**תיאור התוכנית:**

מדובר בתוכנית אשר מדמה תקופה של x ימים בחנות נעליים. בסוף הסימולציה, כלומר כשהתוכנית עוצרת, תוצאות הסימולציה יודפסו למסך (כמה נעליים נקנו בסה"כ, סוגי נעליים שנקנו בכל יום, הזמנות ייצור שנעשו למפעלי נעליים איתם החנות עובדת וכו'). הקלט לתוכנית הינו בצורת קובץ json, שמכיל את המידע הרלוונטי על חנות העליים ועובדיה, על הלקוחות ועל המפעלים. הלקוחות, המפעלים ועובדי החנות מיוצגים כ-threads בתוכנית, והם פועלים במקביל.

מנהל החנות אחראי על פרסום הנחות לנעליים בזמנים מסוימים, ועל הזמנת נעליים ממפעלי הנעליים במקרה שקיבל פנייה ממוכר בחנות לנעל מסוימת שאזלה במלאי ושקיים לקוח המעוניין בה. מוכרי החנות אחראים על טיפול בלקוחות, ועל פנייה למנהל על מנת שיזמין נעליים שאזלו כפי שתואר קודם. הלקוחות הם צרכנים אשר מעוניינים בנעליים מסוימות בחנות. המפעלים הם אלה האחראים על ייצור נעליים במקרה שקיבלו בקשה להזמנה ממנהל חנות הנעליים.

**הוראות הפעלה:**

ההוראות מתייחסות ל- Windows. בנוסף, יש לוודא ש- Maven מותקן במחשב.

(1). יש להכין קובץ json שיכיל את המידע הרלוונטי לסימולציה, כפי שיתואר כעת. תחילה, יש לפתוח קובץ טקסט ולשמור אותו תחת שם מסוים ללא רווחים, בסיומת json. את הקובץ יש לכתוב לפי הפורמט המופיע במסמך jsonInputFormat.pdf (הסבר עליו בהמשך). **הפורמט מעט מסובך להבנה, ולכן ייתכן שעדיף לעבור על הקלט לדוגמה בקובץ jsonInputExample במקביל להסבר שיצורף מיד.**

הסבר לפורמט:

- **Initial-storage-** פריטי הנעליים אשר זמינים בחנות בתחילת הסימולציה. יש להחליף כל $shoe_i$ בסוג הנעל, וכל $amount_i$ בכמות שלה בחנות.
- **Services::time-** בסימולציה זו נתייחס לימים כאל טיקים בשעון, ונקבע כמה זמן עובר בין טיק לטיק. <speed> יוחלף בזמן החולף בין טיק לטיק (במילישניות). duration יוחלף במספר הטיקים אשר מעוניינים שיתבצעו בסימולציה.
- **Services::manager::discountSchedule** – מנהל החנות אחראי בין היתר על פרסום הנחות בטיקים כלשהם לסוגי נעליים כלשהם. אם מנהל החנות רוצה לפרסם הנחה על m סוגי נעליים, יש להחליף כל $d\_shoe_i$ בסוג הנעל המבוקשת, כל $d\_amount_i$ בכמות של $d\_shoe_i$ עליה תינתן הנחה, וכל $tick_i$ במספר הטיק שבו המנהל מעוניין לפרסם את ההנחה. ייתכן שהמנהל לא יפרסם הנחות בכלל. במצב כזה יש לרשום: "discountSchedule":[].
- **Services::factories-** יש להחליף את <num_of_factories> בכמות מפעלי הנעליים איתם החנות עובדת.
- **Services::sellers-** יש להחליף את <num_of_sellers> במספר מוכרי החנות.
- **Services::customers-** יש להזין c לקוחות. לכל אחד יש שם (name), wishList (רשימת נעליים שמעוניין לקנות במבצע בלבד) ו-purchaseSchedule (רשימת נעליים שמעוניין לקנות בטיקים כלשהם. את נעליים אלה יקנה גם אם במבצע וגם אם לא, אך כמובן יעדיף לקנות בהנחה במקרה שיש כזאת). יש להחליף כל <$name_i$> בשם הלקוח, כל <$w\_shoe_{i_r}$> בנעליים אותן מעוניין הלקוח ה-i לרכוש בהנחה בלבד וכל <$p\_shoe_{i_y}$> בנעליים אותן מעוניין לקנות בטיק שהינו <$tick_{i_y}$>.

  ייתכן שלא יהיו לקוחות בכלל (ואז נזין "customers"=[]), ש-wishList של לקוח יהיה ריק (נזין "wishList"=[]) וש-purchaseSchedule של לקוח יהיה ריק (נזין "purchaseSchedule"=[]).

(2). הורד את התיקייה shoe-store למחשב, ושמור בתוכה את הקובץ שיצרת ב-(2).

(3). פתח טרמינל בתיקייה shoe-store, והזן את הפקודה הבאה למטרת קומפילציה: mvn compile .

לאחר מכן, הזן את הפקודה הבאה כדי להריץ את הסימולציה, כאשר <filename> יוחלף בשם הקובץ שבחרת עבורו:

```
mvn exec:java -Dexec.mainClass:"bgu.spl.app.ShoeStoreRunner" -Dexec.args="<filename>.json"
```

(4). כעת יודפסו המאורעות אשר קורים בכל טיק. בסיום, יודפס מידע על הנעליים שנשארו בחנות ועל הרכישות שהתבצעו.

**הערה לסיום:**

ניתן להריץ את הקובץ largeExample.json (ראה (4) ) כדי לראות דוגמת פלט על קלט גדול יותר, אשר בו קיים גם שימוש גדול יותר במקביליות.