פרויקט משחק Book Scrabble (צד שרת)

אבן דרך 3

שרת גנרי

נתון לכם הממשק Client Handler שנועד לטיפול בלקוח. הממשק מגדיר מתודת שיחה (handle client) ומתודת סגירה של ה streams.

בהמשך לנלמד בשיעור, עליכם לכתוב את המחלקה MyServer אשר עומדת בתנאים הבאים:

1. בבנאי היא תקבל את ה port ומופע של ClientHandler
2. המתודה start תפעיל את השרת ברקע
    a. בכל שנייה השרת ימתין מחדש לחיבור של לקוח
    b. כאשר לקוח מתחבר הוא יטפל בו באמצעות ה ClientHandler
    c. בסוף הטיפול בלקוח השרת ימתין ללקוח הבא
        i. (השרת מטפל בלקוחות אחד אחרי השני)
3. המתודה close תפסיק את פעולתו של השרת ברקע ותסגור את השרת
    a. אם מתקיימת שיחה עם לקוח ניתן לשיחה זו להסתיים, ואז נסגור את השרת
    b. יש להקפיד על יציאה מסודרת
        i. סגירת ה streams בשיחה עם הלקוח
        ii. סגירת ה socket שמייצג את הלקוח
        iii. יציאה מלולאת ההמתנה ללקוח, כך שתוביל גם לסגירת הת'רד שב היא רצה.
        iv. סגירת המופע של ה ServerSocket
        v. יש לוודא את כל הסגירות ב finally.
            1. אם מופע כלשהו מאותחל ולא סגור אז יש לסגור אותו

בדיקת השרת לבדו נעשית במתודה testServer שב MainTrain, חובה לעבור בדיקה זו כדי להמשיך לשאר הבדיקות.

שרת Book Scrabble

כזכור, באבן הדרך הראשונה יצרנו את התשתית לצד הלקוח עם לוח המשחק. אולם נרצה בעתיד שהשאילתות לגבי חוקיותן של מילים ע"פ המילון יבדקו בצד השרת.

באבן הדרך הקודמת יצרנו את המחלקה Dictionary שהפעילה במתודות query ו challenge מופעים שונים של CacheManager (עם CacheReplacementPolicy ומימושים של LRU ו LFU), BloomFilter ו IOSearcher, ובאמצעותם בדקה בצורה יעילה האם מילה נמצאת במילון או לא.

לכן הגיעה העת ליצור ClientHandler שישתמש ב Dictionary כדי לתת מענה לבקשות הלקוחות השונים.

בסמסטר הבא נתמוך באפשרות שמס' לקוחות יתחברו במקביל, וישחקו ביניהם באמצעות השרת.

אך כבר מעתה אנו מבינים שלא כל לקוח מקבל Dictionary משלו... הרי אם יותר מלקוח אחד משתמש במילון עבור אותם הספרים, אז רצוי שישתמשו באותם המילונים, כלומר באותם המופעים של Dictionary. כך מנגנוני ה cache השונים ינצלו את השאילתות שנעשו ע"י מס' לקוחות.

לכן עליכם לממש את המחלקה DictionaryManager, שבדומה לתבנית Proxy ו Flyweight היא תיצור מילונים חדשים רק ע"פ הצורך, ותספק את תשובותיהם של המילונים המתאימים.

- המחלקה תחזיק מפה בין String ל Dictionary, כאשר כל ספר יקבל Dictionary בלעדי משלו
- בדומה ל Dictionary, גם מחלקה זו תממש את המתודות challenge ו query
  - אלא פרט למילת החיפוש, נקבל גם את רשימת הספרים.
    - String...args כאשר המחרוזת האחרונה היא שאילתת החיפוש
  - אם ספר כלשהו לא קיים במפה, אז נכניס אותו למפה
  - כל שאילתה תחושב עבור כל ספר ונחזיר את התשובה הסופית המתאימה.

הערה: לכאורה ניתן להחזיר תשובה שהמילה קיימת כבר אחרי שאחד הספרים אמר שהמילה קיימת אצלו מבלי לבדוק האם היא קיימת בשאר הספרים. אולם, נרצה בהכרח לעבור על כל ספר בשאילתה כדי לעדכן את ה cache של כל מילון רלוונטי. זה יכול לחסוך זמן עבור שאילתות עתידיות עבור ספרים אלו.

- המתודה getSize תחזיר את מספר הרשומות במילון
- המתודה הסטטית get תחזיר מופע סינגלטוני של DictionaryManager

ה Dictionary Manager נבדק במתודה testDM של ה MainTrain.

כעת כל שנותר הוא לממש ClientHandler בשם BookScrabbleHandler אשר פועל כך:

1. הוא קורא מחרוזת מהלקוח עד לתו ירידת שורה
2. המחרוזת תתחיל ב "Q," עבור query או ב "C," עבור challenge. בהמשך המחרוזת יופיעו מילים מופרדות בפסיק, כאשר הן מציינות את שמות הספרים, פרט למילה האחרונה שמציינת את השאילתה עצמה.
3. באמצעות DictionaryManager נחזיר את התשובה כמחרוזת "true" או "false" ולאחריה תו ירידת שורה.
4. השיחה עם הלקוח תסתיים לאחר שאילתה אחת.

הבדיקה של ה ClientHandler לעיל מתבצעת במתודה testBSCH והוא נבדק כחלק מהשרת.

פרטי ההגשה נמצאים במודול.

בהצלחה!