פרויקט תכנות מתקדם תשפ"ד – תרגיל 1

רקע

אחת הדרכים הטובות ביותר להבין כיצד ספריות שימושיות עובדות היא ע"י כתיבה של חיקוי פשוט שלהן שממחיש את עיקרון פעולתן. בפרויקט זה אנו נממש חיקוי פשוט מספר מנגנונים מורכבים, תרגיל אחר תרגיל, כך שלבסוף יתאחדו לכדי פרויקט אחד שלם. נממש בפרויקט זה תבניות עיצוב וארכיטקטורה, שרת גנרי, וצדדי לקוח שונים. בתרגיל זה נממש את התשתית הבסיסית לשכבת המודל בפרויקט.

ברצוננו לממש מערכת בארכיטקטורת publisher / subscriber. באמצעות ארכיטקטורה זו נוכל לממש גרף חישובי (computational graph) מתוך מטרה לבצע חישובים מורכבים - הבנויים מקודקודי חישוב החיים במקביל זה לזה, כאשר פלטים של קודקוד מסוים מהווים את הקלטים של קודקודים אחרים.

למשל קודקוד אחד קורא אות ממצלמת וידאו ומפרסם תמונה מכווצת. קודקוד אחר נרשם אליו ובכל פעם שמגיע פריים של תמונה, יבצע עיבוד תמונה באמצעות רשת בינה מלאכותית שמגלה בני אדם ומפרסמת מערך נתונים של "השלד" שלהם. קודקוד שלישי ירשם אליו ובכל פעם שמגיע עדכון לשלד הוא ינסה לזהות איזו מחווה ויזואלית מביע האדם בתמונה, וכדומה.

טרמינולוגיה:

Message – הודעה הנושאת מידע רלוונטי כלשהו.

Topic – נושא שאליו ניתן להירשם כדי לקבל הודעות, או לפרסם דרכו הודעות לאחרים.

Agent – סוכן תוכנה שיכול להירשם ל Topics, להגיב להודעות שהתקבלו דרכם באמצעות חישוב כלשהו ופרסום התוצאות ב Topics אחרים.

זכרו, כל Agent יכול להירשם להאזנה ל Topics רבים, ואף לפרסם הודעות ל Topics רבים.

תרגיל 1:

נתחיל במחלקה Message המייצגת הודעה.

- הודעה היא Immutable כלומר אינה ניתנת לשינוי.
  - המשמעות היא שכל השדות שלה הם final
  - כשזה המצב, יש לאתחל את כולם בבנאי,
  - ורצוי שיהיו public, שכן לא ניתן לשנותם.
  - נגדיר את כולם כ public final.
- הודעה צריכה להחזיק מידע, חתימת זמן, וכמה המרות נפוצות. להמחשה נסתפק בשדות הבאים:
  - data - מערך של בתים (תוכן ההודעה)
  - asText כהמרה למחרוזת
  - asDouble כהמרה ל double
  - date – מופע של Date.
- לפיכך נרצה 3 בנאים, המאתחלים בהתאמה מופע של Message בהינתן: מערך של בתים, מחרוזת, או double. עם כל יצירת מופע של הודעה יאותחל מופע חדש של Date הזוכר את זמן יצירתה.
- למעשה מספיק לכתוב בנאי אחד שמקבל פרמטר מסוג מסוים (למשל String) ודואג לעשות המרות לכל השאר, (וכמובן ליצור מופע של Date), ואילו הבנאים האחרים יעשו לו פשוט reuse באמצעות קריאה ל ()this עם ההמרה המתאימה.

- שימו לב שלא כל מידע בהכרח יכול להפוך ל double ועלולה להיזרק חריגה מתאימה. במקרה כזה על ערך ה double להיות שווה ל Double.NaN (המשמעות היא not a number).

נתון לנו הממשק Agent המייצג סוכן:

```
public interface Agent {
    String getName();
    void reset();
    void callback(String topic, Message msg);
    void close();
}
```

כפי שניתן לראות, בדומה ל observer design pattern, סוכן צריך להגיב באמצעות המתודה callback לקבלת הודעה שהגיע מ topic מסוים המצוין ע"י שמו כמחרוזת.

כעת נוכל להגדיר את המחלקה Topic המייצגת "נושא" שדרכו ניתן לפרסם הודעות לכל מי שנרשם להאזין לו. השדות יהיו:

- public final String name; - שם הנושא
- רשימה של סוכנים שנרשמו להאזנה לנושא זה (subs).
- רשימה של סוכנים שעתידים לפרסם לנושא זה (pubs).

המתודות יהיו:

- subscribe() אשר בהינתן סוכן היא תרשום אותו כמאזין (רשימת ה subs).
- unsubscribe() אשר בהינתן סוכן היא תסיר אותו מרשימת המאזינים.
- publish() אשר בהינתן Message היא תפרסם את ההודעה לכל הסוכנים הרשומים להאזנה.
  - כלומר תפעיל אצלם את המתודה callback
- addPublisher() / removePublisher() אשר בהינתן סוכן יוסיפו \ יסירו אותו בהתאמה מרשימת ה pubs.

נדגיש כי הבנאי של Topic צריך להיות בהרשאת package – כלומר ללא הגדרה של public / protected / private, וזאת כדי שרק מחלקות שנמצאות איתו באותו ה package תוכלנה ליצור אותו.

בפרויקט, את כל המחלקות עד כה יש ליצור ב package בשם graph. אולם, בעת ההגשה למערכת הבדיקות יש לעדכן תמיד את ה package של המחלקות ל test לשם פשטות הבדיקה.

כעת נרצה להבטיח שהדרך היחידה ליצור Topics חדשים היא באמצעות המחלקה TopicManager ושגם לזו יהיה מופע אחד ויחיד.

לשם כך נגדיר את המחלקה TopicManagerSingleton.

כאן נעשה שימוש חכם ב concurrency pattern שיאפשר לנו ליצור Singleton שהוא גם thread safe.

תיאור התבנית:

- בתוך המחלקה TopicManagerSingleton ניצור את המחלקה הסטטית TopicManager.
- ל TopicManager יהיה בנאי פרטי
- יהיה לה משתנה פרטי, סטטי וסופי (final) מסוג TopicManager המאותחל להיות מופע חדש של TopicManager. נקרא למשתנה זה instance.
  - נשים לב שמותר לנו להפעיל את הבנאי הפרטי כי אנו בתוך המחלקה.

הערה: אם היינו עוצרים כאן, היתה לנו בעיה שיצרנו את המופע גם מבלי שאף אחד ביקש, בסתירה לעיקרון ה laziness. אולם העובדה שמדובר במחלקה פנימית מזכה אותנו בשני יתרונות, כפי שנראה עוד רגע.

במחלקה החיצונית (TopicManagerSingleton) נממש את המתודה הבאה:

```
    public static TopicManager get(){
        return TopicManager.instance;
    }
```

בנגידות למחלקות החיצוניות שנטענות עם תחילת התוכנית, מחלקות פנימיות נטענות **רק על פי דרישה!** בנוסף, כל מחלקה Java-ית יכולה להיטען לזיכרון **רק פעם אחת!** (ה JVM כבר דאגה לזה). כך שעם הקריאה הראשונה ל get (גם אם יהיו 1000 כאלו באותו הזמן) תהיה "נגיעה" במחלקה TopicManager שתיטען בתורה לזיכרון פעם אחת בלבד! כלומר בצורה שהיא thread safe, ויחד איתה יטען המשתנה הסטטי שלה instance. כלומר הוא נטען רק ע"פ דרישה ולא מראש.

כל זה מבלי להסתבך עם synchronized, משתנים שהם volatile או תבניות כגון double check locking.

מכל מקום בפרויקט נוכל לקבל בבטחה רפרנס למופע היחיד של TopicManager ע"י:

```
        TopicManager tm=TopicManagerSingleton.get();
```

המחלקה TopicManager:

- תחזיק מפה ממחרוזת (שם הנושא) למופע של Topic.
  - רצוי שהמפה תהיה מסוג ConcurrentHashMap כדי שכל פעולה שלה תהיה אטומית.
- תגדיר מתודה בשם getTopic אשר בהינתן שם (מחרוזת), אם קיים כבר מיפוי משם זה למופע קיים של Topic היא תחזיר אותו, אחרת תיצור אותו במפה ותחזיר אותו (בדומה לבנית flyweight). חישבו על thread safety.
- תגדיר מתודה getTopics אשר תחזור פשוט Collection של כל ה Topics במפה.
- המתודה clear() תמחק את הרשומות מהמפה

הגשה:

- עליכם להגיש את המחלקות לעיל ע"פ הוראות ההגשה במודול \ במערכת הבדיקות.
- זכרו שיש לשים את המחלקות תחת package בשם test.
- כדי לעודד אתכם לבצע בדיקות בכוחות עצמכם, תוכלו להגיש כמה פעמים שתרצו למערכת הבדיקות במוד אימון או הגשה, אולם הפלט היחיד שתקבלו הוא האם הבדיקות עברו ללא שגיאות קומפילציה או ריצה. לא תקבלו משוב של טסטים שנכשלו.
- שימו לב שיש להגיש במוד הגשה כדי שההגשה תיחשב.

אופי הבדיקה:

- בדיקת עמידה בהגדרות שונות בתרגיל
- בדיקת ריצה של גרף חישובי עם נתונים אקראיים, נרצה לוודא שהתוצאה הסופית המתקבלת ב Topic התוצאה היא אכן התוצאה הרצויה.

בהצלחה!