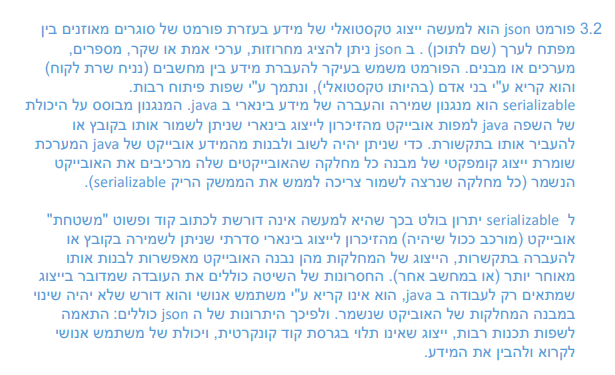
**יתרונות:**

חסכוני במקום – דחיסת האובייקט לרצף בינארי חסכונית מאוד במקום.

**חסרונות:**

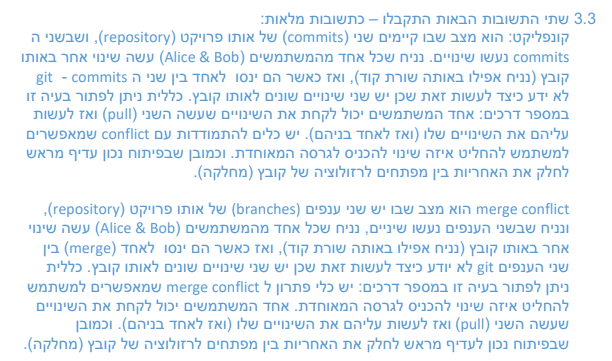
לא קריא לעין אנושית – לא ניתן לצפות במידע השמור בקובץ הבינארי וגם אם כן, לא ניתן לקרוא ולהבין אותו עם עין אנושית.

תומך רק ב-java – ניתן לשימוש אך ורק ב-java, כלומר ניתן ליצור רק ב-java ולקרוא רק ב-java, כך שעל מנת להשתמש חובה להשתמש בשפת תכנות מאוד ספציפית.  
  
**תשובה פורמלית:**

  
**סעיף 3.3**

Conflict הוא מצב בו רוצים לבצע merge בין branch כלשהו לבין הmain branch בrepository וקיימות סתירות בין הקוד ב-branch לבין הקוד ב-main branch. במצב זה יש לבדוק מה הסתירות ולתקן אותן על מנת שיתאימו לשני ה-branch-ים ורק לאחר מכן יהיה ניתן לעשות merge.

**תשובה פורמלית:**

  
  
  
תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

**סעיף 4.1**

המערכת תכיל את המחלקות הבאות:

**בניין:** יכיל אוסף של מעליות, מספר קומה מינימלית ומספר קומה מקסימלית

**מעלית**: תכיל קומה נוכחית, קומת יעד, זמן פתיחת דלתות, זמן סגירת דלתות, זמן מעבר מקומה לקומה(כמה קומות היא עוברת בשניה) ומבנה נתונים להחזקת הקריאות שהוקצו למעלית

**קריאה למעלית:** תכיל את זמן הקריאה, זמן ההקצאה, זמן ההגעה לקומת המקור, זמן ההגעה לקומת היעד, זמן קומת המקור, קומת היעד וסטטוס הקריאה(טרם הוקצאה, טרם הגיעה לקומת מקור, טרם הגיעה לקומת יעד, בוצעה בהצלחה)

**מבנה נתונים:** יאפשר להכניס את הקריאות החדשות באופן שיאפשר ליצור דיליי מינימלי לקריאות אחרות

**אלגוריתם שיבוץ:** מקבל קריאות למעלית בבניין מסוים ומשבץ אותן למעליות השונות בזמן אמת.

**סעיף 4.2**

1. קבל קריאה חדשה
2. הגדר משתנה המייצג את הדיליי המינימלי עבור קריאה
3. הגדר משתנה המייצג את אינדקס המעלית אליה נקצה את הקריאה
4. עבור על כל המעליות הקיימות
   1. חשב למעלית ה- כמה דיליי יגרם בעקבות הקצאת הקריאה למעלית הזאת ושמור את התוצאה במשתנה
   2. אם
5. הקצה את למעלית ה-

**סעיף 4.3**

תחילה נמיין את רשימת הקריאות לפי הזמן בו הקריאה צריכה להשלח.  
נגדיר את גודל הקפיצות שנעשה מבחינת הזמן, כלומר מתי בפועל תשלח כל קריאה על ידי סימון , כלומר פעם בשנייה נריץ את זה. נגדיר את משתנה הזמן שלנו .

ניצור העתק בהעתקה עמוקה של רשימת הקריאות ונסמנו

נרוץ כל עוד callsCopy לא ריק.

נרוץ עד להודעה חדשה(while(true))

אם זמן הקריאה הראשונה ב-callsCopy קטן מ-t

שלח את הקריאה לאלגוריתם ומחק את הקריאה מ-callsCopy

אחרת שבור את הלולאה

הוסף את קפיצת הזמן לזמן של הסימולציה

**תשובה פורמלית לכל השאלה:**

