

תוכנה 1 – אביב תשע"ג

תרגיל מספר 8

(2 נקודות)

הנחיות כלליות:

- קראו בעיון את קובץ נהלי הגשת התרגילים אשר נמצא באתר הקורס.
- הגשת התרגיל תיעשה במערכת ה-moodle בלבד (<http://moodle.tau.ac.il/>).
- יש להגיש קובץ zip יחיד הנושא את שם המשתמש האוניברסיטאי שלכם ומספר התרגיל (לדוגמא, עבור שם המשתמש aviv יקרא הקובץ aviv_hw8.zip). קובץ ה-zip יכול:
 - א. קובץ פרטים אישיים בשם details.txt המכיל את שמכם ומספר ת.ז.
 - ב. קבצי ה-java של התוכניות אותם התבקשתם לממש.
 - ג. קובץ טקסט answers.txt עם התשובות לשאלות (אפשר קובץ טקסט או קובץ Word).

שימו לב,

- לכל אורך התרגיל ניתן להוסיף שירותי עזר בתנאי שאינם פוגעים בנדרש בסעיפים אחרים.
- כדי להקל עליכם, סיפקנו מחלקות הבודקות חלק מהדוגמאות המופיעות בגוף התרגיל. אם הקוד שמימשתם עונה נכונה על הדוגמאות, תתקבל הודעה שהבדיקה עברה בהצלחה. אחרת, תתקבלנה הודעות שגיאה מתאימות.
- חשוב:** הודעת הצלחה אינה מעידה על נכונות הקוד אלא על התאמה לדוגמאות הספציפיות אותן סיפקנו. לעומת זאת, הודעות שגיאה מעידות על כך שהמימוש שגוי.
- חשוב מאוד:** קוד שאינו מתקמפל עם תוכניות הבדיקה לא ייבדק.
- בהצלחה

חלק א' (40 נקודות)

המנשק IPAddress המופיע למטה מייצג כתובת של Internet Protocol (IP). דוגמאות לכתובות IP הן:

127.0.0.1
192.168.1.10

כתובת IP, כפי שנתן לראות, מורכבת מארבעה חלקים. ערכו של כל אחד מהחלקים הוא מספר שלם בין 0 ל-255. נממש את המנשק בעזרת שלושה ייצוגים שונים: הראשון עושה שימוש במחרוזות, השני במערך של מספרים מסוג short ואילו השלישי משתמש ב-int יחיד (הסבר מפורט בהמשך).

- א. כתבו **שלוש** מחלקות שונות המממשות את המנשק IPAddress המוגדר למטה:
 - מחלקה בשם IPAddressString, המממשת את המנשק בעזרת ייצוג פנימי של String.

2. מחלקה בשם IPAddressShort, המממשת את המנשק בעזרת ייצוג פנימי של מערך בגודל 4 של short. כל תא במערך יחזיק מספר בתחום 0..255.
3. מחלקה בשם IPAddressInt, המממשת את המנשק בעזרת int יחיד.

לכל אחת מהמחלקות יהיה בנאי המתאים לייצוג הפנימי שלה וכמובן כל אחת מהן מממשת את המנשק. ניתן להניח בחוזה שהבנאים מקבלים קלט תקין ליצירת כתובת IP (בהתאם לייצוג הפנימי של כל מחלקה).

```
public interface IPAddress {

    /**
     * Returns a string representation of the IP address, e.g.
     * "192.168.0.1"
     */
    public String toString();

    /**
     * Compares this IPAddress to the specified object
     * @param other
     *         the IPAddress to compare the current against
     * @return true if both IPAddress objects represent the same
     *         IP address, false otherwise.
     */
    public boolean equals(IPAddress other);

    /**
     * Returns one of the four parts of the IP address. The parts
     * are indexed from left to right. For example, in the IP
     * address 192.168.0.1 part 0 is 192, part 1 is 168,
     * part 2 is 0 and part 3 is 1.
     * (Each part is called an octet as its representation
     * requires 8 bits.)
     * @param index
     *         The index of the IP address part (0, 1, 2 or 3)
     * @return The value of the specified part.
     */
    public int getOctet(int index);

    /**
     * Mask the current address with the given one. The masking
     * operation is a bitwise 'and' operation on all four parts of
     * the address.
     * @param mask
     *         the IP address with which to mask
     * @return A new IP address representing the result of the mask
     *         operation. The current address is not modified.
     */
    public IPAddress mask(IPAddress mask);

}
```

הייצוגים השונים :

1. מחרוזת – יש להשתמש במחרוזת יחידה לצורך ייצוג כתובת ה IP . כל הפעולות יבוצעו בעזרת מחרוזת זו.
2. מערך – כל אחד מחלקי הכתובת (מספר שלם 0-255) יוחזק בתא במערך.
3. int – נשים לב שכל אחד מחלקי הכתובת הוא מספר שלם בתחום 0-255 (כולל), לפיכך ניתן לייצג אותו בעזרת 8 ביטים. לפיכך, את ארבעת חלקי הכתובת ניתן לייצג באמצעות 32 ביטים שזהו בדיוק גודלו של int.

נשתמש ב-int לא כמספר אלא כרצף של 32 ביטים. את הפעולות הדרושות נבצע לא בעזרת פעולות אריתמטיות כי אם בעזרת פעולות על ביטים (&, <<, >> וכדומה).

לדוגמא, הכתובת 127.0.0.1 תיוצג ע"י רצף הביטים 01111111000000000000000000000001. החלק הראשון ייצוג ע"י הביטים במקומות 0-7 (משמאל לימין), החלק השני ע"י הביטים 8-15, השלישי ע"י 16-23 והרביעי ע"י ביטים 24-31.

הערה חשובה: עליכם לממש את המתודות באופן שונה בכל מחלקה בהתאם לייצוג הפנימי. אין להמיר את הייצוג הפנימי לייצוג אחר לצורך מימוש פעולה (רק לצורך פלט). המחלקות השונות לא "ייעזרו" זו בזו (למשל, אסור להשתמש ב- IPAddressString על מנת לממש את IPAddressInt).

בנוסף, ממשו את המחלקה IPAddressFactory המגדירה את המתודות הבאות :

```
public class IPAddressFactory {
    public static IPAddress createAddress(String ip) {
        ...
    }
    public static IPAddress createAddress(short[] ip) {
        ...
    }
    public static IPAddress createAddress(int ip) {
        ...
    }
}
```

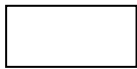
כל אחת מהמתודות הסטטיות יוצרת אובייקט מטיפוס IPAddress, כשהאובייקט הקונקרטי נקבע על סמך טיפוס הקלט.

הערה: מחלקה שתפקידה היחיד הוא יצור אובייקטים של מחלקות אחרות נקראת *factory* class. מחלקות אלו מסתירות את פרטי יצור האובייקטים מלקוחות של אובייקטים אלו. השימוש בטכניקה זו נועד להסתיר את המחלקות הקונקרטיות שמממשות מנשק.

דוגמה לפעולת mask :

'and' (&) truth table

Bit1/bit2	0	1
0	0	0
1	0	1



Mask example: if we mask 192.168.0.1 with the mask 127.0.0.1 then the result in binary is:
 01000000.00000000.00000000.00000001
 → IP address 64.0.0.1

להלן תכנית המדגימה את השימוש במחלקה IPAddressFactory ובמנשק.

```
public class TestIPAddress {  
  
    public static void main(String[] args) {  
        int address1 = -1062731775; // 192.168.0.1  
        short[] address2 = { 10, 1, 255, 1 }; // 10.1.255.1  
  
        IPAddress ip1 = IPAddressFactory.createAddress(address1);  
        IPAddress ip2 = IPAddressFactory.createAddress(address2);  
        IPAddress ip3 = IPAddressFactory.createAddress("127.0.0.1");  
  
        for (int i = 0; i < 4; i++) {  
            System.out.println(ip1.getOctet(i));  
        }  
  
        System.out.println(ip1.mask(ip2));  
        System.out.println("equals: " + ip1.equals(ip2));  
    }  
}
```

חלק ב' (60 נקודות)

בשאלה זו נדון בחבילת תוכנה לעיבוד תמונה.

ייצוג: תמונה תיוצג ע"י מטריצה דו-מימדית, כאשר הערך במקום ה-x,y הוא מספר שלם המציין צבע. ערכי ה-x גדלים משמאל לימין ואילו ערכי ה-y גדלים מלמעלה למטה. כלומר, x הוא קואורדינאטת העמודות ואילו y הוא קואורדינאטת השורות. נקודה בתמונה נקראת פיקסל.

צבעים: לשם פשטות נעסוק רק בתמונות בגווני אפור, בהן הצבע מיוצג ע"י מספר בין 0, המציין את הצבע השחור ל-255 המציין את הצבע הלבן. כל ערך ביניים מייצג רמת אפור מתאימה.

בתוכנה שנבנה, תמונות ייוצגו ע"י המנשק:

```
public interface Image {  
    /**  
     * @pre 0 <= x < getWidth()  
     * @pre 0 <= y < getHeight()  
     */  
    public int getPixel(int x, int y);  
  
    /**  
     * @post $ret >= 0  
     */  
    public int getWidth();  
    /**  
     * @post $ret >= 0  
     */  
    public int getHeight();  
}
```

פעולות: התמרה (טרנספורמציה) של תמונה היא פעולה המשנה את התמונה. בחבילת התוכנה המוצעת, כל התמרה תמומש במחלקה ייעודית המממשת אף היא את המנשק Image.

תמונת המקור ופרמטרים הנוגעים להתמרה יועברו בבנאי של המחלקה הייעודית; במימוש מתודות המנשק Image ניתן להיעזר במתודות של תמונת המקור כדי לחשב את הפלט המתאים.

שימו לב שחבילת התוכנה פועלת על פי עיקרון דומה לחבילת הזרמים ב-Java, בה פעולות על זרמים ממומשות בצורה דומה.

לדוגמא, הקוד הבא מפעיל את ההתמרה Flip, ההופכת את תמונת הקלט (שיקוף ציר ה-y):
 Image input = ...
 Image output = new Flip(input);

בתרגיל זה אתם נדרשים לממש את שש ההתמרות המופיעות למטה. לכל התמרה נתונות שתי דוגמאות, האחת להפעלת ההתמרה על מטריצת מספרים, והשנייה להפעלתה על תמונה. תמונות הקלט והפלט מצויות בספרייה resources בפרוייקט הבדיקות באתר.

1. Flip – התמרה המשקפת את ציר ה-y של התמונה.

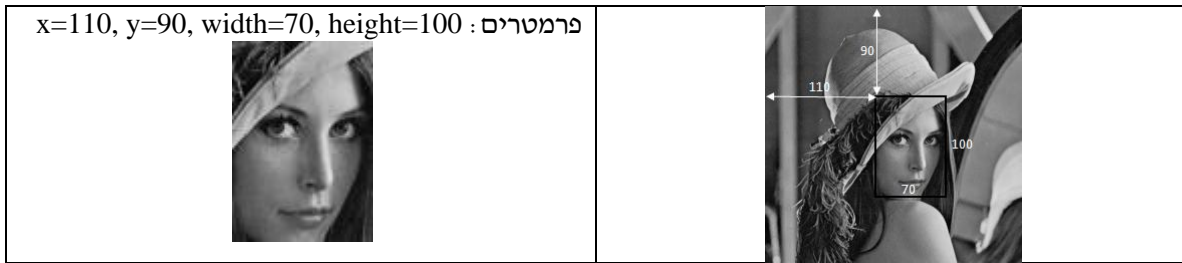
דוגמאות:

תמונה מעובדת (output)				תמונה מקורית (input)			
5	6	7	8	1	2	3	4
1	2	3	4	5	6	7	8
							

2. Crop – התמרה המחלצת אזור מלבני מתוך התמונה. הפרמטרים הנוספים (x,y,width, height) לבנאי מציינים את האזור הרצוי. כאשר (x,y) היא הנקודה השמאלית עליונה של המלבן, width הוא רוחבו ו-height הוא גובהו.

דוגמאות:

תמונה מעובדת (output)		תמונה מקורית (input)			
פרמטרים: x=1, y=1, width=2, height=1		1	2	3	4
6	7	5	6	7	8



3. ClockwiseRotate – התמרה המסובבת את התמונה ב-90 מעלות עם כוון השעון.

דוגמאות:

תמונה מקורית (input)	תמונה מעובדת (output)																
<table border="1"> <tr><td>1</td><td>2</td><td>3</td><td>4</td></tr> <tr><td>5</td><td>6</td><td>7</td><td>8</td></tr> </table>	1	2	3	4	5	6	7	8	<table border="1"> <tr><td>5</td><td>1</td></tr> <tr><td>6</td><td>2</td></tr> <tr><td>7</td><td>3</td></tr> <tr><td>8</td><td>4</td></tr> </table>	5	1	6	2	7	3	8	4
1	2	3	4														
5	6	7	8														
5	1																
6	2																
7	3																
8	4																

4. Segmentation – התמרה המצמצמת/מגבילה את מספר רמות הצבעים בתמונת הקלט למספר נתון (לדוגמה Segmentation לשתי רמות צבעים יוצרת תמונת שחור/לבן ללא גווני אפור). מספר רמות הצבעים הרצוי מועבר כפרמטר לבנאי (n).

כדי לבצע את ההתמרה יש לקבוע ספים המחלקים את מרחב הצבעים (0-255) ל-n קבוצות. צבעה של כל קבוצה ייקבע ע"י משוואה ליניארית הממפה את הקבוצה בה ערכי הצבעים הם הקטנים ביותר לצבע השחור (0) ואת הקבוצה בה ערכי הצבעים הם הגדולים ביותר לצבע הלבן (255).

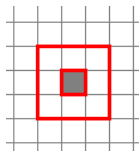
לדוגמה: עבור $n=2$, ייקבע ערך סף יחיד (127). כל צבע שערכו קטן מ-127 יצבע בשחור בתמונת הפלט, וכל צבע שערכו גדול מ-127 ייצבע בלבן בתמונת הפלט.

דוגמאות:

תמונה מקורית (input)	תמונה מעובדת (output)																								
<table border="1"> <tr><td>1</td><td>200</td><td>200</td><td>4</td></tr> <tr><td>5</td><td>196</td><td>140</td><td>8</td></tr> </table>	1	200	200	4	5	196	140	8	<p>פרמטרים: $n=2$</p> <table border="1"> <tr><td>0</td><td>255</td><td>255</td><td>0</td></tr> <tr><td>0</td><td>255</td><td>255</td><td>0</td></tr> </table> <p>פרמטרים: $n=3$</p> <table border="1"> <tr><td>0</td><td>255</td><td>255</td><td>0</td></tr> <tr><td>0</td><td>255</td><td>127</td><td>0</td></tr> </table>	0	255	255	0	0	255	255	0	0	255	255	0	0	255	127	0
1	200	200	4																						
5	196	140	8																						
0	255	255	0																						
0	255	255	0																						
0	255	255	0																						
0	255	127	0																						



5. Denoise – התמרה המנקה את התמונה מ"רעשים". נשתמש באלגוריתם הנקרא Median Filter לניקוי הרעשים (אנו מניחים כי רעשים הם פיקסלים בהירים או כהים במיוחד).
- א. נגדיר כ"רעש" פיקסלים בהירים או כהים מידי (ערכי סף מתאימים dark ו-bright הקובעים אם פיקסל כהה או בהיר, יועברו לבנאי).
- ב. לכל פיקסל "רועש" בתמונת המקור:
- נאסוף את כל הפיקסלים הסמוכים לו שאינם רועשים. פיקסלים סמוכים הם כל הפיקסלים שמרחקם באחת הקואורדינטות אינו עולה על n , פרמטר המועבר בבנאי. דוגמא, פיקסלים סמוכים עבור $n=1$:



- אם אוסף הפיקסלים ריק (כל הפיקסלים רועשים) נחזיר את הפיקסל המקורי.
- אחרת, נחזיר את הצבע הנתון ע"י החציון של אוסף הפיקסלים.

דוגמאות:

תמונה מקורית (input)						תמונה מעובדת (output)																	
<table border="1"> <tr> <td>190</td><td>255</td><td>190</td><td>100</td><td>100</td><td>100</td> </tr> <tr> <td>190</td><td>190</td><td>190</td><td>100</td><td>0</td><td>100</td> </tr> </table>						190	255	190	100	100	100	190	190	190	100	0	100	פרמטרים: dark=20, bright=249, n=1					
						190	255	190	100	100	100												
190	190	190	100	0	100																		
<table border="1"> <tr> <td>190</td><td>190</td><td>190</td><td>100</td><td>100</td><td>100</td> </tr> <tr> <td>190</td><td>190</td><td>190</td><td>100</td><td>100</td><td>100</td> </tr> </table>						190	190	190	100	100	100	190	190	190	100	100	100						
190	190	190	100	100	100																		
190	190	190	100	100	100																		

פרמטרים : dark=10, bright=245, n=5



ב ה צ ל ח ה