

# תכנות מונחה עצמים – תרגיל 3

תאריך ההגשה: 03.03.24 בשעה 23:55

תרגיל זה מומלץ להגיש בזוגות.

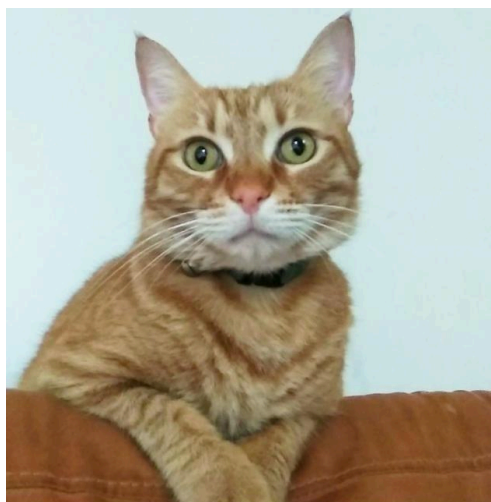
לדרישות התרגיל [לחצו פה](#).

## 0. הקדמה

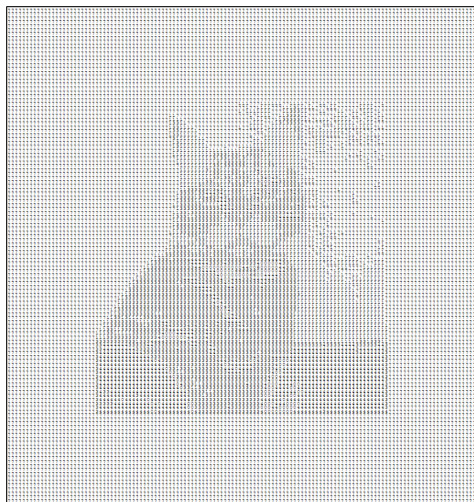
בתרגיל נבנה תוכנה הממירה קבצי תמונה לאמנות ASCII. התוכנה תקבל כקלט כתובת לקובץ תמונה, אוסף תווי ASCII בהם נרצה להשתמש בהמרה, וכן את הרזולוציה שבה נרצה להמיר את התמונה, ותמיר אותה לתמונת ASCII. הקלט יודפס כתמונה לקונסול או יישמר כקובץ html. נוסף על כך, נבנה ממשק משתמש.

דוגמה להמרה של תמונה בשני מקרים, 1. המרה עם תווי הספרות 0-9 הקיימים ברזולוציה של 128 תוויים בשורה 2. המרה עם כל התוויים בטבלת ASCII ברזולוציה 256.

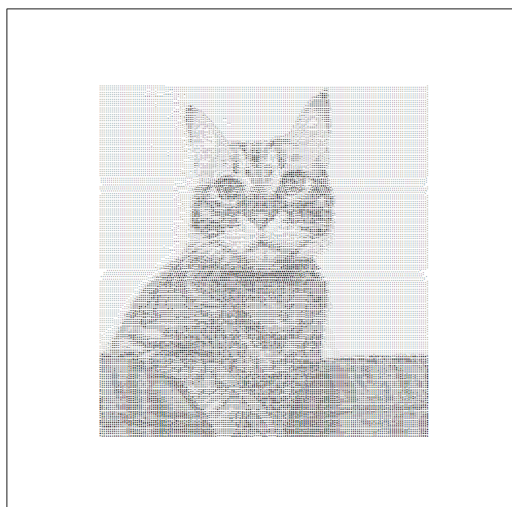
תמונת המקור



תמונת היעד מקרה 1



תמונת היעד מקרה 2

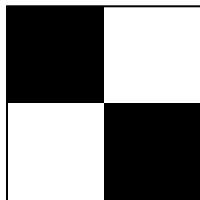


התרגיל מורכב משני חלקים: בחלק הראשון נפתח את אלגוריתם אומנות האסקי, ובחלק השני נפתח ממשק משתמש שבו יבחרו הפרמטרים להמרה ע"י המשתמש. יש להגיש את שניהם יחד במועד ההגשה. שימו ❤️: קבצי העזר לשימוש בתרגיל זה מופיעים ב-moodle. לפני שניגש לתרגיל עצמו, נציג בחלקים הבאים של ההקדמה כמה נושאי רקע שחשוב להכיר לפני תחילת העבודה על התרגיל.

## 0.1. ייצוג תמונות במחשב

כדי לייצג תמונות במחשב, אנחנו נחזיק מערך דו מימדי אשר כל תא במערך מייצג פיקסל בתמונה והוא מכיל מידע על הצבע של הפיקסל. הצבעים מורכבים משלושה צבעי יסוד – ירוק (G), אדום (R) וכחול (B) – ונקבעים על פי עוצמת ההארה של שלושת הצבעים הללו. כל עוצמת הארה תהיה בסולם של 0-255. לפיכך, נסמן כל צבע כשלושה של מספרים (R,G,B) כאשר כל מספר מסמן את עוצמת ההארה של צבע היסוד שלו. בשפת java, כל תא במערך שלנו יחזיק אובייקט של המחלקה Color, ולכל מופע של Color יהיו שדות של שלושת הצבעים (לא תצטרכו לממש מחלקה זו).

לדוגמא, השלושה (0,0,0) מייצגת צבע שחור, ואילו השלושה (255,255,255) מייצגת צבע לבן. כעת, אם נביט על התמונה הזו:



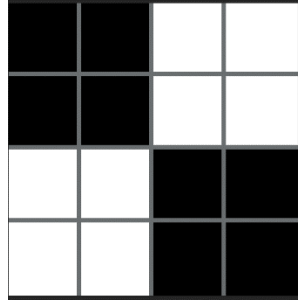
נוכל להציג אותה כמערך בגודל 2\*2:  
שורה ראשונה:

```
img[0][0] = Color(0,0,0), img[0][1] = Color(255,255,255)
```

שורה שנייה:

```
img[1][0] = Color(255,255,255), img[1][1] = Color(0,0,0)
```

באופן דומה, ניתן לייצג תמונה זו כמערך של 4\*4 פיקסלים:



מספר הפיקסלים שמייצגים את התמונה מגדירים את הרזולוציה שלה. ככל שהתמונה מיוצגת על ידי יותר פיקסלים, נגיד שהרזולוציה שלה גבוהה יותר, והיא תראה "חדה" יותר.

## 0.2. תווים

בג'אווה (וגם בשפות נוספות), char הוא משתנה פרימיטיבי המחזיק ערך מספרי כלשהו המייצג תו. תוכלו לראות את ההמרה בין התווים למספרים באמצעות חיפוש פשוט בגוגל של טבלת ASCII (לדוגמה, התו a מיוצג על ידי המספר 65). ככזה, ג'אווה מאפשרת לנו לבצע autocasting בין char ל-int כך שנוכל לבצע המרה בכיוון השני בצורה מפורשת. תוכלו להשתמש בתכונה הזו בעת כתיבת התרגיל.

## 0.3. ASCII ART

סוג אומנות שנוצרת מאותיות ascii, על ידי המרה של קבוצת פיקסלים לאות ascii שדומה לה בבהירות (ראה דוגמאות בתחילת ההקדמה). מס' הפיקסלים שמומרים בתו הוא פרמטר של התהליך; ככל שממירים מספר נמוך יותר של פיקסלים לאות, ככה נקבל תמונה שדומה יותר לתמונה המקורית. שימו לב שאין הכרח להשתמש בכל תווי ה-ascii, אלא ניתן להגדיר תת קבוצה של תווים שנשתמש רק בהם. בנוסף, שימו לב כי בכל גופן (font), לתו יש נראות מעט שונה, ולכן התמונה תיראה קצת שונה בגופנים שונים. הסבר ופירוט נוסף ניתן לראות ב[קישור לויקיפדיה](#).

## 1. חלק א': אלגוריתם אומנות ה-ASCII

בחלק זה של התרגיל נפתח את האלגוריתם בשלבים:

- 1.1. ראשית, נעבור במבט על על שלבי האלגוריתם לפני שנכנס לפרטים.
- 1.2. לאחר מכן נציג את עיצוב העל של התרגיל שהוא החלוקה לחבילות לפי תחומי האחריות של הקוד. את מבנה החלוקה של התרגיל לחבילות שונות אנחנו מספקים לכם, ובתוך שלד העיצוב הזה תידרשו לעשות בחירות עיצוביות מקומיות.
- 1.3. לאחר החלוקה לחבילות נציג את הקוד המסופק לכם שאותו אין צורך לממש.
- 1.4. לבסוף נציג בצורה מסודרת את דרישות התרגיל לצורך מימוש האלגוריתם.

## 1.1. שלבי האלגוריתם

### 1.1.1. שלב קדם - קביעת בהירות תווי ה-ASCII:

עבור כל תו בסט התווים שנבחר לצורך ההמרה, נחשב את הבהירות שלו (באמצעות נוסחה שתפורט בהמשך). את הבהירויות של התווים נשמור במבנה נתונים לבחירתכם. את השלב הזה נבצע פעם אחת עבור סט תווים נתון, ובעת הצורך נפנה למבנה הנתונים. הדגש שלנו בתרגיל הזה הוא גם על יעילות בזמני ריצה וזיכרון, לכן תצטרכו לחשוב על מבנה או מבני הנתונים המתאימים ביותר.

### 1.1.2. שלב א' - ריפוד התמונה:

בשלב הראשון נרפד את התמונה בפיקסלים לבנים (במידת הצורך) כדי שאורכה ורוחבה יהיו חזקות של 2 ונוכל לחלק אותה לריבועים.

### 1.1.3. שלב ב' - המרת התמונה לתתי תמונות כתלות ברזולוציה:

כל תמונה אפשר לייצג ברזולוציות שונות – נוכל להמיר כל פיקסל בודד לתו ASCII, אך ייתכן שנרצה להמיר ריבוע פיקסלים בגודל כלשהו לתו ASCII יחיד. הרזולוציה של ההמרה שלנו תוגדר ע"י מספר התווים שאנחנו רוצים להכניס בשורה בתמונת היעד, ובהתאם לכך, יוגדר גם מספר הריבועים שאליו תחולק התמונה כך שכל תת תמונה תוחלף בתו ASCII בודד. ככל שהתמונה המקורית תחולק ליותר תתי-תמונות כך הרזולוציה תהיה גבוהה יותר ותמונת ה-ASCII תראה דומה יותר לתמונה המקורית.

בהנתן רזולוציה שנבחרה ע"י המשתמש, נחלק את התמונה המקורית לתתי תמונות של ריבועים בגודל המתאים (כך שתתי התמונות מרכיבות יחד את התמונה המקורית).

### 1.1.4. שלב ג' - עבור כל תת תמונה המרה לתו בבהירות מתאימה:

עבור כל ריבוע (תת תמונה), נחשב את עוצמת הבהירות שלה (בנוסחה אותה נפרט בהמשך). לאחר מכן, נחליף את תת התמונה בתו ASCII עם הבהירות הדומה ביותר.

## 1.2. חלוקה לחבילות

חלוקה לחבילות היא שלב ראשון בתכנון העיצוב של התוכנה. אנחנו נחלק את התוכנה שלנו ל-4 חבילות (מודולים) לפני תחומי האחריות הבאים:

### 1.2.1. החבילה image:

תכיל את כל הלוגיקה של עיבוד התמונה. כפי שתואר בחלקים הקודמים, אנו רוצים לרפד את התמונה וכן לחלק את התמונה לתתי תמונות ולחשב ערך בהירות של תתי תמונות.

### 1.2.2. החבילה `ascii art`

החבילה שתהיה אחראית על הרצת האלגוריתם.

### 1.2.3. החבילה `image_char_matching`

החבילה שתהיה אחראית על התאמה של תת תמונה לתו בבהירות הכי קרובה. חבילה זו תשתמש בחבילה `image` בכל הקשור לעבודה עם התמונות.

### 1.2.4. החבילה `ascii_output`

חבילה שמטפלת בהצגת התמונה. חבילה זו מסופקת לכם במלואה.

## 1.3. הקוד המסופק (נמצא במודל [כאן](#))

### 1.3.1. המחלקה `KeyboardInput` בחבילה `ascii art`

המחלקה מכילה את המתודה `readLine` לצורך קבלת קלט מהמשתמש. נשתמש בה בחלק השני של התרגיל. (בדומה לתרגיל 1)

### 1.3.2. המחלקה `CharConverter` בחבילה `image_char_matching`

המחלקה מכילה מתודה סטטית פומבית אחת שממירה תו ASCII למערך בוליאני המייצג את תמונת התו בשחור לבן:

```
public static boolean[][] convertToBoolArray(char c)
```

בהינתן תו מסוים, המתודה תחזיר מערך בוליאני דו מימדי (בגודל 16X16). מערך זה מייצג את התמונה בשחור-לבן של התו בגופן `Courier New` (נעבוד רק עם הגופן הזה במהלך כל התרגיל) כאשר הערך `true` מייצג פיקסל לבן ו-`false` מייצג פיקסל שחור. בפונקציה זו נעזר כאשר נרצה לחשב את ערך הבהירות של התווים (שיהיה בעצם מספר הפיקסלים הלבנים מתוך כל הפיקסלים).

### 1.3.3. המחלקה `Image` בחבילה `image`

מחלקה המייצגת תמונה במחשב ע"י מערך דו מימדי של אובייקטים מסוג `Color` (כמו שמתואר בהקדמה). למחלקה יש שני בנאים:

1. בנאי המקבל שם של קובץ תמונה וטוען את הפיקסלים של התמונה לתוך המערך. הבנאי זה יכול לזרוק `IOException` אם יש בעיה עם קובץ התמונה. שימו לב שמי שקורא לבנאי צריך לדאוג שיש מי שתופס את החריגה ומטפל בה אם היא נזרקת.
2. בנאי המקבל מערך דו מימדי של `Color`, וכן את אורך ורוחב התמונה המיוצגת במערך.

למחלקה ארבע מתודות פומביות:

1. `getWidth` המחזירה את רוחב התמונה בפיקסלים
2. `getHeight` המחזירה את אורך התמונה בפיקסלים
3. `getPixel(int x, int y)` המחזיקה את הפיקסל  $(x, y)$  במערך התמונה.
4. `saveImage(String fileName)` השומרת את התמונה בקובץ בשם `fileName` עם הסיומת `.jpg`.

#### 1.3.4 החבילה `ascii_output`:

החבילה `ascii_output` מסופקת לכם במלואה כדי שתוכלו לייצא בקלות את הפלט של האלגוריתם בצורה הנוחה לכם. החבילה מכילה את הקבצים הבאים:

1. הממשק `AsciiOutput` המגדיר את המתודה `out` שבהינתן מערך תווים דו מימדי מייצא אותו לפלט בהתאם למימוש.
2. המחלקה `ConsoleAsciiOutput` שיודעת לייצא מערך תווים דו מימדי לקונסול (חלון ההרצה)
3. המחלה `HtmlAsciiOutput` שיודעת לייצא מערך תווים דו מימדי לקובץ בשם `out.html` כך שיהיה ניתן להציג את תמונת ה`ascii` בדפדפן ויהיה אפשר לעשות לה `zoom-in/zoom-out`.

שימו לב! אין לבצע שינויים בקוד של חבילה זו!

## 1.4 דרישות

למטה מתוארות דרישות התרגיל. בחלק מהדרישות לא ניתן לכם API ספציפי לממש, אלא רק הסבר על מה הקוד אמור לעשות. במקרים אלו, חישוב ראשית מה הקלט והפלט הרצויים, ובמידה וזה לא צויין – באיזו מחלקה/מודול (חדשים או קיימים) כדאי לממש את הדרישה. קחו בחשבון היכן משתלב הקוד הזה: מי קורא לו ולמי הוא קורא. לאחר מכן ממשו את הדרישות.

שימו :

- על המימוש להיות הגיוני ומתאים, ולעמוד בכללים והעקרונות הנלמדים בקורס.
- יש לקרוא את התרגיל עד הסוף כולל חלק ב' שבו מוסבר על ממשק המשתמש לפני מימוש הדרישות. חשוב מאוד לקבל את התמונה השלמה של התוצר הסופי כדי לבחור בחירות עיצוביות נכונות.
- תכננו את הקוד שלכם בעזרת דיאגרמת UML, אתם תדרשו להגיש את הדיאגרמה המתאימה לקוד שלכם בהגשת התרגיל ולהסביר את העיצוב שלכם ב-`README`.
- בשלב הראשון של התרגיל, נממש את האלגוריתם עם ערכי ברירת מחדל, ובשלב השני של התרגיל נוסיף את הממשק שבו המשתמש יוכל לשנות את הפרמטרים.

- בבירור המחדל התמונה איתה נעבוד תהיה התמונה "cat.jpeg" המסופקת לכם, נעבוד עם תווי הספרות 0-9 בלבד, ועם רזולוציה של 128, כלומר התמונה המקורית תחולק כך שבכל שורה יהיו 128 ריבועים, ובהתאם בכל שורה של התוצאה יהיו 128 תווים.

#### 1.4.1. שלב קדם: קביעת בהירות תווי ה-ASCII:

1.4.1.1. הוסיפו לחבילה image\_char\_matching את המחלקה SubImgCharMatcher שתהיה אחראית להתאים תו ASCII לתת תמונה עם בהירות נתונה. מחלקה זו תשמש את אלגוריתם אומנות ה-ASCII שלנו בהמשך כדי להחליף תתי תמונות בתווים.

לפני שנסביר איך לחשב את ערך הבהירות של תו ASCII, נציג ה-API של המחלקה:

1. `public SubImgCharMatcher(char[] charset)`

בנאי המקבל כפרמטר מערך של תווים שיהוו את סט התווים לשימוש האלגוריתם.

2. `public char getCharByImageBrightness(double brightness)`

בהינתן ערך בהירות brightness (של תת תמונה), המתודה תחזיר את התו (מתוך סט התווים) עם הבהירות הכי קרובה בערך מוחלט לבהירות הנתונה. שימו לב! אם יש מספר תווים מתוך הסט בעלי בהירות זהה, יוחזר התו עם ערך ה-ASCII הנמוך מביניהם.

3. `public void addChar(char c)`

מתודה שמוסיפה את התו c לסט התווים.

4. `public void removeChar(char c)`

מתודה שמסירה את התו c מסט התווים.

5. מתודות פומביות נוספות לפי הצורך. אם תוסיפו מתודות ל-API, תתבקשו להסביר את הסיבה לכך ב-README.

המימוש של המחלקה נתון לשיקול דעתכם. תוכלו כמובן להוסיף למחלקה שדות ומתודות פרטיים לפי הצורך, וכן להוסיף מחלקות נוספות לחבילה image\_char\_matching שישמשו את המחלקה SubImgCharMatcher. חשבו מראש באילו מבני נתונים תרצו לשמור את הנתונים והביאו בחשבון שיקולי יעילות וזיכרון. אתם תתבקשו להסביר ב-README באילו מבני נתונים השתמשתם ומדוע.

#### 1.4.1.2. כעת נסביר איך לחשב את ערך הבהירות של תו ASCII:

עבור כל תו בודד:

1. ראשית נמיר את התו למערך דו מימדי בוליאני בגודל 16X16 בעזרת המתודה `CharConverter.convertToBoolArray` המסופקת (ר' לעיל). לאחר מכן נספור את ערכי התאים בהם הערך הוא true (אינדיקציה שזהו פיקסל בצבע לבן). שימו לב- בתרגיל הזה

תמיד נרנדר את התווים לפי רזולוציה של 16 פיקסלים בלי תלות ברזולוציה של התמונה ושל תת התמונות.

2. ננרמל את המספר שקיבלנו בסך מספר הפיקסלים במערך על מנת לקבל מספר בין 0-1 המייצג את רמת הבהירות של התו.

לאחר שחישבנו את ערך הבהירות של כל התווים:

3. ננרמל את כל ערכי הבהירות של התווים שחישבנו ע"י מתיחה ליניארית של ערכי הקצוות בסט התווים הנתון. זאת על מנת למנוע מצב שבתמונה בעלת גוונים דומים נקבל את אותה האות לכל פיקסל בתמונה. המתיחה מחדדת לנו את הבדלי הבהירות בין האותיות השונות וכך אנחנו נמנעים ממצב כזה. את המתיחה נבצע לפי החישוב הבא:

$$newCharBrightness = \frac{charBrightness - minBrightness}{maxBrightness - minBrightness}$$

- שימו לב שכדי לבצע את שלב הנרמול (3) יש צורך לחשב קודם כל את ערכי הבהירות של כל התווים בסט התווים כדי למצוא את הבהירות המינימלית ואת הבהירות המקסימלית מתוך כל התווים.
- הניחו כי לא תתבקשו לנרמל סט עם תו אחד בלבד (המכנה מתאפס)

#### 1.4.2. שלב א'ב': ריפוד התמונה וחלוקת התמונה לתתי תמונות:

עבור שלבים אלו לא נדרוש מכם API ספציפי, והעיצוב נתון לשיקולכם. עליכם להוסיף את הפונקציונליות המפורטת מטה ע"י הוספת מחלקה או מחלקות מתאימות לחבילה image.

1.4.2.1. ריפוד תמונה בפיקסלים לבנים כך שהיא תהיה באורך וברוחב של חזקה של 2 (זו שגדולה אך קרובה ביותר למקורי, התמונה לא צריכה להפוך לריבוע).  
למשל: תמונה שגודלה 30x14 נגדיל את התמונה להיות 32x16 בעזרת ריבועים לבנים בקצוות.

- הריפוד צריך להיות סימטרי בין הצדדים השונים (אפשר להניח שהמימדים של הקלט תמיד זוגיים).
- אם אחד המימדים של התמונה הוא חזקה של 2, אין צורך לרפד אותו.

1.4.2.2. חלוקה של תמונה לתת תמונות לפי רזולוציה נתונה (כך שתתי התמונות מרכיבות יחד את התמונה המקורית והן לא חופפות). הקלט יהיה הרזולוציה הרצויה, כלומר מספר תתי התמונות בשורה, והפלט יהיה מבנה נתונים המכיל את כל תתי התמונות של התמונה (כל תת תמונה היא ריבוע של פיקסלים). בהמשך כל תת תמונה שכזו תומר לתו ASCII. המימוש הספציפי הוא כראות עיניכם.

#### 1.4.2.3. חישוב בהירות של תמונה/תת תמונה.

- ראשית יש להמיר כל פיקסל בתמונה לגוון אפור. כדי לעשות זאת, נמיר כל אובייקט מסוג Color למספר בין 0-255. לצורך ההמרה נשתמש בנוסחה הבאה:



```
greyPixel = color.getRed() * 0.2126 + color.getGreen() * 0.7152  
+ color.getBlue() * 0.0722
```

- לאחר מכן יש לסכום את כל גווני האפור של הפיקסלים ולחלק בסך הפיקסלים בתת-התמונה ובציון ה-RGB המקסימלי 255. (על מנת לנרמל את הגוון האפור להיות בין 0-1).

### 1.4.3. שלב ג': מימוש אלגוריתם ה-ASCII ART עצמו

הוסיפו את המחלקה `AsciiArtAlgorithm` לחבילה `ascii_art`. מחלקה זו תהיה אחראית על הרצת האלגוריתם. הפרמטרים עבור ריצת האלגוריתם יתקבלו בבנאי של המחלקה. הריצה תהיה תלויה כמובן בתמונה שנבחרה, ברזולוציה שנבחרה (כמה תווים יהיו בשורה בתוצאה), וכן בסט התווים שנבחר. הדרך שבה נתונים אלו יתורגמו לפרמטרים של המחלקה תלוי במימוש שלכם לחלקים הקודמים. למחלקה תהיה מתודה פומבית אחת בלבד:

```
public char [][] run()
```

המתודה מחזירה מערך דו מימדי של תווים (ASCII) המייצגים תמונה (אותה תמונה שהתקבלה בבנאי). כל תת תמונה תוחלף בתו עם רמת הבהירות הקרובה ביותר (בערך מוחלט).

## 1.5. הערה על יעילות האלגוריתם

נשים לב כי ניתן לייעל את הקוד כדי לחסוך בזמן ריצה של חישובים חוזרים בהרצות חוזרות של האלגוריתם בכמה היבטים:

1.5.1. נרצה לנסות ולחסוך בחישוב הבהירות של כל תת תמונה. למעשה, אם אנחנו מתכוונים לקרוא שוב לאלגוריתם עבור אותה תמונה ואותה רזולוציה, נוכל לשמור את ערכי הבהירות שחושבו בריצה הראשונה ולהשתמש בהם בפעמים הבאות שנרוץ על התמונה.

1.5.2. נרצה לנסות ולחסוך בחישוב הבהירות של תווי האסקי שנבחרו לסט התווים, שכן הבהירות של התווים (לפני הנרמול) לא משתנה בין ריצה לריצה.

1.5.3. נרצה לנסות ולחסוך בחישוב הבהירות המנורמלת של התווים אם סט התווים לא השתנה בין ריצה לריצה. שימו לב שאם סט התווים כן השתנה, ייתכן והבהירות המנורמלת של תו מסויים תשתנה אם ערכי המינימום/המקסימום של הבהירות בסט השתנו בעקבות הוספה או מחיקה של תווים מהסט.

מצופה מכם לייעל את הקוד לפי מנגנון שעושה שימוש במבני נתונים כדי לחסוך חישובים חוזרים. אתם תידרשו להסביר את הבחירה שלכם בקובץ ה-README בהתבסס על סיבוכיות זמן הריצה והזיכרון של המבנים בהם השתמשתם. בתרגיל הזה אתם רשאים להשתמש בספריית ה-`collections` או כל מבנה נתונים אחר שמסופק בספריות הסטנדרטיות של ג'אווה.

## 1.6. בדיקת שפיות

הריצו את הקוד שלכם עם הפרמטרים הבאים:

- התמונה תהיה "board.jpeg" המסופקת לכם בקבצים
- סט התווים יהיה 's','m'
- הרזולוציה תהיה 2 תווים בשורה

ובדקו שהפלט הוא:

```
[[m, o], [o, m]]
```

בדיקות נוספות תוכלו לערוך מול [הדוגמאות](#) של תמונת החתול המובאות לכם בהקדמה עם הפרמטרים המתאימים. כמובן שבדיקות אלו לא מספיקות כדי לוודא שהקוד שלכם תקין, אבל הן מקום טוב להתחיל ממנו.

## 2. חלק ב' - ממשק משתמש

בחלק זה ניצור ממשק משתמש שבו המשתמש יוכל לבחור ולשחק עם הפרמטרים של אלגוריתם ה-ASCII art. התוכנית תתחיל עם ערכי ברירת המחדל שהזכרנו בחלק 1.4: התמונה תהיה התמונה "cat.jpeg", סט התווים יכיל את תווי הספרות 0-9 בלבד, והרזולוציה תהיה 128. בנוסף, ברירת המחדל הפלט שלנו יודפס למסך.

**שימו לב!** יש למקם את הקובץ cat.jpeg כ"אח" של התיקיה src (שניהם יחד באותה תיקיית אב), ובתוך הקוד להשתמש בנתיב היחסי (relative path) אל התמונה כך שהוא יהיה פשוט שם התמונה cat.jpeg.

הפקודות הבסיסיות שאנחנו רוצים לייצר לממשק שלנו הן:

1. [exit](#) - יציאה מהתוכנה.
2. [chars](#) - צפייה במאגר התווים הנוכחי.
3. [add](#) - הוספה תווים לסט התווים הנוכחי.
4. [remove](#) - הסרה של תווים מסט התווים הנוכחי.
5. [res](#) - שליטה ברזולוציה של התמונה שלנו – הגדלה והקטנה של הרזולוציה.
6. [image](#) - בחירת קובץ תמונה.
7. [output](#) - בחירת ה-Output – אם התוצר יודפס ל-console או לחילופין יוחזר כקובץ html
8. [asciiArt](#) - הרצת האלגוריתם על הפרמטרים הנוכחיים.

### 2.1 יצירת ממשק המשתמש

ראשית, נייצר מחלקה חדשה בשם Shell בחבילה ascii\_art. המחלקה תבצע את כל פעולות ממשק המשתמש. ה-API של המחלקה יכיל מתודה פומבית אחת בלבד:

```
public void run()
```

אשר תפקידה לתרגם את הפקודות המתקבלות מהמשתמש ולבצע אותן.

שימו לב, מומלץ לבדוק אחרי כל הוספה של פונקציונליות כי הממשק אכן עובד.

### 2.1.1. שימוש במנגנון החריגות:

כאשר הקוד נתקל בשגיאה ויש צורך להדפיס הודעה למשתמש, יש להשתמש במנגנון החריגות של java (exceptions). זרקו חריגה מתאימה מהמקום בקוד שבו התגלתה השגיאה, ותפסו את כל החריגות בפונקציה run שם יש להדפיס את ההודעה המתאימה לפי ההוראות בהמשך. השתמשו בחריגות בהתאם לעקרונות שנלמדו בהרצאה ובתרגול, וצרו מחלקות חריגה חדשות לפי הצורך. אתם תתבקשו להסביר ב-README שלכם על הדרך בה הקוד שלכם מתמודד עם טיפול בחריגות, פרטו על מטרת המחלקות שהוספתם.

## 2.2. קבלת input מהמשתמש ויציאה

הוסיפו פונקציונליות למתודה run כך שכאשר יקראו למתודה, תודפס למסך המחרוזת

```
>>>
```

והמשתמש יוכל לכתוב לתוכה.

**שימו לב!** השתמשו רק במחלקה `KeyboardInput` המסופקת בקוד לקבלת קלט מהמשתמש (כמו בתרגיל 1).

בשלב זה, כאשר המשתמש ילחץ `enter`, נאפשר את שתי האופציות הבאות:

1. לכל מחרוזת השונה מהמחרוזת `exit` נדפיס שוב את `>>>` וחוזר חלילה.
2. כאשר המשתמש יקליד את המחרוזת `exit` נסיים את התוכנית.

## 2.3. צפייה במאגר התווים

הוסיפו למתודה `run` אפשרות שכאשר המשתמש יזין את המחרוזת `chars` נדפיס ל-`console` את כל התווים שאנחנו יכולים להשתמש בהם בבניית התמונה. ההדפסה תהיה לפי סדר `ascii`, כל התווים בשורה אחת עם רווח בודד בין תו לתו.

אנא אתחלו את התוכנית כך שבאופן דפולטיבי מאגר התווים יכיל את המספרים 0-9.

## 2.4. הוספה למאגר התווים

הוסיפו למתודה `run` את האפשרות להוסיף תווים באמצעות המחרוזת `add`. הפקודה תאפשר להכניס תווים באמצעות שלושת האפשרויות הבאות:

2.4.1. הפקודה `add` ולאחריה תו בודד (מחרוזת באורך 1) – נכניס למאגר שלנו את התו. לדוגמא הפקודה:

```
add a
```

(עלינו להכניס למאגר את התו `a`)

#### 2.4.2. הפקודה add ולאחריה המילה all .

עבור המילה all נכניס למאגר שלנו את כל התווים האפשריים ליצירת התמונה - מהתו ' ' (רווח) עד התו ~. כלומר, מתו ASCII שמספרו 32 עד 126, כולל.  
לדוגמא הפקודה:

```
add all
```

#### 2.4.3. הפקודה add ולאחריה אחת המילה space. עבור המילה space נכניס למאגר שלנו את התו רווח ' ' . לדוגמא הפקודה:

```
add space
```

#### 2.4.4. הפקודה add ולאחריה טווח תווים (כל סוג טווח במסגרת תווי ascii, אותיות גדולות, קטנות, מספרים וכדו') מהצורה a-d – נכניס למאגר שלנו את כל התווים בטווח a-d. למשל הפקודה:

```
add m-p
```

(עלינו להכניס למאגר את התווים m,n,o,p)

שימו לב שעליכם להתמודד גם עם קלט הפוך, למשל עבור הפקודה:

```
add p-m
```

תצטרכו להכניס למאגר את התווים m,n,o,p.

#### 2.4.5. הפקודה add ולאחריה כל דבר אחר שלא מפורט לעיל, יחשב כקלט לא תקין ומצופה מכם להדפיס למסך הודעת שגיאה:

```
Did not add due to incorrect format.
```

שימו לב! במידה והמשתמש מבקש להוסיף תו שכבר קיים במאגר, אין צורך להדפיס הודעת שגיאה ויש פשוט להתעלם מהבקשה.

## 2.5. מחיקה ממאגר התווים

הוסיפו לשיטה run את האופציה להסיר תווים מהמאגר באמצעות המחרוזת **remove** אשר כמו add גם תקבל את כל אחת מהאפשרויות לעיל אבל תסיר מהמאגר את האותיות הנבחרות. גם כאן, במקרה של קלט לא תקין לאחר הפקודה remove, הדפיסו הודעת שגיאה.

```
Did not remove due to incorrect format.
```

ובמקרה של הסרת תו שאינו במאגר – אין צורך להדפיס הודעת שגיאה ויש פשוט להתעלם מהבקשה.

## 2.6. שליטה ברזולוציה של התמונה שלנו

הוסיפו לשיטה run את האופציה להגדיל או להקטין את הרזולוציה של התמונה באמצעות המחרוזת **.res**. פקודה זו יכולה לקבל אחת משתי מחרוזות up, down:

2.6.1. עבור הפקודה `res up` נכפיל את מספר התווים בשורה ב-2.

2.6.2. עבור הפקודה `res down` נחלק את מספר התווים בשורה ב-2.

2.6.3. הדפיסו לאחר השינוי את מספר התווים העדכני למסך באופן הבא:  
`Resolution set to <current resolution>.`

2.6.4. אתחלו את הרזולוציה הדיפולטיבית, כלומר את מספר התווים ההתחלתיים בשורה, להיות 128.

2.6.5. לכל תמונה, בהתאם לנתוני הרוחב והגובה שלה צריכים להיות ערכים שהם המינימום והמקסימום של מספר תתי-התמונות הריבועיות שאפשר להכניס בשורה, שזה בעצם הרזולוציה המדוברת. המקסימום הוא מספר הפיקסלים ברוחב התמונה. המינימום יהיה תלוי ביחס הרוחב והגובה של התמונה. ניתן לחשב אותו לפי הנוסחה הבאה:

`minCharsInRow = max(1, imgWidth/imgHeight)`

אם בעקבות הפקודה עברתם את הערך המקסימלי/מינימלי, אין לשנות את הרזולוציה, ויש להדפיס את הודעת השגיאה הבאה:

`Did not change resolution due to exceeding boundaries.`

בהתאם.

2.6.6. הפקודה `res` ללא פרמטר נוסף מהווה קלט לא תקין. בכל מקרה של קלט לא תקין לאחר המחרוזת `res` יש להדפיס הודעת שגיאה:

`Did not change resolution due to incorrect format.`

## 2.7. שינוי קובץ התמונה

הוסיפו לשיטה `run` את הפקודה `image` שמשנה את תמונת המקור שלנו. אחרי הפקודה `image` יש להכניס את שם הקובץ החדש (עם נתיב אבסולוטי או רלטיבי), למשל:

`image board.png`

אם מסיבה כלשהי התמונה החדשה לא נטענת בהצלחה יש להדפיס את הודעת השגיאה:

`Did not execute due to problem with image file.`

שימו לב! הבנאי של `Image` זורק חריגה מסוג `IOException`. עליכם לדאוג לתפוס אותה במקום המתאים.

## 2.8. שינוי שיטת ה-output

הוסיפו לשיטה `run` את הפקודה `output` שמשנה את יעד הפלט שלנו. פקודה זו יכולה לקבל אחת משתי מחרוזות `html`, `console`. באופן דיפולטיבי, ה-`output` שלנו יהיה הדפסה למסך (`console`).

2.8.1. עבור הפקודה `output html` יש לדאוג לכך שהפלט של האלגוריתם בריצות הבאות יכתב לקובץ בשם "out.html" עם הגופן "Courier New".

2.8.2. עבור הפקודה `output console` יש לדאוג לכך שהפלט של האלגוריתם בריצות הבאות יודפס למסך.

2.8.3. הפקודה `output` ללא פרמטר נוסף מהווה קלט לא תקין. בכל מקרה של קלט לא תקין לאחר המחרוזת `output` יש להדפיס הודעת שגיאה:

```
Did not change output method due to incorrect format.
```

## 2.9. הרצת האלגוריתם

הוסיפו לשיטה `run` את הפקודה `asciiArt` אשר מריצה את אלגוריתם ה-ASCII art על פי הפרמטרים המוגדרים כאשר קוראים לה (תמונה, רזולוציה, מאגר תווים). בסיום הריצה הפלט יודפס ליעד שנבחר `console` או `html`.

במקרה של קריאה ל-`asciiArt` עם מאגר תווים ריק הדפיסו הודעת שגיאה:

```
Did not execute. Charset is empty.
```

## 2.10. התמודדות עם קלט לא תקין

שנו את המתודה `run()` כך שכאשר היא תקבל פקודה לא חוקית (כלומר לא נמצאת ברשימת 8 הפקודות) תודפס הודעת שגיאה:

```
Did not execute due to incorrect command.
```


אם הפקודה עצמה חוקית אבל הפרמטרים לא טובים, הדפיסו הודעת שגיאה ספציפית לפקודה כמפורט לעיל עבור כל פקודה.

למשל: עבור `add aaa` הדפיסו הודעת שגיאה של ההוספה

```
Did not add due to incorrect format.
```

אבל עבור קלט `add` הדפיסו את הודעת השגיאה הכללית:

```
Did not execute due to incorrect command.
```

שימו : השתמשו ב-`system.out.println` על מנת להדפיס את הודעות השגיאה, ההודעה תודפס בשורה אחת בדיוק עם ירידת שורה אחת בלבד בסיום ההדפסה. אין צורך להוסיף `\n`.

## 3. הוראות הגשה

### 3.1. קבצים להגשה

הגישו את התרגיל כקובץ `jar/tar/zip` בשם `ex3` (והסיומת בהתאם). הקובץ יכיל את הקבצים הבאים:


1. החבילה `ascii_art` הכוללת את הקבצים

- `AsciiArtAlgorithm.java`
  - `Shell.java`
  - `KeyboardInput.java` - אין להגיש קובץ זה
  - קבצים נוספים בהתאם לעיצוב שבחרתם.
2. החבילה `image` הכוללת את הקובץ

- `Image.java`
  - קבצים נוספים בהתאם לעיצוב שבחרתם.
3. החבילה `image_char_matching` הכוללת את הקבצים:

- `CharConverter.java`
  - `SubImgCharMatcher.java`
  - קבצים נוספים בהתאם לעיצוב שבחרתם.
4. קובץ בשם **README** (ללא סיומת) לפי ההוראות בסעיף 3.2.

5. קובץ **UML.pdf** המכיל דיאגרמת UML המתארת את הקוד שלכם.  
יש להגיש את הקובץ בפורמט PDF ולוודא שהדיאגרמה נכנסת במלואה בקובץ `submission.pdf`  
הנוצר בהגשה!

שימו : אין להגיש את החבילה `ascii_output`.

## 3.2 קובץ README

צרכו להגשה שלכם קובץ README הכולל את הפרטים הבאים:

- בבשורה הראשונה בקובץ יופיע שם המשתמש *CSE* שלכם.  
אם אתם מגישים בזוג, יש להפריד בין שמות המשתמש עם פסיק (שניהם באותה שורה).
- בשורה השניה מספר תעודת הזהות.  
אם אתם מגישים בזוג, יש להפריד בין מספרי הזהות עם פסיק (שניהם באותה שורה).
- השורה השלישית ריקה.
- **החל מהשורה הרביעית ענו בקובץ על השאלות הבאות לפי הסדר:**

1. בהתאם לדיאגרמת UML של הקוד שיצרתם, הסבירו בקצרה מה תפקידה של כל מחלקה בקוד שלכם ועל הקשרים בין המחלקות.
2. הסבירו בקצרה עבור כל מבנה נתונים של `java` בו השתמשתם בקוד, היכן השתמשתם בו ולמה בחרתם דווקא במבנה נתונים זה עבור המשימה. התייחסו בהסבר לשיקולי סיבוכיות זמן ריצה וזיכרון.

3. הסבירו איך השתמשתם במנגנון החריגות של java על מנת לטפל בשגיאות בקלט המשתמש והדפסת הודעות שגיאה.

4. אם הכנסתם שינויים ב-API של המחלקה SubImgCharMatcher הסבירו למה השינויים היו נצרכים בקוד.

5. אם הכנסתם שינויים בקוד של המחלקות שסופקו לכם הסבירו את השינויים ולמה הם היו נצרכים. אין להכניס שינויים בקוד של המחלקות בחבילה `ascii_output` שכן לא מגישים חבילה זו!

### 3.3. דוגמה לקובץ פלט

סיפקנו לכם בתיקיית חומר העזר קבצי [קלט ופלט](#) לדוגמת החתול עבור מקרים 1 ו-2 מתוך ההקדמה לתרגיל. הקובץ הראשון בשם `out1` התקבל עבור רזולוציה 128 עם התווים 0-9 והקובץ השני בשם `out2` התקבל עבור רזולוציה 256 עם כל התווים האפשריים. תוכלו להשתמש בקבצים אלו לצורך בדיקה עצמית.

**בהצלחה!**