הטכניון – מכון טכנולוגי לישראל
הפקולטה להנדסת תעשיה וניהול
הנדסת תוכנה חורף תשפ"ב

פתרון מבחן מסכם מועד א – הנדסת תוכנה 094219

מרצה: ד"ר אחמד ג'בארה

מתרגל אחראי: מר דביר טויטו

תאריך: 15.02.2022

תעודת זהות: _____

חדר בחינה: _____

הנחיות לנבחנת:

- ❑ מומלץ לקרוא את כל המבחן לפני תחילת הפתרון.
- ❑ חלק מן השאלות מכילות בסופן הערות והכוונות. מומלץ לא להתחיל לפתור שאלה לפני שקראת את כל סעיפיה.
- ❑ משך המבחן: 3 שעות.
- ❑ במבחן יש 7 שאלות. עלייך לענות על כולן.
- ❑ משקל כל שאלה רשום לידה.
- ❑ אין חומר עזר אשר מותר לשימוש.
- ❑ יש לפתור את המבחן בעט כחול או שחור. ניתן להשתמש בעפרון לצורך סרטוט התרשימים בלבד.
- ❑ יש לענות כל השאלות במחברת הבחינה פרט לשאלת ה-UML עליה יש לענות על גבי טופס הבחינה.
- ❑ יש להקפיד על כתב יד ברור.
- ❑ בשאלות בהן נדרש לסרטט תרשים, יש להקפיד על בהירותו ועל כיווני החיצים.
- ❑ חלק מקטעי הקוד עושים שימוש במחלקות סטנדרטיות של Java. ניתן להניח כי בוצע ייבוא כנדרש.
- ❑ ההנחיות במבחן מנוסחות בלשון נקבה מטעמי נוחות בלבד, אך מתייחסות לנבחנים ולנבחנות כאחד.

בהצלחה!

שאלה 1 (8 נקודות)

בהרצאה דנו בבעיית מגדלי האנוי: נתונים שלושה עמודים A, B, C, כאשר על עמוד A ישנן דיסקיות אשר מסודרות אחת על גבי השנייה מן הדיסקית הגדולה ביותר ועד הקטנה ביותר. בבעיה, עלינו להעביר את כל הדיסקיות מעמוד A לעמוד C בעזרת עמוד B ולבנות את מגדל הדיסקיות מחדש, כאשר לא ניתן להניח דיסקית כלשהי על גבי דיסקית אשר קטנה ממנה ובכל צעד ניתן להעביר דיסקית אחת בדיוק.

שאלה זו עוסקת בגרסה של בעיית מגדלי האנוי אשר מכונה Adjacent Pegs (עמודים סמוכים). בגרסה זו מתווסף לבעיה אילוץ נוסף: מותר להעביר דיסקית לעמוד סמוך בלבד. לדוגמה, ניתן להעביר דיסקית מעמוד B לעמוד C או לעמוד A בתנאי שהעברה זו עומדת בתנאי הבעיה המקורית, אך לא ניתן להעביר דיסקית ישירות מעמוד A לעמוד C או ההפך.

כתבי ב-Java פעולה סטטית בעלת הכותרת הבאה:

```
public static void adjacentHanoi(int n, String A, String B, String C)
```

הפעולה מקבלת את מספר הדיסקיות ואת שלושת העמודים ומדפיסה למסך את רצף ההוראות המינימלי אשר נדרש על מנת להעביר את כל הדיסקיות מעמוד A לעמוד C בעזרת עמוד B.

פתרון

```
public static void adjacentHanoi(int n, String A, String B, String C) {
    if (n == 0) {
        return;
    }

    adjacentHanoi(n - 1, A, B, C);
    System.out.println("Move disk from " + A + " To " + B);
    adjacentHanoi(n - 1, C, B, A);
    System.out.println("Move disk from " + B + " To " + C);
    adjacentHanoi(n - 1, A, B, C);
}
```

שאלה 2 (11 נקודות)

נתונה המחלקה Node והפעולה הסטטית hasSomething:

```
class Node {
    private int info;
    private Node next;

    public Node(int info, Node next) {
        this.info = info;
        this.next = next;
    }

    public Node getNext() {
        return next;
    }
}

public static boolean hasSomething(Node list) {
    Node first = list;
    Node second = list;

    while (first != null && second != null && second.getNext() != null) {
        first = first.getNext();
        second = second.getNext().getNext();
        if (first == second) {
            return true;
        }
    }
    return false;
}
```

סעיף א (6 נקודות)

תני דוגמה לרשימה מקושרת עבורה הפעולה hasSomething תחזיר שקר (false) ודוגמה לרשימה מקושרת עבורה הפעולה hasSomething תחזיר אמת (true). על כל אחת משתי הרשימות להכיל לפחות שלושה איברים.

סעיף ב (5 נקודות)

הסבירי במילים מה הפעולה hasSomething מבצעת עבור רשימה מקושרת כלשהי.

פתרון

הפעולה מקבלת רשימה מקושרת ובודקת האם קיים בה מעגל. במידה וכן היא מחזירה אמת ואחרת היא מחזירה שקר.

עבור סעיף א' מספיק לסרטט רשימה מקושרת אחת עם מעגל ואחת בלי מעגל.

שאלה 3 (28 נקודות)

לפנייך קוד התחלתי של המחלקה Set אשר מייצגת קבוצה של מספרים שלמים. כזכור, בקבוצה כל איבר מופיע לכל היותר פעם אחת, ואין חשיבות לסדר בין האיברים.

```
public class Set {
    private int[] values;
    private int index;  // Next index for inserting an element

    public Set(int maxSize) {
        values = new int[maxSize];
        index = 0;
    }
}
```

סעיף א (7 נקודות)

הגדירי במחלקה Set פעולה בשם addElement אשר מקבלת מספר שלם ומוסיפה אותו לקבוצה במידה וניתן. אם האיבר מופיע כבר בקבוצה תיזרק חריגה בלתי מסומנת בשם ElementAlreadyExists. במידה והקבוצה הגיעה לגודלה המקסימלי (לפני הוספת האיבר) תיזרק חריגה בלתי מסומנת בשם SetOverflowException.

שימי לב: חריגות אלו אינן מוגדרות ב-Java ועלייך להגדירן באופן עצמאי כך שניתן יהיה לתפוס את שתיהן על ידי שימוש בבלוק catch **יחיד**.

פתרון

נגדיר מחלקת חריגה אשר ממנה ירשו שתי החריגות החדשות:

```
class SetException extends RuntimeException { }
```

נגדיר את החריגות:

```
class ElementAlreadyExists extends SetException { }

class SetOverflowException extends SetException { }
```

הפעולה המבוקשת:

```
public void addElement(int num) {
    for (int i = 0; i < index; i++) {
        if (values[i] == num) {
            throw new ElementAlreadyExists();
        }
    }

    if (index == values.length) {
        throw new SetOverflowException();
    }

    values[index] = num;
    index++;
}
```

טעויות נפוצות:

- סטודנטים רבים הגדירו את החריגות כך שהן ירשו מן המחלקה Exception ולכן היו מסומנות.
- סטודנטים רבים זרקו את החריגות בתוך בלוק try ותפסו אותן בבלוק catch. זוהי טעות מאחר ועל מי שמשתמש בפעולה לדאוג לתפוס את החריגות במידה ויזרקו.
- סטודנטים רבים בדקו האם הקבוצה מלאה לפני שבדקו האם האיבר שרוצים להוסיף נמצא כבר בקבוצה. לא הורדו על כך נקודות אך מבחינה לוגית יותר נכון לבצע את הבדיקות בסדר הפוך.

<u>סעיף ב (6 נקודות)</u>

ממשי במחלקה Set את הממשק Cloneable ודרסי את הפעולה clone כדי לאפשר העתקה עמוקה של עצמים מן המחלקה. עלייך לדרוס את הפעולה כך שהפעולה הבאה תעבור הידור:

```
public static void doSomething() {
    Set mySet = new Set(17);
    Set myClonedSet = mySet.clone();
}
```

שימי לב כי לא נדרשת המרה בשורה השנייה, וכי שורות הקוד אינן מוקפות בבלוק try.

<u>פתרון</u>

```
public class Set implements Cloneable {
    /* Code in question */

    @Override
    public Set clone() {
        try {
            Set clonedSet = (Set) super.clone();
            clonedSet.values = values.clone();
            return clonedSet;

        } catch (CloneNotSupportedException e) {
            return null;
        }
    }
}
```

שימי לב כי המערך values מכיל ערכים פרימיטיביים, ולכן אין צורך להעתיק את איבריו אחד אחרי השני ומספיק לקרוא לפעולה clone של המערך (אשר באופן כללי מבצעת העתקה רדודה).

סעיף ג (5 נקודות)

דרסי במחלקה Set את הפעולה equals כך ששתי קבוצות יהיו שוות זו לזו אם ורק אם הן מכילות את אותם האיברים, ללא חשיבות לסדר. יש להקפיד על שאר הדרישות אשר קיימות עבור יחס השוויון ופעולת ה-equals.

הערה: בסעיף זה **אין** צורך לדרוס את הפעולה hashCode.

פתרון

נגדיר תחילה פעולת עזר אשר מקבל קבוצה ובודקת האם כל הערכים של הקבוצה הנוכחית נמצאים בקבוצה שהתקבלה:

```
private boolean isSubset(Set otherSet) {
    for (int i = 0; i < index; i++) {
        boolean flag = false;
        for (int j = 0; j < otherSet.index; j++) {
            if (values[i] == otherSet.values[j]) {
                flag = true;
            }
        }
        if (!flag) {
            return false;
        }
    }
    return true;
}
```

הפעולה המבוקשת:

```
@Override
public boolean equals(Object other) {
    if (!(other instanceof Set)) {
        return false;
    }

    Set otherSet = (Set) other;
    return isSubset(otherSet) && otherSet.isSubset(this);
}
```

טעויות נפוצות:

- סטודנטים רבים הגדירו את טיפוס הפרמטר בתור Set. דבר זה שגוי מאחר ויש לדרוס את הפעולה clone ולא להעמיס אותה.
- סטודנטים רבים בדקו רק האם הקבוצה הנוכחית מוכלת בקבוצה שהתקבלה כפרמטר, אך לא בדקו את הכיוון ההפוך. בדיקה זו הכרחית במידה ולא מתבצעת בדיקה האם התכונה index של שתי הקבוצות שווה (לא מספיק לבדוק את אורך שני המערכים).

סעיף ד (10 נקודות)

נרצה לאפשר מעבר על איברי הקבוצה על ידי לולאת foreach. לשם כך נגדיר שני סוגי סריקה אפשריים: מעבר על איברי הקבוצה לפי הסדר בו הם מופיעים במערך, ומעבר על האיברים בסדר הפוך מן הסדר בו הם מופיעים במערך.

לשם כך, תחילה עלייך להגדיר במחלקה Set פעולה בשם setScanningType אשר מקבלת מספר שלם. המספר אשר מועבר לפעולה קובע את סוג הסריקה הנוכחי של האיברים עד לקריאה הבאה לפעולה, כאשר הערך 0 מציין סריקה לפי הסדר והערך 1 מציין סריקה בסדר הפוך. ניתן להניח שלא יועבר לפעולה פרמטר שאינו תקין.

לאחר מכן, עלייך ליצור מחלקה בשם SetIterator אשר מממשת את הממשק Iterator<Integer>. יש לממש במחלקה זו את הפעולות hasNext ו-next בהתאם לסוג הסריקה.

לבסוף, עלייך לממש במחלקה Set את הממשק Iterable<Integer> ואת הפעולה iterator אשר מוגדרת בממשק.

פתרון

מחלקת האיטרטור:

```
class SetIterator implements Iterator<Integer> {
    private int[] values;
    private int scanningType;
    private int scanningIndex;
    private int setIndex;

    public SetIterator(int[] values, int scanningType, int setIndex) {
        this.values = values;
        this.scanningType = scanningType;
        this.setIndex = setIndex;
        scanningIndex = scanningType == 0 ? 0 : setIndex - 1;
    }

    @Override
    public boolean hasNext() {
        if (scanningType == 0) {
            return scanningIndex < setIndex;
        }
        return scanningIndex > 0;
    }

    @Override
    public Integer next() {
        if (scanningType == 0) {
            return values[scanningIndex++];
        }
        return values[scanningIndex--];
    }
}
```

השינויים במחלקה Set:

```
public class Set implements Iterable<Integer> {
    /* All fields which defined in the question */
    private int scanningType;

    public Set(int maxSize) {
        /* Constructor like in the question */
        scanningType = 0;
    }

    public void setScanningType(int scanningType) {
        this.scanningType = scanningType;
    }

    @Override
    public Iterator<Integer> iterator() {
        return new SetIterator(values, scanningType, index);
    }
}
```

קוד השאלה בשלמותו:

```
class SetException extends RuntimeException { }
class ElementAlreadyExists extends SetException { }
class SetOverflowException extends SetException { }

public class Set implements Cloneable, Iterable<Integer> {
    private int[] values;
    private int index;
    private int scanningType;

    public Set(int maxSize) {
        this.values = new int[maxSize];
        index = 0;
        scanningType = 0;
    }

    public void addElement(int num) {
        for (int i = 0; i < index; i++) {
            if (values[i] == num) {
                throw new ElementAlreadyExists();
            }
        }

        if (index == values.length) {
            throw new SetOverflowException();
        }

        values[index] = num;
        index++;
    }


    @Override
    public Set clone() {
        try {
            Set clonedSet = (Set) super.clone();
            clonedSet.values = values.clone();
            return clonedSet;

        } catch (CloneNotSupportedException e) {
            return null;
        }
    }
```

```
    private boolean isSubset(Set otherSet) {
        for (int i = 0; i < index; i++) {
            boolean flag = false;
            for (int j = 0; j < otherSet.index; j++) {
                if (values[i] == otherSet.values[j]) {
                    flag = true;
                }
            }
            if (!flag) {
                return false;
            }
        }
        return true;
    }

    @Override
    public boolean equals(Object other) {
        if (!(other instanceof Set)) {
            return false;
        }

        Set otherSet = (Set) other;
        return isSubset(otherSet) && otherSet.isSubset(this);
    }

    public void setScanningType(int scanningType) {
        this.scanningType = scanningType;
    }

    @Override
    public Iterator<Integer> iterator() {
        return new SetIterator(values, scanningType, index);
    }
}

class SetIterator implements Iterator<Integer> {
    private int[] values;
    private int scanningType;
    private int scanningIndex;
    private int setIndex;

    public SetIterator(int[] values, int scanningType, int setIndex) {
        this.values = values;
        this.scanningType = scanningType;
        this.setIndex = setIndex;
        scanningIndex = scanningType == 0 ? 0 : setIndex;
    }
```

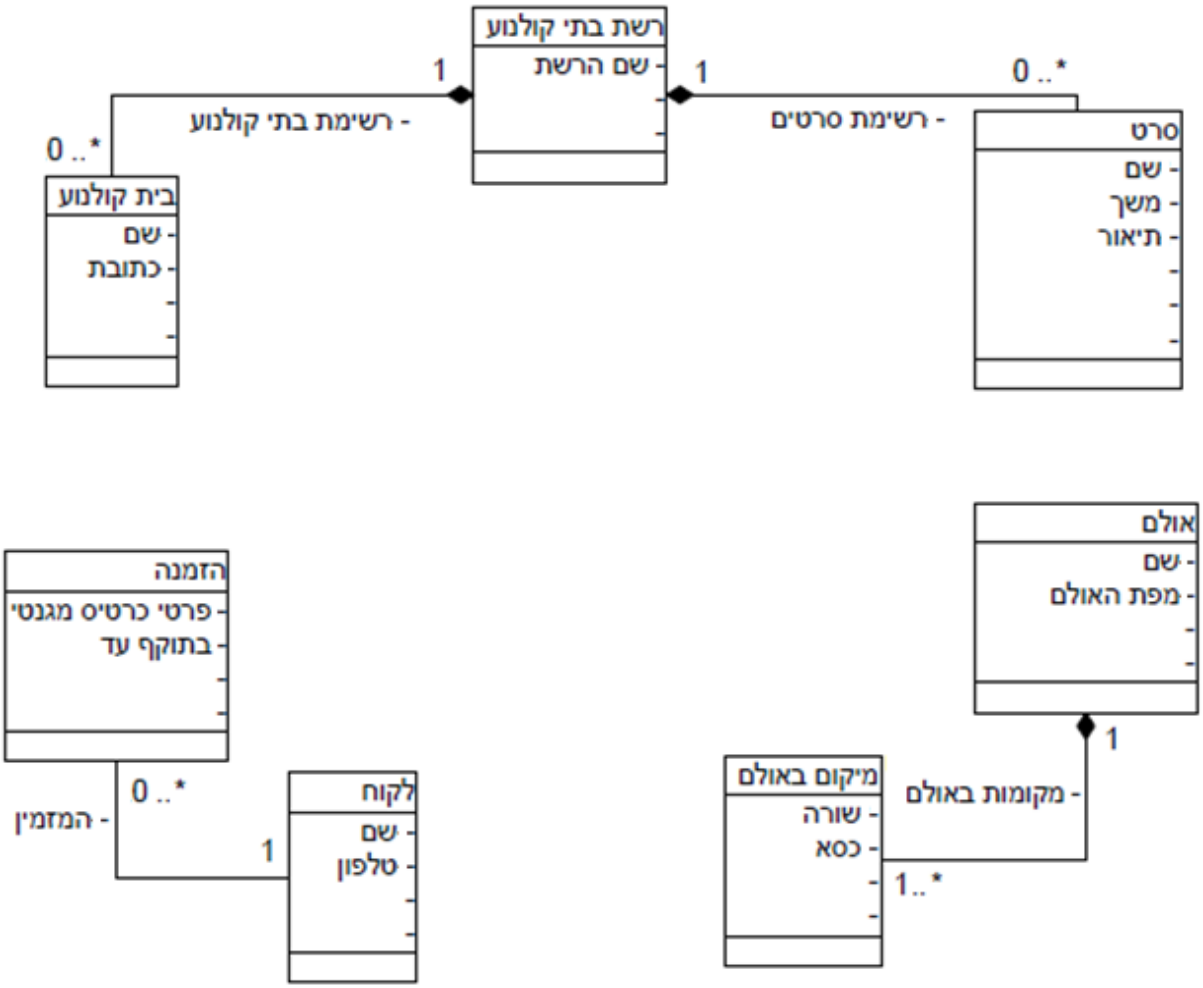
```
    @Override
    public boolean hasNext() {
        if (scanningType == 0) {
            return scanningIndex < setIndex;
        }
        return scanningIndex > 0;
    }

    @Override
    public Integer next() {
        if (scanningType == 0) {
            return values[scanningIndex++];
        }
        return values[scanningIndex--];
    }
}
```

שאלה 4 (15 נקודות)

בבעלותה של רשת בתי הקולנוע Technion Planet בתי קולנוע בכל רחבי הארץ. הרשת מקימה מערכת ממוחשבת דרכה ימכרו כרטיסים לכל בתי הקולנוע השייכים לרשת. בכל בית קולנוע יש מספר אולמות קולנוע. הסרטים המוקרנים יכולים להיות שונים מבית קולנוע אחד למשנהו, אך לפעמים אותו סרט מוקרן בבתי קולנוע שונים. עלות צפיה בסרט הנה קבועה בכל בתי הקולנוע של הרשת ולכל הסרטים. לרשת יש מנויים המקבלים הנחה קבועה של X אחוזים על צפיה בכל אחד מהסרטים.

להלן מידול התחלתי של המערכת המוצגת על ידי תרשים UML:



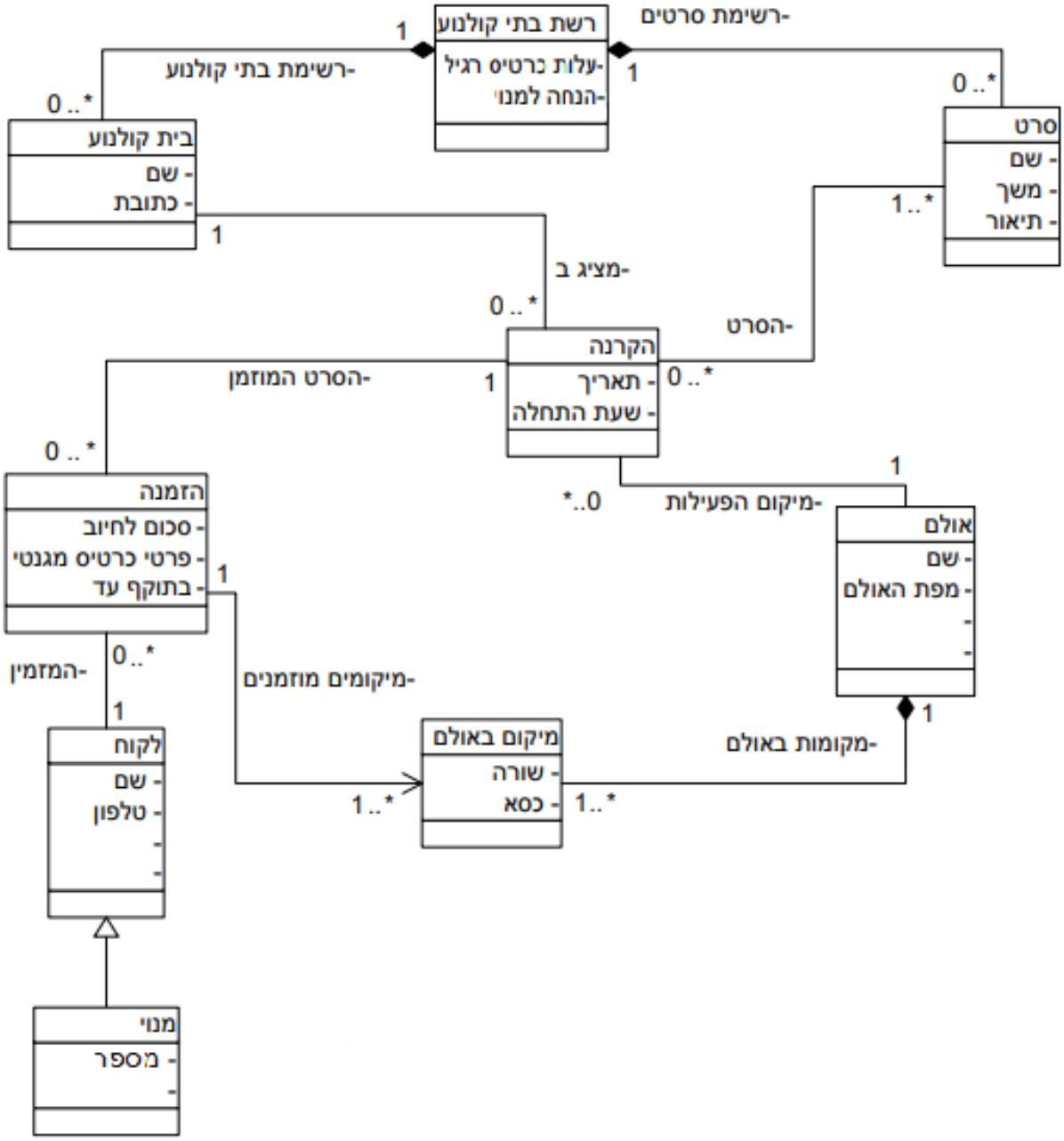
עלייך להוסיף לתרשים **אך ורק** את המידע הבא באמצעות יחסים, ריבויים (Multiplicity), תפקידים (Roles), מחלקות ומאפיינים:

- באילו בתי קולנוע מוקרן סרט מסוים, באיזה אולם בבית הקולנוע המסוים הסרט מוקרן, באילו תאריכים ובאילו שעות.
- מי הזמין כרטיס(ים), לאיזו הקרנה של הסרט הוזמן הכרטיס והאם מדובר במנוי או בלקוח מזדמן (לקוח שאינו מנוי).
- עלות צפיה בסרט ללקוח מזדמן ולמנוי.
- מה שמו ומספרו הסידורי של כל מנוי.

שימי לב:

- הוספת מחלקה / מאפיין / יחס מיותרים או לא נכונים עשויה להוריד נקודות.
- לא בהכרח נדרש להוסיף יחס הורשה לשרטוט.
- אין חובה לרשום מאפיינים אשר מייצגים יחס אך הוספתם לא תוריד ניקוד.
- אל תמהרי להוסיף מחלקות. עדיף להוסיף יחס או מאפיין במידה והם עונים על הדרישה.
- יחס יכול להיות יחס פשוט (Association), הכלה חזקה (Composition), הכלה חלשה (Aggregation) והורשה (Inheritance).
- התבליטים הריקים אשר מופיעים בכל מחלקה אינם מעידים על כך שעלייך להוסיף תכונות למחלקה וכן הם אינם מעידים על מספר התכונות של המחלקה.
- יש לענות על שאלה זאת **על גבי טופס הבחינה**.
- לנוחיותך, בדף האחרון של הבחינה מופיע עותק נוסף של התרשים. ניתן להיעזר בעותק זה לצורך ביצוע סקיצות של התרשים. על התשובה הסופית להופיע בשאלה זו ולא על גבי העותק.

פתרון


רשת בתי קולנוע
-עלות כרטיס רגיל
-הנחה למנוי
-רשימת סרטים
-רשימת בתי קולנוע
בית קולנוע
- שם
- כתובת
סרט
- שם
- משך
- תיאור
-מציג ב
הקרנה
- תאריך
- שעת התחלה
-הסרט
-הסרט המוזמן
הזמנה
- סכום לחיוב
- פרטי כרטיס מגנטי
- בתוקף עד
-מיקום הפעילות
אולם
- שם
- מפת האולם
-מיקומים מוזמנים
-המזמין
לקוח
- שם
- טלפון
מיקום באולם
- שורה
- כסא
-מקומות באולם
מנוי
- מספר

שאלה 5 (18 נקודות)

בשווקים רבים מוכרים יכולים להציע מוצר למכירה על ידי ביצוע מכרז. כל מכרז מורכב ממוכר יחיד וממספר לא מוגבל של משתתפים. בכל מכרז המשתתפים מציעים מחיר בעבור המוצר באופן שאינו גלוי לשאר המשתתפים. בעבור המוכר שומרים את שמו, את מספר הטלפון שלו ואת הפריט אשר הוא מציע למכירה. בעבור כל משתתף במכרז שומרים את שמו, את מספר הטלפון שלו ואת הבִיד (Bid) שלו (הסכום אשר הוא הציע במכרז).

ישנם שני סוגים של מכרזים – מכרז מחיר ראשון ומכרז מחיר שני.

במכרז מחיר ראשון המשתתף אשר הציע את המחיר הגבוה ביותר מקבל את המוצר ומשלם את המחיר שהציע.

במכרז מחיר שני המשתתף אשר הציע את המחיר הגבוה ביותר מקבל את המוצר אך משלם את הסכום השני בגובהו מבין הסכומים אשר הוצעו.

סעיף א (6 נקודות)

כתבי מחלקות אשר מייצגות מוכר במכרז ומשתתף במכרז. במחלקות יש לכלול תכונות מתאימות ובנאים.

פתרון

מאחר ולמוכר ולקונים יש תכונות משותפות, נאגד אותן במחלקה אחת:

```
class Person {
    private String name;
    private String phone;

    public Person(String name, String phone) {
        this.name = name;
        this.phone = phone;
    }
}
```

מחלקת מוכר:

```
class Buyer extends Person {
    private int bid;

    public Buyer(String name, String phone, int bid) {
        super(name, phone);
        this.bid = bid;
    }

    public int getBid() {
        return bid;
    }
}
```

מחלקת קונה:

```
class Buyer extends Person {
    private int bid;

    public Buyer(String name, String phone, int bid) {
        super(name, phone);
        this.bid = bid;
    }

    public int getBid() {
        return bid;
    }
}
```

טעות נפוצה בסעיף זה הייתה לא ליצור את מחלקת Person, מה שהוביל לשכפול קוד.

<u>סעיף ב (6 נקודות)</u>

כתבי מחלקה מופשטת בשם Auction אשר מייצגת מכרז כלשהו. במחלקה יש לכלול תכונות מתאימות ואת הפעולות הבאות:

- בנאי אשר מקבל עצם המייצג את המוכר במכרז.
- פעולה אשר מוסיפה משתתף למכרז.
- פעולה בשם getWinner אשר מחזירה את המשתתף שניצח במכרז. ניתן להניח כי יש רק משתתף אחד כזה.
- פעולה מופשטת בשם getWinningBid אשר מחזירה את הסכום אותו המנצח במכרז צריך לשלם. ניתן להניח כי בעת הקריאה לפעולה זו יש במכרז לפחות שני משתתפים.

פתרון

```
public abstract class Auction {
    private Seller seller;
    protected List<Buyer> buyers;

    public Auction(Seller seller) {
        this.seller = seller;
        buyers = new ArrayList<>();
    }

    public void addBuyer(Buyer buyer) {
        buyers.add(buyer);
    }

    public Buyer getWinner() {
        int maxBid = Integer.MIN_VALUE;
        Buyer maxBuyer = null;

        for (Buyer buyer : buyers) {
            int bid = buyer.getBid();
            if (bid > maxBid) {
                maxBid = bid;
                maxBuyer = buyer;
            }
        }

        return maxBuyer;
    }

    public abstract int getWinningBid();
}
```

סעיף ג (6 נקודות)

כתבי מחלקות בשם FirstPriceAuction ו-SecondPriceAuction אשר מייצגות מכרז מחיר ראשון ומכרז מחיר שני בהתאמה. על המחלקות לרשת מן המחלקה Auction ולממש את הפעולה המופשטת.

הערות לכל סעיפי השאלה:

- יש לתכנן את המחלקות על פי עקרונות תכנות מונחה עצמים נכונים.
- ניתן להשתמש בפעולות get ו-set לתכונות המחלקות מבלי לממש אותן.
- אין צורך להקפיד על ביצוע import.

פתרון

```
class FirstPriceAuction extends Auction {
    public FirstPriceAuction(Seller seller) {
        super(seller);
    }

    @Override
    public int getWinningBid() {
        return getWinner().getBid();
    }
}

class SecondPriceAuction extends Auction {
    public SecondPriceAuction(Seller seller) {
        super(seller);
    }

    @Override
    public int getWinningBid() {
        int maxBid = getWinner().getBid();
        int secondMax = Integer.MIN_VALUE;

        for (Buyer buyer : buyers) {
            int currentBid = buyer.getBid();
            if (currentBid > secondMax && currentBid < maxBid) {
                secondMax = currentBid;
            }
        }
        return secondMax;
    }
}
```

פתרון נוסף עבור המחלקה SecondPriceAuction הוא להסיר באופן זמני מן רשימת המשתתפים את המשתתף אשר הציע את הביד הגבוה ביותר, להשתמש בפעולה getWinner פעם נוספת, ולאחר מכן להחזיר את המשתתף לרשימה.

שאלה 6 (15 נקודות)

לפנייך קטע קוד אשר מכיל הגדרה של המחלקה A ומספר שורות קוד נוספות. מיד לאחריו מופיע קוד המקור של הפעולות put ו-get של המחלקה הגנרית HashMap.

```
1  class A {
2      private int a;
3      public A(int a){
4          this.a = a;
5      }
6      public int hashCode(){
7          return a;
8      }
9  }
10 HashMap<A, Integer> m = new HashMap<A, Integer>();
11 A obj;
12 System.out.println(m.put(obj = new A(5), 1));
13 System.out.println(m.put(obj = new A(21), 2));
14 System.out.println(m.put(obj = new A(5), 3));
15 System.out.println(m.put(obj, 4));
16 System.out.println(m.get(new A(5)));
```

קוד המקור:

```
1  public class HashMap<K,V> extends AbstractMap<K,V> implements Map<K,V>, Cloneable, Serializable {
2      static final int TREEIFY_THRESHOLD = 8;
3      Node<K,V>[] table;
4      public V put(K key, V value) {
5          return putVal(hash(key), key, value, false, true);
6      }
7      final V putVal(int hash, K key, V value, boolean onlyIfAbsent,boolean evict) {
8          Node<K,V>[] tab; Node<K,V> p; int n, i;
9          if ((tab = table) == null || (n = tab.length) == 0)
10             n = (tab = resize()).length;
11         if ((p = tab[i = (n - 1) & hash]) == null)
12             tab[i] = new Node<>(hash, key, value, null);
13         else {
14             Node<K,V> e; K k;
15             if (p.hash == hash && ((k = p.key) == key || (key != null && key.equals(k))))  e = p;
16             else if (p instanceof TreeNode)   e = ((TreeNode<K,V>)p).putTreeVal(this, tab, hash, key, value);
17             else {
18                 for (int binCount = 0; ; ++binCount) {
19                     if ((e = p.next) == null) {
20                         p.next = new Node<>(hash, key, value, next);
21                         if (binCount >= TREEIFY_THRESHOLD - 1)  treeifyBin(tab, hash);
22                         break;
23                     }
24                     if (e.hash == hash && ((k = e.key) == key || (key != null && key.equals(k))))
25                         break;
26                     p = e;
27                 }
28             }
29             if (e != null) {
30                 V oldValue = e.value;
31                 if (!onlyIfAbsent || oldValue == null)  e.value = value;
32                 return oldValue;
33             }
34         }
35         ++modCount;
36         if (++size > threshold) resize();
37         return null;
38     }
```

```
39      public V get(Object key) {
40          Node<K,V> e;
41          return (e = getNode(hash(key), key)) == null ? null : e.value;
42      }
43      final Node<K,V> getNode(int hash, Object key) {
44          Node<K,V>[] tab; Node<K,V> first, e; int n; K k;
45          if ((tab = table) != null && (n = tab.length) > 0 &&
46              (first = tab[(n - 1) & hash]) != null) {
47              if (first.hash == hash &&
48                  ((k = first.key) == key || (key != null && key.equals(k))))
49                  return first;
50              if ((e = first.next) != null) {
51                  if (first instanceof TreeNode)
52                      return ((TreeNode<K,V>)first).getTreeNode(hash, key);
53                  do {
54                      if (e.hash == hash &&((k = e.key) == key || (key != null && key.equals(k))))
55                          return e;
56                  } while ((e = e.next) != null);
57              }
58          }
59          return null;
60      }
61      static class Node<K,V> implements Map.Entry<K,V> {
62          final int hash;
63          final K key;
64          V value;
65          Node<K,V> next;
66          Node(int hash, K key, V value, Node<K,V> next) {
67              this.hash = hash; this.key = key;
68              this.value = value; this.next = next;
69          }
70      }
71  }
```

<u>סעיף א (9 נקודות)</u>

ציירי תרשים עצמים עבור התכונה `table` של המחלקה `HashMap` לאחר ביצוע שורות 10-15 בקטע הקוד הנתון.

עלייך לכלול בתרשים את ערכי התכונות של כל העצמים. במידה והתכונות מתעדכנות במהלך ביצוע קטע הקוד, יש למחוק את הערך הישן של התכונה על ידי קו (באופן שעדיין יהיה ניתן לקרוא אותו) ולרשום ליד את הערך החדש.

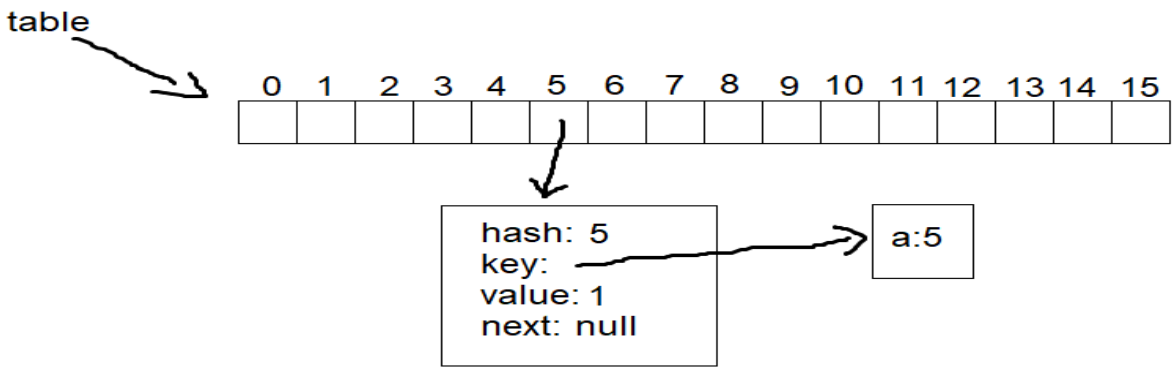
בנוסף, עלייך להסביר את התרשים ולציין באופן מפורש את מספרי השורות של פעולות ה-`put` וה-`get` אשר התבצעו והובילו לתרשים.

פתרון

הטבלה לאחר ביצוע שורה 10 בקוד:

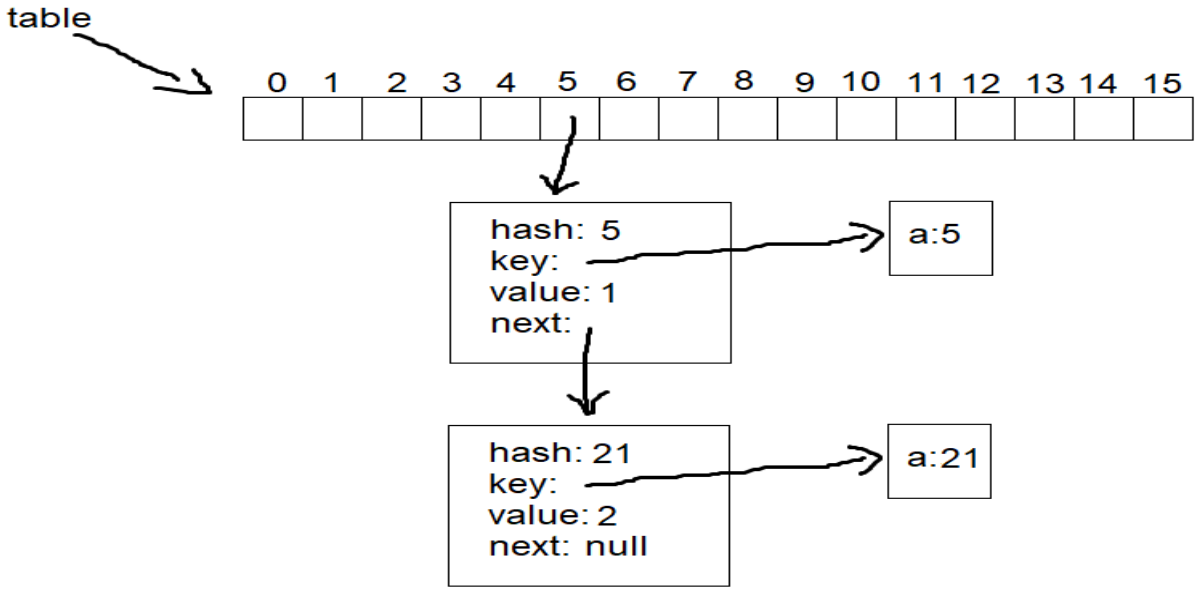
```
table = null
```

הטבלה לאחר ביצוע שורה 12 בקוד:



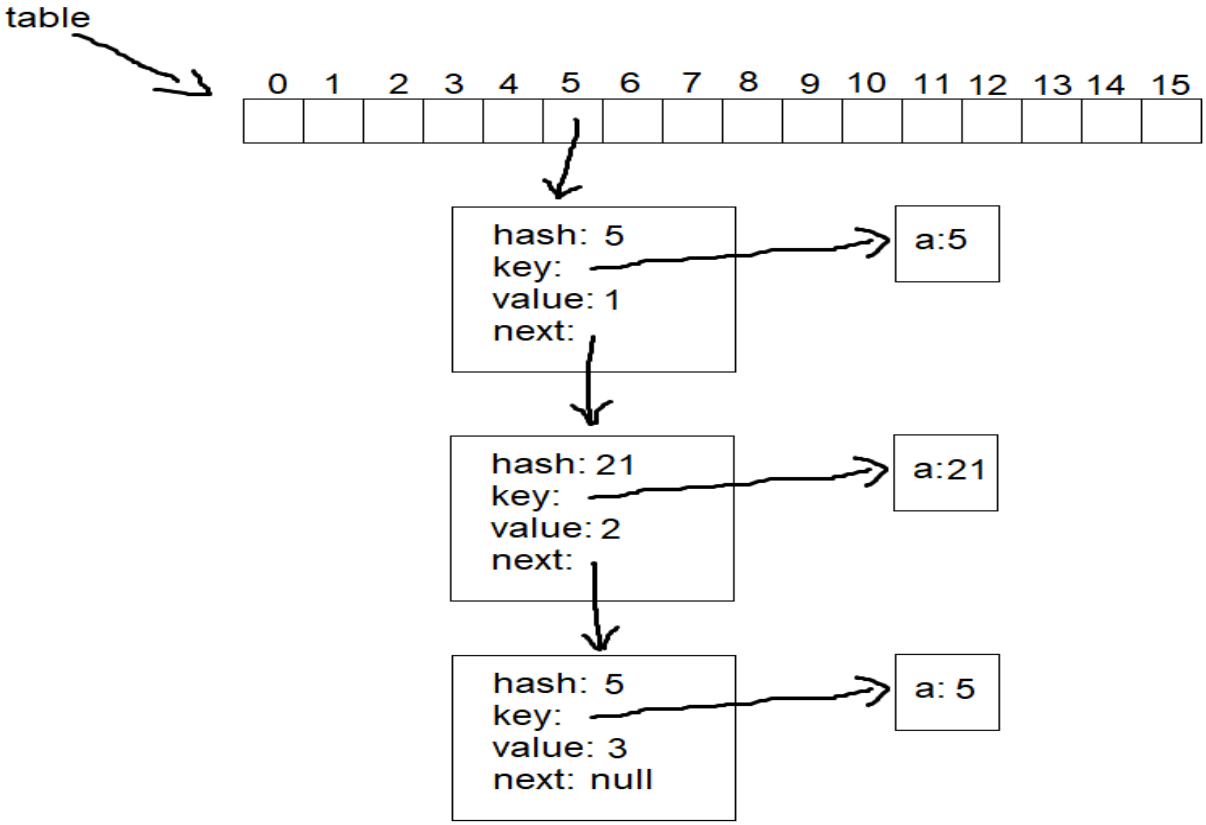
השורות אשר התבצעו והובילו לתרשים: 10, 12, 35, 37.

הטבלה לאחר ביצוע שורה 13 בקוד:



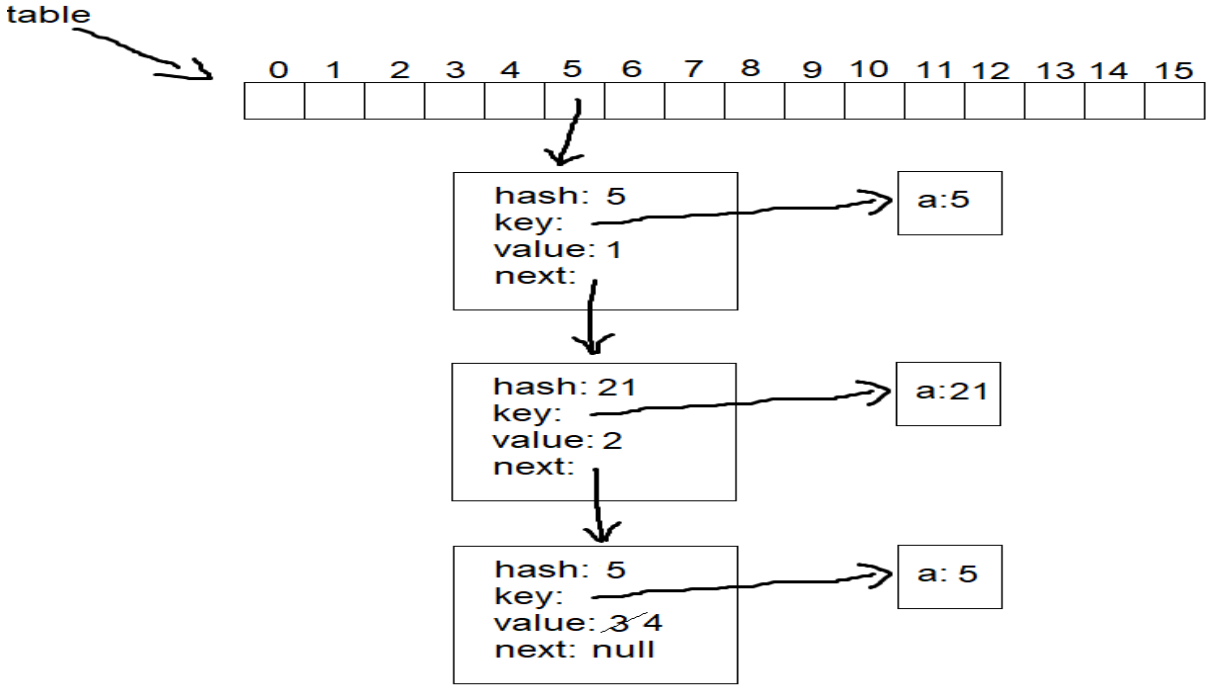
השורות אשר התבצעו והובילו לתרשים: 14, 18, 19, 20, 29, 35, 37.

הטבלה לאחר ביצוע שורה 14 בקוד:



השורות אשר התבצעו והובילו לתרשים: 14, 18, 26, 18, 20, 35, 37.

הטבלה לאחר ביצוע שורה 15 בקוד:



השורות אשר התבצעו והובילו לתרשים: 14, 18, 26, 18, 24, 29, 30, 32.

הטבלה לאחר ביצוע שורה 16 בקוד נשארת אותו הדבר.

השורות אשר התבצעו: 53, 56, 53, 56, 59.

<u>סעיף ב (6 נקודות)</u>

מה יהיה הפלט לאחר ביצוע קטע הקוד הנתון?

יש להסביר את תשובתך ולציין באופן מפורש את מספרי השורות של פעולות ה-put וה-get אשר התבצעו והובילו לפלט זה.

תזכורות:

- פעולת ה-hash של המחלקה HashMap משתמשת בפעולה hashCode של המפתח אותו מוסיפים.
- בקריאה הראשונה לפעולה resize של המחלקה HashMap חוזר הערך 16.
- לכל שני מספרים שלמים m, n כאשר n הוא חזקה שלמה של 2, הפעולה $m \,\&\, (n-1)$ שקולה לפעולה $m \,\%\, n$ כאשר % הוא אופרטור המודולו.

פתרון

הפלט המתקבל:

null

null

null

3

null

הסברים:

null – The first added key does not exist, so the put function returns null. You can follow the lines presented earlier.

null – This added key does not exist.

null – It seems that the `new A(5)` object does exist (because we added it before), but the put function will not find it as the equals method is not overridden and thus the default one will be returning false. That means, the put method will consider this key, `new A(5)`, as a new one and it will be added as a new entry.

3 – In this case the sent key is `obj` which is not a new one but simply a reference to a previous one so the put function will find its entry and it will replace that entry with the new value 4 and returns the old value 3.

null – Again, we are creating a new key which will not be found as the equals method is not overridden.

שאלה 7 (5 נקודות)

אחת מתבניות העיצוב אשר נלמדו בקורס הינה סינגלטון (Singleton). בתרגול ראינו כי ניתן לממש את תבנית העיצוב בעזרת שתי גישות שונות: אתחול עצל (Lazy Initialization) ואתחול להוט (Eager Initialization).

בתרגול ראינו כי המימוש העצל אינו בטוח לעבודה עם מספר תהליכונים (Thread Safe) ומטרתה של השאלה היא לפתור את בעיה זו.

סעיף א (2 נקודות)

תארי במילים דרך בה ניתן להפוך את שיטת האתחול העצל לבטוחה לעבודה עם מספר תהליכונים.

פתרון

ניתן להוסיף את המילה synchronized לכותרת הפעולה getInstance ובכך למנוע משני תהליכונים לבצע את הפעולה באותו הזמן.

לחילופין, ניתן להגדיר משתנה סטטי ולהשתמש בו בתור Monitor Object בבלוק synchronized בתוך הפעולה getInstance או להגדיר מנעול סטטי.

סעיף ב (3 נקודות)

ממשי את הצעתך מסעיף ב'. לצורך כך, עלייך ליצור מחלקה בשם Singleton אשר מממשת את תבנית העיצוב סינגלטון ונעזרת באתחול עצל ובהצעתך מן הסעיף הקודם.

פתרון

פתרון על ידי פעולה synchronized:

```
public class Singleton {
    private static Singleton instance = null;

    private Singleton() { }

    public static synchronized Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }

        return instance;
    }
}
```

פתרון על ידי בלוק synchronized:

```
public class Singleton {
    private static Singleton instance = null;
    private static final Object obj = new Object();

    private Singleton() { }

    public static Singleton getInstance() {
        synchronized (obj) {
            if (instance == null) {
                instance = new Singleton();
            }
        }

        return instance;
    }
}
```

פתרון על ידי שימוש במנעול:

```
public class Singleton {
    private static Singleton instance = null;
    private static final Lock lock = new ReentrantLock();

    private Singleton() { }

    public static Singleton getInstance() {
        try {
            lock.lock();
            if (instance == null) {
                instance = new Singleton();
            }
        } finally {
            lock.unlock();
        }

        return instance;
    }
}
```

עותק נוסף של תרשים המחלקות: