מבוא לתכנות – עבודה 5

תכנות מונחה עצמים

מועד פרסום: 7.1.18

מועד אחרון להגשה: 18.01.18

מתרגל אחראי: אלעד מרקו

הוראות מקדימות

בעבודה זו תממשו את מבנה הנתונים רשימה מקושרת. כמו כן תתרגלו שימוש בהורשה, ממשקים ופולימורפיזם.

1. העבודה כתובה בלשון זכר (מטעמי נוחיות) אך פונה לשני המינים.
2. קראו את העבודה במלואה לפני שאתם מתחילים לפתור אותה.
3. ניתן להגיש את העבודה **בזוגות**. מי שמעוניין יכול להגיש את עבודתו **לבד**.
4. יש להוסיף את הפתרונות שלכם לקבצי השלד שסיפקנו לכם ולהגיש אותם בקובץ zip יחיד. **אין להגיש קבצים נוספים** ואין ליצור תיקיות מיותרות (כמו אלו הנוצרות לדוגמה ביצירת package חדש ב-eclipse).
5. שם הקובץ המכווץ יהיה מספר תעודת/תעודות הזהות של הסטודנט/ים. לדוגמה, 123456789.zip או 123456789_987654321.zip.
6. **אין לשנות את שמות המחלקות או השיטות אותם קיבלתם!**
7. **בדומה לעבודה הקודמת, יש לשמור בעבודה זו על עיקרון ההכמסה!**
8. את קובץ ה-zip יש להגיש דרך מערכת ההגשה: https://subsys.ise.bgu.ac.il/submission/login.aspx
9. ניתן להגיש את הקובץ מספר פעמים עד לתאריך ההגשה, ההגשה האחרונה היא זו שתיבדק.
10. העבודה תיבדק באופן אוטומטי לפי הפלט אשר התוכניות שלכם תדפיס למסך. לכן, אנא הדפיסו אך ורק את הפלט הנדרש, הקפידו על ההוראות ובצעו אותן במדויק. כל הדפסה אשר אינה עונה באופן מדויק לדרישות המופיעות בעבודה (כולל שורות, רווחים, סימני פיסוק, או כל תו אחר מיותר, חסר, או מופיע בסדר שונה מהנדרש), תגרור פגיעה משמעותית בציון (80 נקודות).
11. סגנון כתיבת הקוד ייבדק באופן ידני (20 נקודות). יש להקפיד על כתיבת קוד ברור, מתן שמות משמעותיים למשתנים, הזחות (אינדנטציה), והוספת הערות בקוד המסבירות את תפקידם של מקטעי הקוד השונים שלדעתכם אינם אינטואיטיביים להבנה במבט ראשון ובתחילת כל פונקציה שאתם כותבים.
12. במידה ואינכם בטוחים מהו הפירוש המדויק להוראה מסוימת או אם יש לכם שאלה אחרת הקשורה לתוכן העבודה, אנא היעזרו בפורום או בשעות הקבלה של האחראית על העבודה. בכל בעיה אישית הקשורה בעבודה (מילואים, אשפוז וכו') אנא צרו את הפניה המתאימה למייל הקורס (ise.intro.181@gmail.com).
13. ניתן לפתור את העבודה במעבדות המחשבים של האוניברסיטה או במחשבים האישיים. שימו לב! העבודות יבדקו במחשבי האוניברסיטה ולכן חובה לוודא שהעבודה עובדת כראוי על מחשבי האוניברסיטה!
14. **אין להשתמש בחומר שאינו נלמד** בכיתה עד לרגע פרסום התרגיל.
15. העבודות יבדקו למציאת העתקות. סטודנטים שיתפסו מעתיקים ישלח לוועדת משמעת. לכן אין לשתף קוד!
16. בעבודה זו נמשש את המשמש `Comparator`. ניתן ללמוד על הממשק בלינק הבא: https://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html

בעבודה זו נממש מערכת לניהול הזמנות בחנות למשחקים וצעצועים.

על המערכת

בתרגיל זה שלושה אובייקטים עיקריים: **לקוח**, **מוצר**, ו**מערכת לניהול הזמנות של לקוחות**, אשר ייוצגו ע"י שלושת המחלקות `Product`, `Client` ו- `ClientsOrdersManagementSystem` בהתאמה.
בשלושת המשימות הבאות, יתוארו כל אחת משלושת המחלקות העיקריות. מחלקות אלו יורחבו בהמשך, ובנוסף נגדיר מחלקות עזר לביצוע המשימות.

הנחיות למשימות בחלק זה:

- לא ניתן לשנות את חתימות השיטות הציבוריות במחלקה.
- פרט לשיטות אותן אתם נדרשים להשלים ,מותר להוסיף שיטות עזר **פרטיות**.
- בכל רשימה בעלת N איברים ,האינדקסים יהיו בין 0 ל N-1.

המערכת עובדת באופן הבא: חלק מהמוצרים נמצאים פיזית בחנות, וחלק מהמוצרים נמצאים במחסן.
ללקוחות ניתנת האפשרות להזמין מוצרים מהחנות או מהמחסן:

- עבור מוצרים שנמצאים בחנות אין דמי משלוח (משלוח חינם).
- עבור מוצרים הנמצאים במחסן דמי המשלוח נקבעים בהתאם לגודל המוצר.

המוצרים והלקוחות ינוהלו באמצעות רשימות מקושרות (Linked List), כמפורט להלן.

משימה 1: מימוש רשימה מקושרת

במשימה זו עליכם להשלים את מימוש המחלקות `Link` ו- `LinkedList` המייצגות חוליה ואת מבנה הנתונים רשימה מקושרת.

המחלקה Link

מחלקה זו מייצגת חוליה בודדת, מהן מורכבת רשימה מקושרת. המחלקה `Link` מוגדרת בקובץ Link.java אותו קיבלתם כשלד, כאשר רוב הקוד של מחלקה זו כבר ממומש.

שדות המחלקה הנתונים:

- `Object data` - תוכן החוליה (המידע המאוחסן בחוליה).
  שימו לב: בעבודה זו חוליה יכולה להכיל `data` מאחד מהטיפוסים: `Client`, `VIPClient` (יפורט בהמשך) או מחלקה כלשהי המרחיבה את `Product` (יפורט בהמשך).
- `Link next` - מצביע לחוליה הבאה בשרשרת.

כמו כן, נתונים לכם הבנאים והשיטות הבאים:

1. `public Link(Object data)` – בנאי ליצירת חוליה חדשה. הבנאי מקבל את תוכן החוליה שעליה להחזיק, ושומר אותו ב-`data`. בבנאי זה ההפניה לחוליה הבאה מאותחלת ב-`null` (`next` יאותחל ב-`null`).
2. `Public Link (Link link)` – בנאי מעתיק. הבנאי מקבל חוליה ויוצרת עותק חדש בעל state זהה. כפי שהוגדר, שדה ה `data` הינו `Client`, `VIPClient` או מחלקה כלשהי המרחיבה את `Product`.
3. `public Link(Object data, Link next)` – בנאי המקבל את המידע שאותו החוליה צריכה לאחסן ומצביע לחוליה הבאה. הבנאי מאתחל את השדות בהתאם.
4. `public Object getData()` – שיטה המחזירה את המידע המאוחסן בחוליה.
5. `public Link getNext()` – השיטה מחזירה את ההפניה לחוליה הבאה.
6. `public void setNext(Link next)` – השיטה מקבלת הפניה לחוליה כפרמטר ומציבה אותו בשדה `next`.
7. `public Object setData(Object data)` – השיטה מקבלת הפניה לאובייקט `data`, מעדנמת את המידע המאוחסן בחוליה להיות `data` ומחזירה את המידע שאוחסן בחוליה טרם השינוי.
8. `public String toString()` – השיטה מחזירה מחרוזת המייצגת את העצם השמור בחוליה.

בנוסף לשיטות הנתונות, עליכם **להשלים** את השיטה הבאה:
`public boolean equals(Object other)` - השיטה מגדירה שוויון בין חוליות. שתי חוליות יוגדרו כשוות במידה והמידע אותו הן שומרות שווה.

המחלקה: LinkedList

מחלקה זו מייצגת רשימה מקושרת ומממשת (implements) את הממשק `List` (המצורף לעבודה). נציין כי הממשק `List` אינו זהה לזה שראינו בכיתה. המחלקה מוגדרת בקובץ LinkedList.java אותו קיבלתם כשלד.

המחלקה `LinkedList` מייצגת רשימה מקושרת, באופן שאינו מאפשר אחסון של הערך `null` בתור `data` של חוליה (מיוחס למימוש המחלקה `LinkedList`).
שימו לב כי השיטות `add(Object element)` ו-`add(int index, Object element)` כבר ממומשות באופן כזה.

שדות המחלקה:

- `Link head` – שדה המצביע לחוליה הראשונה בשרשרת.
- `Link tail` - שדה המצביע לחוליה האחרונה בשרשרת.

כמו כן, נתונים הבנאי והשיטות הבאים:

1. `public LinkedList()` - בנאי שאינו מקבל פרמטרים ומאתחל את המצביעים לחוליה הראשונה והאחרונה בערך `null`.
2. `Public LinkedList(LinkedList list)` – בנאי מעתיק. מקבל רשימה ויוצר עותק חדש שזהה לרשימה שהתקבלה.
3. `public void add(Object element)` - השיטה מקבלת איבר לאחסון, ומוסיפה אותו לחוליה חדשה בסוף הרשימה. במידה ו- `element` הוא `null` על השיטה לזרוק חריגה מסוג `NullPointerException`.
4. `public void add(int index, Object element)` - השיטה מקבלת איבר לאחסון ומוסיפה אותו לרשימה כחוליה חדשה במקום ה – `index`. אם `index` שלילי או גדול מגודל הרשימה לזרוק חריגה מסוג `IndexOutOfBoundsException`. אם `element` הוא `null` יש לזרוק חריגה מסוג `NullPointerException`.

בנוסף, עליכם **להשלים** במחלקה את השיטות הבאות:

1. `public boolean isEmpty ()` - השיטה אינה מקבלת פרמטרים ומחזירה את הערך `true` אם הרשימה ריקה ואת הערך `false` אחרת.
2. `public Object get(int index)` - השיטה מקבלת כפרמטר `index` ומחזירה את המידע של החוליה הנמצאת במיקום זה בשרשרת (כאשר האיבר הראשון ברשימה מוגדר כמיקום האפס). אם `index` אינו חוקי יש לזרוק חריגה מסוג `IndexOutOfBoundsException`.
3. `public int size()` – שיטה המחזירה את גודל הרשימה (מספר החוליות ברשימה).
4. `public boolean contains(Object element)` - השיטה מקבלת איבר ובודקת האם הוא שמור ברשימה. מותר חזרות, כלומר, יכולות להיות שתי חוליות בעלות data שווה.
5. `public Object set(int index, Object element)` - השיטה מקבלת `index` ו-`element` ומעדכנת את המידע השמור בחוליה הנמצאת במיקום `index` ברשימה להיות `element`. השיטה מחזירה את המידע שאוחסן טרם השינוי. אם `index` שהתקבל שלילי או חורג מגבולות הרשימה יש לזרוק חריגה מסוג `IndexOutOfBoundsException`. אם `element` הוא `null` יש לזרוק חריגה מסוג`NullPointerException`.
6. `public String toString()` – השיטה מחזירה מחרוזת המייגת שמייצגת רשימה בפורמט הבא: "List: $data_0$, $data_1$, …, $data_{n-1}$" כאשר $data_i$ הוא המידע המאוחן בחוליה עם אינדקס `i`. (תחשבו איך להדפיס $data_i$ עבור i כלשהו). אחרי כל data יש לרדת שורה.
7. `public boolean equals(Object other)` – השיטה מגדירה שוויון בין רשימות משורשרות כשוויון בין החוליות שלהן לפי הסדר. כלומר, השוואת תוכן החוליות הראשונות מכל רשימה, השוואת השנייה לשנייה וכן הלאה. השיטה מקבלת משתנה מטיפוס `Object` ומחזירה`true` אם הוא מצביע לרשימה השווה לרשימה עליה הופעלה השיטה equals או `false` אחרת.

8. `public void sortBy(Comparator comp)` – השיטה מקבלת אובייקט מטיפוס `Comparator` המגדיר יחס סדר בין זוג איברים ברשימה, **וממיינת** את הרשימה בהתאם ליחס סדר זה.
שימו לב: על המיון להתבצע ללא יצירה של רשימה חדשה או חוליות חדשות, אלא רק בעזרת השיטות של המחלקה `Link`. אם ה- `Comparator` שהתקבל הוא `null` יש לזרוק חריגה מסוג `NullPointerException`.
המיון שעליכם לממש בשיטה זו הוא מיון בחירה (selection sort) שראיתם בהרצאות.
עליכם להתאים את מיון הבחירה שנלמד עבור המערך למיון של רשימה מקושרת.

משימה 2: ייצוג של מוצרים במערכת לניהול הזמנות

המחלקה Product

במערכת שלנו, מוצרים מיוצגים על ידי המחלקה `Product` ומחלקות המרחיבות אותה.
בסך הכל מוגדרים חמישה טיפוסים (מחלקות) בהתאם להיררכיה הבאה:

* מוצר `Product` – מחלקה אבסטרקטית.
  - מוצר הנמצא במחסן ונפחו קטן – `ProductInStorageSmall`.
  - מוצר הנמצא במחסן ונפחו בינוני – `ProductInStorageMedium`.
  - מוצר הנמצא במחסן ונפחו גדול – `ProductInStorageLarge`.
  - מוצר הנמצא בחנות – `ProductInStore`.

לדוגמה: **`ProductInStorage`** מרחיב את **`Product`**.
כל סוגי המוצרים **דומים מאוד** בהתנהגותם, **ונבדלים** רק באופן שבו מחושבת עלות המשלוח בכל טיפוס (משימה 3).

כל מחלקה תיכתב בקובץ Java בעל אותו שם.
לכל סוגי המוצרים ישנן שלוש תכונות המגדירות אותם: שם מוצר, מספר סידורי ומחיר. לכל מוצר ישנו מספר סידורי יחודי (שם המוצר לא חייב להיות ייחודי).
במחלקה `Product` נייצג תכונות אלו באמצעות השדות הפרטיים הבאים:

* נייצג שם המוצר באמצעות משתנה מטיפוס `String`.
* נייצג מספר סידורי באמצעות מספר שלם `int`.
* נייצג מחיר באמצעות מספר מטיפוס `double`.

1. השלימו במחלקה `Product` את הבנאי `public Product(String name, int serialNum, double price)` המאתחל את השדות המתאימים. על הבנאי לבדוק כי הקלט חוקי המוגדר באופן הבא:
   - הפרמטר `name` חוקי רק אם הוא כולל מחרוזת חוקית שאורכה ≥ 0.
   - הפרמטרים `number` ו- `price` חוקיים עבור ערכים > 0.

עבר כל ערך אחר על הבנאי לזרוק חריגה מטיפוס `IllegalArgumentException`.

2. השלימו את הבנאים במחלקות המרחיבות את המחלקה האבסטרקטית Product.

השלימו את השיטות הבאות במחלקה `Product`:
3. `public String getProductName()` – שיטה המחזירה את שם המוצר.
4. `public int getProductSerialNumber()` – שיטה המחזירה את המספר הסידורי של המוצר.
5. `public double getProductPrice()` – שיטה המחזירה את המחיר של המוצר.
6. `public String toString()` – השיטה מחזירה מחרוזת המייצגת את המוצר בפורמט הבא: `"Product: name, serialNum, price"`.

7. `public boolean equals(Object other)` - השיטה מגדירה שוויון בין מוצרים ,כשוויון בין המספרים הסידוריים שלהם. השיטה מקבלת משתנה מטיפוס `Object` ומחזירה `true` אם המשתנה מצביע למוצר השווה למוצר עליו הופעלה השיטה `equals`.

המחלקה Client

במערכת שלנו ישנם שני סוגים לקוחות: לקוח רגיל המיוצג ע"י המחלקה `Client`, ולקוח VIP המיוצג ע"י המחלקה `VIPClient` המרחיבה את `Client`. שני הטיפוסים מוגדרים בקבצים Client.java ו- VIPClient.java אותם קיבלתם. לשני סוגי הלקוחות התנהגות דומה, אולם לקוחות VIP מקבלים הנחות בעלות דמי המשלוח עבור מוצרים המוזמנים מהמחסן.

נגדיר לקוח לקוח ע"י ארבעת התכונות הבאות: שם פרטי, שם משפחה, מספר תעודת הזהות ורשימת המוצרים אותם הוא מעוניין להזמין. לכל לקוח מספר תעודת זהות **ייחודי** (השם אינו חייב להיות ייחודי). שימו לב, מספר המוצרים אותם הלקוח רוצה להזמין **אינו מוגבל**.
כדי לממש תכונות אלו נגדיר:

* נייצג שם פרטי באמצעות `String`.
* נייצג שם המשפחה באמצעות `String`.
* נייצג מספר תעודת זהות באמצעות מספר שלם `int`.
* רשימת המוצרים אותם מעוניין הלקוח להזמין תיוצג באמצעות רשימה מקושרת מהסעיפים הקודמים.

1. השלימו במחלקה `Client` את הבנאי הבא: `public Client(String firstName, String lastName, int id)`, המקבל שם פרטי, שם משפחה ותעודת זהות ומאתחל את השדות המתאימים, רשימת המוצרים תאותחל לרשימה ריקה. על הבנאי לבדוק כי הקלט חוקי: הפרמטרים `firstName` ו- `lastName` מחרוזות חוקיות שאורכן ≤ 0 ומספר הזהות > 0.. אם אחד מהתנאים הללו לא מתקיים על הבנאי לזרוק חריגה מטיפוס `IllegalArgumentException` .
2. השלימו במחלקה `VIPClient` בנאי המקבל שם פרטי ,שם משפחה ומספר זהות.

השלימו במחלקה `Client` את השיטות הבאות:

3. `public String getLastName()` - השיטה מחזירה את שם המשפחה של הלקוח.
4. `public String getFirstName()` - השיטה מחזירה את השם הפרטי של הלקוח.
5. `public String getId()` - השיטה מחזירה את מספר תעודת הזהות של הלקוח.
6. `public LinkedList getProducts()` – השיטה מחזירה את רשימת המוצרים שהלקוח מעוניין להזמין.
7. `public boolean isInterestedIn(Product product)` – השיטה מקבלת משתנה מטיפוס `Product` ומחזירה `true` אם המוצר מופיע ברשימת המוצרים שהלקוח מעוניין להזמין. ניתן להניח כי קלט שונה מ – `null`.
8. `public boolean addProduct(Product product)` - השיטה מקבלת כפרמטר ערך מטיפוס `Product` ומוסיפה אותו אל רשימת המוצרים של הלקוח בתנאי שהמוצר אינו מופיע כבר ברשימת המוצרים. השיטה מחזירה true אם המוצר לא הופיע ברשימה קודם לכן (ומוסיפה את המוצר), `false` אחרת. ניתן להניח כי הקלט שונה מ – `null`.
9. `public double calculateTotalPriceOfProducts()` – השיטה מחזירה את המחיר הכולל של המוצרים ברשימת ההזמנות של הלקוח **על סמך מחירי המוצרים בלבד**.
10. `public String toString()` – השיטה מחזירה מחרוזת המייצגת את מצב הלקוח. בפורמט הבא: `"Client: name, serialNum, price, \nProducts"` (\n מסמל ירידת שורה).

11. `public boolean equals(Object other)` - השיטה מגדירה שוויון בין לקוחות כשוויון בין מספרי הזהות שלהם. השיטה מקבלת משתנה מטיפוס `Object` ומחזירה `true` אם המשתנה מצביע ללקוח השווה ללקוח המפעיל את השיטה ו-`false` אחרת.

המחלקה ClientNameComparator:

יש להגדיר את הטיפוס `ClientNameComparator` בקובץ ClientNameComparator.java. **על המחלקה לממש את הממשק `Comparator` המצורף לעבודה.**

השוואה בין שני לקוחות תתבצע על פי הסדר הלקסיקוגרפי של **שמות המשפחה** .אם שמות המשפחה של הלקוחות זהים סדרם יקבע על פי הסדר הלקסיקוגרפי של **שמותיהם הפרטיים**.

- <u>הבהרה</u>: ניתן להניח כי אין שני לקוחות עם שם פרטי ושם משפחה זהים.

<u>הבהרה</u>: בעבודה זו <u>הסדר הלקסיקוגרפי</u> של מחרוזות מוגדר על ידי השיטה `compareTo` של המחלקה `String`.

דוגמה:

```
Comparator nameComp = new ClientNameComparator();
Client yossi = new Client("Yossi", "Cohen", 1);
Client aviC = new Client("Avi", "Cohen", 2);
Client aviA = new Client("Avi", "Avraham", 3);
System.out.println(nameComp.compare(aviA, yossi) < 0);   // returns
true
System.out.println(nameComp.compare(yossi, aviC) > 0);   // returns
true
System.out.println(nameComp.compare(aviC, yossi) > 0);   // returns
false
```

אם ה-Instance Type של האובייקטים המתקבלים כפרמטרים אינו `Client` (או יורש מ - `Client`) יש לזרוק חריגה מסוג `ClassCastException`.

המחלקה ClientTotalProductsPriceComparator:

יש להגדיר את הטיפוס `ClientTotalProductsPriceComparator` בקובץ ClientTotalProductsPriceComparator.java.

על המחלקה לממש את הממשק `Comparator` המצורף לעבודה.
השוואה בין שני לקוחות תתבצע על בסיס המחיר הכולל של המוצרים ברשימת ההזמנות שלהם (ללא הוספת דמי המשלוח שיפורטו בהמשך) **בסדר הפוך**. כלומר, לקוח המזמין בסכום גבוה יותר יחשב כקטן יותר בהשוואה.
נגדיר זאת באופן הזה מכיוון שבמיון רשימת הלקוחות נרצה לקבל סדר הפוך: כלומר, כאשר נמיין את רשימת הלקוחות ע"י השיטה `sortBy` של המחלקה `LinkedList`, לקוחות בעלי המוצרים בסכומים הגבוהים יותר יופיעו לפני לקוחות בעלי המוצרים בסכומים הנמוכים.
דוגמה:

```
Comparator totalPriceComp = new ClientTotalProductsPriceComparator();
Product domino = new ProductInStore ("Domino", 1, 150);
Product mastermind= new ProductInStore ("Mastermind", 2, 250);
Client yossi = new Client("Yossi",”Choen”,1);
Client avi = new Client("Avi",”Choen”,2);
yossi.addProduct(domino);
yossi.addProduct(mastermind);
avi.addProduct(domino);
System.out.println(totalPriceComp .compare(yossi,avi) > 0);    //
returns false
System.out.println(totalPriceComp .compare(avi,yossi) > 0);    //
returns true
```

אם ה-Instance Type של האובייקטים המתקבלים כפרמטרים אינו `Client` (או יורש מ - `Client`) יש לזרוק חריגה מסוג `ClassCastException`.

המחלקה ClientProductManagementSystem:

יש להגדיר את הטיפוס `ClientProductManagementSystem`
בקובץ ClientProductManagementSystem.java. על מערכת ניהול הזמנות הלקוחות לאחסן את רשימות המוצרים והלקוחות הקיימים במערכת. מספרי הלקוחות והמוצרים במערכת **אינם מוגבלים**. כדי לממש תכונות אלו, נשתמש בשדות הבאים:

* רשימת הלקוחות הקיימים במערכת: נייצג על ידי רשימה מקושרת מהסעיפים הקודמים.
* רשימת המוצרים הקיימים במערכת: נייצג על ידי רשימה מקושרת מהסעיפים הקודמים.

1. הוסיפו למחלקה את ארבעת השדות.

השלימו במחלקה את הבנאי והשיטות הבאים:

2. `public ClientProductManagementSystem()` – בנאי ללא פרמטרים המאתחל את שדות המחלקה.
3. `public boolean addClient(Client client)` - השיטה מקבלת כפרמטר לקוח ומוסיפה אותו אל רשימת הלקוחות במערכת במידה והוא לא קיים במערכת. על השיטה להחזיר `true` אם ההוספה התבצעה בהצלחה ו- `false` אחרת. ניתן להניח כי הקלט שונה מ- `null`.

4. `public boolean addProduct(Product product)` - השיטה מקבלת כפרמטר ערך מטיפוס `Product` ומוסיפה אותו אל רשימת המוצרים במערכת במידה והוא לא קיים כבר במערכת. על השיטה להחזיר `true` אם ההוספה התבצעה בהצלחה ו- `false` אחרת. ניתן להניח כי קלט שונה מ- `null`.
5. `public boolean addProductToClient(Client client, Product product)` - השיטה מקבלת שני פרמטרים, לקוח ומוצר, ומוסיפה את המוצר לרשימת המוצרים אותם הלקוח מעוניין להזמין. על השיטה לבדוק שהמוצר והלקוח קיימים במערכת, וכן שהמוצר אינו מופיע כבר ברשימת הקניות של הלקוח. על השיטה להחזיר `true` אם ההוספה התבצעה בהצלחה ו-`false` אחרת. ניתן להניח כי הקלט שונה מ - `null`.
6. `public LinkedList getFirstKClients(Comperator comp, int k)` - השיטה מקבלת עצם מטיפוס `Comparator` המייצג יחס סדר בין לקוחות ומספר חיובי `k`. השיטה מחזירה רשימה ממוינת המכילה את `k` הלקוחות הראשונים ברשימה הממוינת לפי יחס הסדר המוגדר על ידי `comp`. בכל מקרה בו לא ניתן לבצעת את הפעולה בהצלחה על השיטה לזרוק חריגה מטיפוס `IllegalArgumentException`.

7. `public int getNumberOfClients()` - שיטה המחזירה את מספר הלקוחות במערכת.
8. `public int getNumberOfProducts()` - שיטה המחזירה את מספר המוצרים במערכת.

משימה 3: הוספת פונקציונליות לביצוע סגירת הזמנה

במשימה זו עליכם לתמוך בביצוע סגירת הזמנה עבור לקוח. חישוב עלות ההזמנה מתבצעת בצורה שונה עבור לקוחות רגילים ולקוחות VIP. לקוחות VIP מקבלים 50% דמי הנחה על עלות דמי המשלוח עבור מוצרים אשר נמצאים במחסן. כמו כן, עבור מוצרים הנמצאים בחנות אין דמי משלוח. לעומת זאת עבור מוצרים המגיעים מהמחסן עלות דמי המשלוח מתשנה בהתאם לגודל המוצר: (על מנת לעגל למטה ניתן להשתמש לדוגמה בפונקציה floor של המחלקה Math)

* עבור מוצרים קטנים עלות דמי המשלוח היא 5% ממחיר המוצר (מעוגל כלפי מטה).
* עבור מוצרים בינוניים עלות דמי המשלוח היא 7% ממחיר המוצר (מעוגל כלפי מטה).
* עבור מוצרים גדולים עלות דמי המשלוח היא 10% ממחיר המוצר (מעוגל כלפי מטה).

על מנת לתמוך בפונקציונליות זו, יש להוסיף למחלקות הבאות את השיטות הבאות:

המחלקה Product והמחלקות המרחיבות אותה

בכל מחלקה תמומש השיטה הבאה:

`public double[] computeFinalPrice()` - השיטה מחזירה מערך מגודל 2, כאשר התא הראשון מכיל את מחיר המוצר והתא השני מכיל את עלות דמי המשלוח.

המחלקה Client והמחלקות המרחיבות אותה

במחלקה `Client` תממשו את השיטות הבאות:

`public double computeFinalProductsPrice()` – השיטה מחזירה את המחיר הכולל של המוצרים ברשימת הקניות של הלקוח.

`public double computeFinalShippingPrice()` – השיטה מחזירה את המחיר הכולל של דמי המשלוח עבור המוצרים שברשימת הקניות של הלקוח.

`public double computeFinalOrderPrice()` – השיטה מחזירה את המחיר הכולל של כלל המוצרים, כולל מחירי המוצרים ודמי המשלוח הכוללים.

במחלקה **`VIPClient`** עליכם לדרוס את השיטה הבאה של **`Client`**:

`public double computeFinalShippingPrice()` - השיטה מחזירה את המחיר הכולל של דמי המשלוח עבור המוצרים שברשימת הקניות של הלקוח עם הנחה של 50%.

המחלקה ClientProductManagementSystem

השלימו את השיטה:

`public double computeFinalOrderPrice(Client client)` - השיטה בודקת האם הלקוח קיים במערכת. אם כן, השיטה מחזירה את המחיר הכולל של רשימת המוצרים ברשימת הקנות של הלקוח (כולל דמי המשלוח), אחרת השיטה תחזיר 0.

המחלקה ClientTotalProductsAndShippingPriceComparator

יש להגדיר את הטיפוס `ClientTotalProductsAndShippingPriceComparator` בקובץ ClientTotalProductsAndShippingPriceComparator.java. **על המחלקה לממש את הממשק `Comparator` המצורף לעבודה.**

השוואה בין שני לקוחות תתבצע על בסיס המחיר הכולל של המוצרים כולל דמי משלוח ברשימת ההזמנות שלהם ב**סדר הפוך** (כפי שהוגדר במחלקה **`ClientTotalProductsPriceComparator`**).

דוגמאות

```
Comparator comp = new ClientTotalProductsAndShippingPriceComparator();

// same product - different client types
Product sudoku = new ProductInStorageSmall("Sudoku", 1, 150);
Client regularClient = new Client("Yossi", ”Cohen”, 1);
Client vipClient = new VIPClient("Avi", ”Cohen”, 2);
System.out.println(comp.compare(regularClient,vipClient)  <  0);     //
returns true (vipClient has a 50% discount on total shipping price -
regularClient pays more in total)

Product  mastermind  =  new  ProductInStorageMedium  ("Mastermind",  2,
250);
Product robot = new ProductInStorageLarge("Spy Robot", 2, 400);
regularClient = new Client("Yossi", ”Cohen”, 1);
vipClient = new VIPClient("Avi", ”Cohen”, 2);
```

```
regularClient.addProduct(sudoku);
regularClient.addProduct(mastermind);
vipClient.addProduct(robot);
// regularClient total shipping = sudoku(150 + 150*0.05[rounded down])
+ mastermind(250 + 250*0.07[rounded down]) = 424
// regularClient total shipping = robot(400 + (400*0.10) * 0.5) = 420
157 +
System.out.println(comp.compare(regularClient, vipClient) < 0);    //
returns true (vipClient total shipping price is 420, regularClient is
424 - regularClient pays more)
```

בהצלחה!