פרויקט פת"מ תשפ"ה – תרגיל 5

רקע

לאחר תרגילים 1-4, יש לנו למעשה את שכבת המודול בפרויקט. כעת אנו נעבוד על שכבת ה controller. השכבה הזו תורכב למעשה משרת http גנרי, אשר בדומה ל RESTful API, בהינתן פקודות http הוא יבצע חישוב (בשכבת המודול) ויחזיר תשובה בפורמט מבוקש.

ברור לנו שיש כיום ספריות מוכנות לשרת HTTP, אך אנו ננמש חיקוי פשוט כדי להבין טוב יותר כיצד הן עובדות. באמצעות הספרייה נממש את שכבת ה controller באפליקציית הרשת שלנו, ומאוחר יותר את שכבת ה view נציג בדפדפן.

תרגיל 5:

הפעם ניצור package בשם server.

תחילה, נרצה לפענח בקשות http. נעשה זאת במחלקה RequestParser במתודה הסטטית parseRequest אשר בהינתן BufferedReader כמקור קריאה, נפענח את הבקשה ונחזיר אובייקט מסוג RequestInfo שמוגדר כמחלקה פנימית.

כפי שניתן לראות בקוד, במחלקה RequestInfo אנו שומרים את:

- פקודת ה http (למשל get)
- ה URI (כתובת)
- החלקים המרכיבים את ה URI כמערך מחרוזות
- מפת פרמטרים (key, value) שנשלוף מהכתובת
- ומערך של בתים עבור התוכן (content) למשל עבור קובץ מצורף בפקודת post.

למשל, הנה פקודת http (מעורבבת) שמשלבת האלמנטים שידרשו בפרויקט:

```
String request = "GET /api/resource?id=123&name=test HTTP/1.1\n" +
                    "Host: example.com\n" +
                    "Content-Length: 5\n"+
                    "\n" +
                    "filename=\"hello_world.txt\"\n"+
                    "\n" +
                    "hello world!\n"+
                    "\n" ;
```

- הפקודה היא GET
- ה URI הוא /api/resource?id=123&name=test
- החלקים הם "api", "resource"
- הפרמטרים שנתנו ב URI (כחלק מפקודת GET) הם id עם הערך 123 ו name עם הערך test

נשים לב שב header של ההודעה לעיל קיים שדה Content-Length שמציין את גודל התוכן. אם השדה קיים אז הערך אמור לייצג את מספר הבתים של תוכן ההודעה שתגיעה בהמשך. בדוגמה לעיל הערך 5 הוא סתמי, ואף ייתכנו הודעות שהגודל אינו נכון!

אנו נדע שה header הסתיים לאחר שורה ריקה. אחריה יכולים להיות פרמטרים נוספים כמו שם הקובץ hello_world.txt כפי שבד"כ ניתן בפקודת post המכילה קובץ. לאחר שורה ריקה נוספת יתחיל התוכן. במקרה שלנו זה !hello world וירידת שורה. נדע שהשידור הסתיים בהגעה לשורה ריקה (או כאשר ה reader.ready() שלנו מחזיר שקר).

כאשר יש לנו קוד טריוויאלי, מוגדר היטב, ומנותק מהסביבה שבה הוא רץ, זה בדיוק המקום לקצר תהליכי פיתוח ולהיעזר ב generative AI כגון ChatGPT ודומיו. הגדירו לו היטב את המשימה, זכרו שהוא לא תמיד מדייק, וע"י חשיבה ביקורתית בתהליך אינטראקטיבי תוכלו לחלץ קוד יעיל ולחסוך בזמן פיתוח.

הבדיקה במוד האימון טריוויאלית ובמקרה זה גם תהיה זהה במוד הגשה. אולם, המבחן האמתי יגיע כשאר המחלקה הזו תצטרך לפרסר הודעת http אמיתית.

<u>יצירת שרת http גנרי</u>

כדי שהשרת שלנו יהיה גנרי, נרצה להפריד בין האלמנטים שמשתנים לאלו שלא. המנגנון שממתין ללקוח ומטפל בו במקביל ללקוחות אחרים אינו משתנה, ואילו פרוטוקול השיחה שעל פיו השרת והלקוח אמורים לשוחח בהחלט משתנה בין פרויקט לפרויקט.

בנוסף אנו נרצה שלמי שיעשה שימוש בספרייה שלנו יהיה נוח להרכיב את האפליקציה, דהיינו להתאים מענה לכל פקודת http שתגיע.

לכן נגדיר ממשק בשם Servlet שמטרתו להגדיר פונקציונאליות של טיפול בלקוח. כל מופע של Servlet בדרכו שלו יחלץ מידע מ RequestInfo, יפעיל משהו בהתאמה בשכבת המודול, ויכתוב ל output stream שיגיע חזרה ללקוח.

```
public interface Servlet {
    void handle(RequestInfo ri, OutputStream toClient) throws IOException;
    void close() throws IOException;
}
```

את הממשק הזה הניחו ב package בשם servlets שם בהמשך תוסיפו servlets שונים.

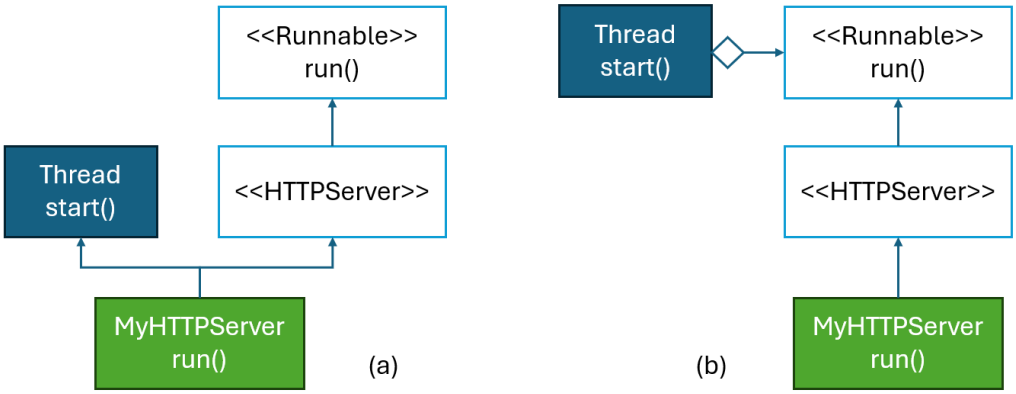
כעת נוכל להגדיר (חזרה ב server package) את הממשק של השרת שלנו:

```
public interface HTTPServer extends Runnable{
    public void addServlet(String httpCommanmd, String uri, Servlet s);
    public void removeServlet(String httpCommanmd, String uri);
    public void start();
    public void close();
}
```

כפי שניתן לראות ניתן להוסיף לו \ להסיר ממנו מופעים של Servlet, וכן לומר לו להתחיל או לסגור אותו. נשים לב שהממשק גם ירש את הממשק של Runnable. זאת משום שדי ברור לנו שהלולאה הראשית של השרת – זו שממתינה כל העת ללקוחות שיתחברו - היא פעולה חוסמת של IO, ולכן עליה לקרות בת'רד משלה. כן, גם עבור תלמידי פת"מ1, זהו המקום בו אין לנו ברירה אלא להריץ את השרת בת'רד ברקע כפי שלמדנו בשיעור.

נרצה לממש את הממשק במחלקה MyHTTPServer. את הלולאה הראשית נממש במתודה run. זה פותח לנו כל מיני אפשרויות לגבי מימוש הממשק. למשל, יכולנו לממש את המתודה start בכוחות עצמנו.

או בדומה לאופציה (a) בתרשים ה UML למטה, את המתודה start פשוט נירש מהמחלקה Thread, וכך בפשטות הקריאה ל start תריץ את הלולאה הראשית ברקע (היא תריץ את run ברקע).



או בדומה לאופציה (b) לאפשר למשתמש הספרייה להכניס מופע של MyHTTPServer כ Runnable לתוך מופע של Thread, ומשם לקרוא ל start. שימושי מאד אם המימוש שלו ל MyHTTPServer היה צריך לרשת משהו נוסף.

בתרגיל זה נשמור על פשטות ונבחר באופציה (a).

המטרה שלנו היא לאפשר שימוש כמו כאן:

```
HTTPServer server=new MyHTTPServer(port:8080,nThreads:5);

server.addServlet(httpCommanmd:"GET", uri:"/publish", new TopicDisplayer());
server.addServlet(httpCommanmd:"POST", uri:"/upload", new ConfLoader());
server.addServlet(httpCommanmd:"GET", uri:"/app/", new HtmlLoader(htmlFolder:"html_files"));

server.start();
System.in.read();
server.close();
```

- בבנאי נוכל לבחור את ה port שעליו השרת יאזין ללקוחות, וכן את הכמות המקסימלית של worker threads שנחזיק פתוחים בעת ובעונה אחת כדי לטפל בלקוחות במקביל.
- באמצעות המתודה addServlet נוכל לשייך מופעים שונים של Servlets לפקודות http מסוג כמו GET, POST וכו', וכן לכתובת ה uri שעבורה נרצה להפעיל את ה servlet.
- נפעיל את השרת ברקע באמצעות המתודה start
- וכשנרצה נסגור אותו. (בדוגמה לעיל זה לאחר enter ב terminal).

לשם כך MyHttpServer תחזיק

- מפות כדי לכרוך בין כתובות ה URI ל Servlets שיתנו להן מענה
  - ***פת"מ2:*** חשוב שהמפות יהיו thread safe
  - בתרגיל נסתפק ב 3 מפות שונות - בהתאמה לפקודות get, post, delete.
  - באמצעות המתודות addServlet ו removeServlet נוסיף \ נסיר servlets למפות אלו
  - ***פת"מ2:*** thread pool (בג'אווה זה executor service) כדי להריץ באמצעותו במקביל את הטיפול בלקוחות - לא יפתח יותר ת'רדים מהערך שניתן בבנאי (nThreads לעיל)
  - ***פת"מ1:*** השרת שלכם בשלב זה יכול פשוט להריץ את הלקוחות בטור (אחר אחרי השני)

אופן הפעולה (מתודה run):

- כל עוד לא ניתנה הוראה לסגור את השרת, הוא ימתין (מחדש) ללקוח במשך שנייה.
- אם לקוח התחבר הטיפול בו יתבצע כמשימה של ה thread pool.
  - נפרסר את הפקדוה באמצעות ה RequestParser
  - ע"פ פקודת ה http וה uri נבחר את ה Servlet המתאים מהמפות ונריץ אותו
    - הבחירה ע"פ ה uri בעל ההתאמה הארוכה ביותר.
    - (ולא התאמה מדויקת)
    - זאת על מנת שאותו ה Servlet יוכל לטפל בבקשות עם פרמטרים שונים שיחלץ מ-ה RequestInfo שקיבל.

ראו למשל את הדוגמה לעיל. כל פקודת get שתתחיל ב publish/ תוביל להפעלה של TopicDisplayer ואילו כל פקודת get שתתחיל ב /app/ תגיע דווקא ל HtmlLoader. דוגמאות:

- GET /app/something/index.html HTTP/1.1
- GET /app/main.html HTTP/1.1
- GET /publish?topic=A&message=5 HTTP/1.1
- GET /publish?topic=B&message=8 HTTP/1.1

הרעיון מאחורי ההתאמה הארוכה ביותר הוא שנוכל ע"פ הצורך לשייך Servlets שונים לכתובות בעלות תחילית דומה, למשל /app/download/ תחזיר פרטים של משתמשים, ואילו כל דבר אחר שמתחיל ב /app/ פשוט יטען קובץ html.

- GET /app/download/user?id=123 HTTP/1.1
- GET /app/index.html HTTP/1.1

<u>אופי הבדיקה:</u>

שימו לב מוד האימון בכוונה אינו מכיל בדיקה של השרת.

עליכם לבדוק את עצמכם באמצעות בדיקה דומה לזו שמתבצעת במוד ההגשה:

- יצירה של Servlet והרשמתו לכתובת uri מסוימת בשרת
  - פעולתו תהיה לחלץ פרמטרים, לבצע עליהם חישוב כלשהו, ולכתוב בחזרה ללקוח את התוצאה.
- וודאו שהפעלת השרת (start) אכן פתחה אך ורק ת'רד אחד נוסף
- צרו לקוח (מופע של Socket) המתחבר לשרת (על localhost וב port שבחרתם)
- תשלחו כתובת שמתאימה להפעלת ה servlet (ראו דוגמאות לעיל)
- זה יוביל לטיפול בבקשה במקביל.
- קראו מה Socket הלקוח את מה שכתב לכם השרת ותוודאו שאכן קיבלתם את התוצאה הרצויה.
- סיגרו את כל המשאבים הפתוחים ואת השרת.
- תמתינו כ 2 שניות ותוודאו שאכן כל הת'רדים נסגרו.

יש להגיש ע"פ הוראות ההגשה במודול, אל תשכחו להעביר את שמות ה package ל test לצורך ההגשה.

בהצלחה!