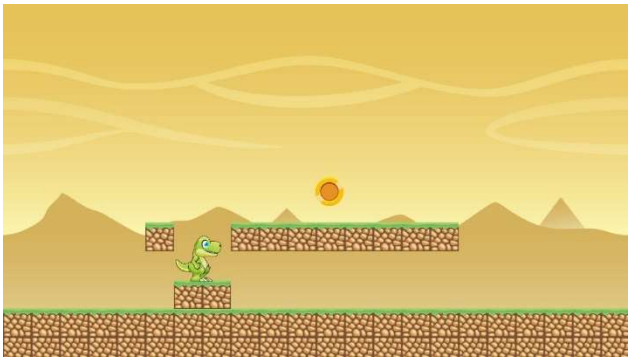
ASCII Art

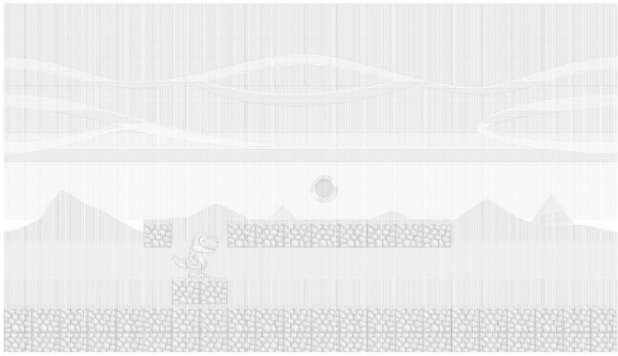
תרגיל 4, חלק א'

קבצי העזר לשימוש בתרגיל זה מופיעים בmoodle.

בתרגיל הבא נבנה תוכנה הממירה קבצי תמונה לאמנות ASCII. אנחנו נכתוב תוכנה שמקבלת כקלט כתובת לקובץ תמונה, וממירה אותה לתמונת ASCII. הקלט יודפס כתמונה לקונסולה או יישמר כקובץ html.

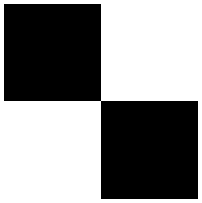
למשל:





כאשר אנחנו מתעסקים בתמונות, אנחנו מדברים על מערך דו ממדי אשר כל תא במערך מיצג פיקסל בתמונה והוא מכיל מידע על הצבע של הפיקסל. כפי שאנחנו יודעים, הצבעים במחשב מורכבים משלושה צבעי יסוד (ירוק, אדום וכחול) ונקבעים על פי עוצמת ההארה של שלושת הצבעים הללו. כל עוצמת הארה תהיה בסולם של 0-255. נסמן כל צבע כשלשה של מספרים (R,G,B) כאשר כל מספר מסמן את עוצמת ההארה של צבע היסוד שלו.
בפועל, כל תא במערך שלנו יחזיק אובייקט מסוג Color כאשר לכל מופע של Color יהיו שדות כנגד שלושת הצבעים (לא תצטרכו לממש אובייקט זה).

נשים לב כי על פי ההסבר לעיל, השלשה (0,0,0) מייצגת צבע שחור, ואילו השלשה (255,255,255) מייצגת צבע לבן.
כעת, אם נביט על התמונה הזו:



נוכל להציג אותה כמערך בגודל 2*2:

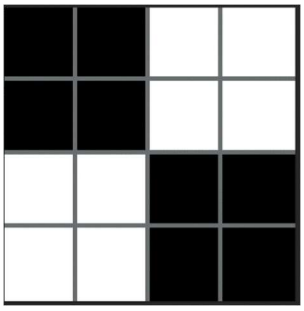
שורה ראשונה:

```
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.

כדי להמיר תמונה צבעונית לתמונה בגווני אפור, נרצה לגרום לכל תא במערך שלנו להכיל מספר בין 0-255 במקום אובייקט מסוג Color. לצורך ההמרה נשתמש בנוסחה הבאה:

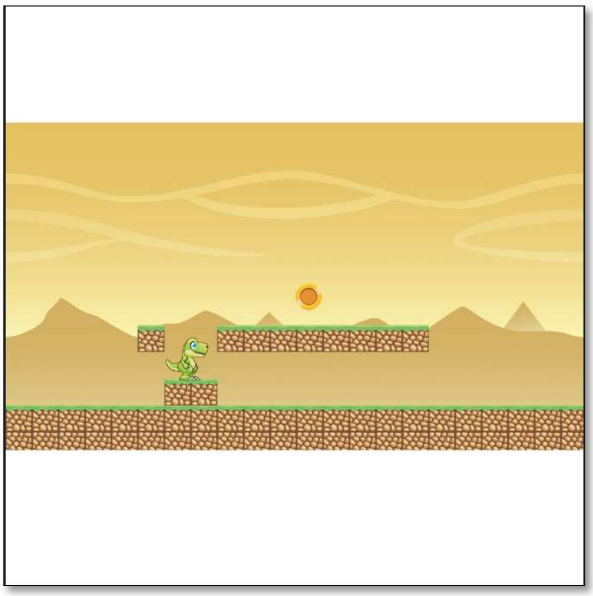
```
greyPixel = color.getRed() * 0.2126 + color.getGreen() * 0.7152 +
color.getBlue() * 0.0722
```

<u>האלגוריתם בקווים כלליים:</u>

האלגוריתם שלנו עובד בשלושה שלבים:

א. המרת התמונה לתתי תמונות לפי הרזולוציה שאנחנו מעוניינים בה.
ב. קביעת בהירות תווי ה-ASCII שבאמצעותם נרצה לצייר את התמונה.
ג. המרת כל תת תמונה לתו בבהירות המתאימה לה.

<u>שלב א:</u> כל תמונה אפשר לייצג ברזולוציות שונות – נוכל להמיר כל פיקסל לתו ASCII, אך ייתכן שנרצה להמיר ריבוע פיקסלים לתו ASCII יחיד. לכן, השלב הראשון הוא להחליט באיזו רזולוציה אנחנו רוצים להמיר את התמונה שלנו ואז בהתאם לכך, לחלק את התמונה המקורית לריבועים בגודל המתאים. ככל שנחלק את התמונה ליותר תתי-תמונות כך הרזולוציה תהיה גבוהה יותר ותמונת ה-ASCII תראה דומה יותר לתמונה המקורית.



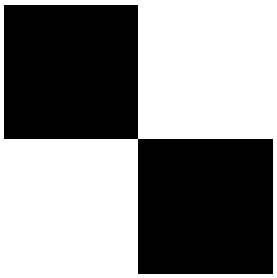
בתרגיל שלנו, כבר סיפקנו לכם את אלגוריתם ההמרה לתתי תמונות. למען הנוחות, האלגוריתם שלנו מחייב את תתי התמונות להיות בעלות צלע באורך חזקה של 2. כדי שהתמונה תמיד תתחלק לריבועים בגדלים שווים - האלגוריתם ירפד, במקרה הצורך, את התמונה בפיקסלים לבנים עד שאורכי הצלעות שלה יהיו בגודל הרצוי.

<u>שלב ב':</u> עבור כל תו נחשב את הבהירות שלנו (באמצעות נוסחה שתפורט בהמשך).

<u>שלב ג':</u> עבור כל ריבוע (תת תמונה), נחשב את עוצמת הבהירות שלה (בנוסחה אותה נפרט בהמשך). במקביל, נחשב לכל תו ASCII את הבהירות שלו, ואז נחליף את תת התמונה בתו ASCII עם בהירות דומה.

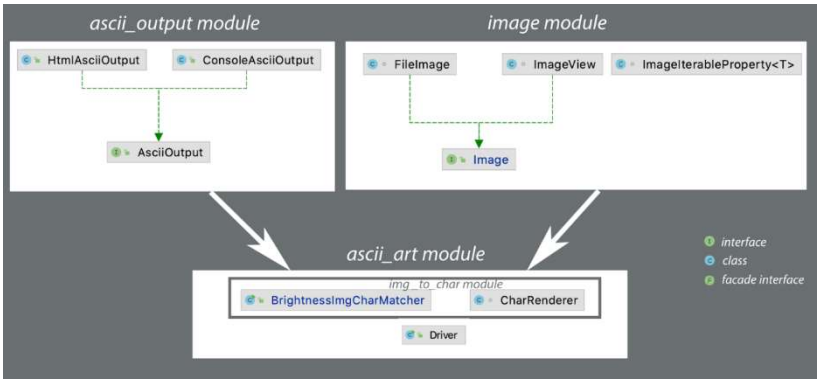
<u>בואו נתחיל...</u>

בתרגיל הזה נממש גרסה פשוטה של הלוגיקה הזו, ונפעיל אותה על קלט ספציפי - תמונת ריבועי שחור-לבן בגודל 64*64 פיקסלים אשר מצורפת לכם בקבצי העזר:



את החלוקה למודולים שונים כבר ביצענו עבורכם ותוכלו לראות אותם בקבצי העזר המצורפים:

1. מודול Image – מכיל את כל הלוגיקה של טיפול בתמונה. כפי שתואר בחלקים הקודמים, אנו רוצים לחלק את התמונה לתתי תמונות ולבצע עליהם כל מיני מניפולציות. המודול הזה יכיל את כל המחלקות התומכות בפונקציונליות הזו.
2. מודול ImgToChr – מודול המממש את כל הלוגיקה של המרת תמונה לתו. מודול זה ישתמש במודול Image בכל הקשור לעבודה עם התמונות.
3. מודול ASCIIOutput - מודול שיודע לטפל בהצגת התמונה. כאן נכניס את כל הלוגיקה המציגה את התמונה למשתמש.
4. מודול ASCIIArt - המודול שיכיל את לוגיקת המנהל (מחלקה בשם Driver) אשר תנהל ותציג את התמונה הגדולה.



<u>אז מה בעצם אתם צריכים לממש?</u>

Driver:

ראשית, הוסיפו למודול ascii_art את המחלקה Driver אשר השיטה היחידה בה תהיה main:

```
public static void main(String[] args)
```

בחלק זה של התרגיל תוכלו להשתמש בשיטה זו לצרכי דיבוג. היא לא תיבדק (מבחינתנו הגישו אותה ריקה). יהיה לה שימוש בחלק ב' של התרגיל.

CharRenderer:

לפני שנתחיל במלאכת המימוש, כדאי שנכיר את המחלקה הסטטית CharRenderer אותה סיפקנו לכם כחלק מקבצי העזר. ה-API של המחלקה כוללת שיטה אחת:

```
Public static boolean[][] getImg(char c, int pixels, String fontname)
```

בהינתן תו מסוים, מספר פיקסלים (נסמן מספר זה ב-n) ושם פונט, השיטה תחזיר מערך בוליאני דו מימדי (בגודל nxn). מערך זה מייצג את התמונה בשחור-לבן של התו כאשר הערך true מייצג פיקסל לבן ו-false מייצג פיקסל שחור.

אתם נדרשים להוסיף שיטה למחלקה CharRenderer בשם printBoolArr המקבלת מערך דו מימדי בוליאני (המייצג תו) ומדפיסה אותו לקונסולה. הקפידו שחתימת הפונקציה תהיה:

```
public static void printBoolArr(boolean[][] arr)
```

BrightnessImgCharMatcher:

צרו מחלקה חדשה בשם הזה תחת המודול img_to_char.

המחלקה הזו צריכה להבטיח את השיטה:

```
public char[][] chooseChars(int numCharsInRow, Character[] charSet)
```

charSet – סט התווים באמצעותם נרצה לצייר את התמונה שלנו.

numCharsInRow – כמה תווים נצייר בכל שורה בתמונת הASCII.

השיטה מחזירה מערך דו מימדי של תווים (ASCII) המייצגים תמונה (אותה תמונה שהתקבלה בבנאי).

ניתן להניח כי המספר numCharsInRow יחלק את רוחב התמונה למספר שלם.

יש מספר דרכים לבחור כיצד להמיר תמונה ל-ASCII. אנחנו נממש מחלקה שממירה בעזרת חישוב בהירויות ומכאן שמה.
בשלב הראשון **נמיין את התווים במערך לפי רמת הבהירות שלהם.** שימו לב כי בכל פונט, לתו יש נראות מעט שונה. לכן, כאשר נדבר על בהירות של תו היא תהיה תלויה בפונט שבו הוא יוצג.
כדי לחשב בהירות של תו בפונט אריאל, נמיר את אותו התו למערך דו ממדי בוליאני בעזרת:

```
CharRenderer.getImg('c', 16, "Ariel")
```

נספור את ערכי התאים בהם הערך הוא true (אינדיקציה שזהו פיקסל בצבע לבן) ואז נחלק את הסכום בסך מספר הפיקסלים על מנת לקבל מספר בין 0-1 המייצג את רמת הבהירות של התו.
שימו לב, בתרגיל הזה תמיד נרנדר את התווים(!!) לפי רזולוציה של 16 פיקסלים. התמונה עצמה תוכל להיות ברזולוציה שונה.

בשלב השני, **נבצע מתיחה ליניארית על המערך שלנו** על מנת לנרמל את ערכי הבהירות של התווים ע"י מתיחה של ערכי הקצוות. 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:

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

הנחת המוצא שלנו היא שהתו הבהיר ביותר גדול בהרבה מהתו הכהה ביותר. (אל תשכחו את זה כשתממשו את האלגוריתם!).

בשלב השלישי, נרצה לבצע **המרה של התמונה ל-ASCII**. עלינו לחלק את התמונה לתתי תמונות (ריבועים), לעבור באיטרציה על כל הריבועים ואז לקבוע עבור כל אחד מהם איזה תו ASCII יחליף אותו בציור שלנו.
<u>חישוב בהירות של תמונה</u> – תוכלו לראות ב-API של Image כי יש שיטה סטטית המקבלת מחרוזת של הכתובת ומחזירה אובייקט מסוג Image. בנוסף, ב-API של Image תוכלו למצוא את הפונקציה הבאה:

```
default Iterable<Color> pixels()
```

השיטה מחזירה iterable. זהו ממשק המבטיח כי מדובר באובייקט שניתן לבצע עליו איטרציות. ניתן לחשוב על אובייקט מסוג זה כמערך של Colors שניתן לבצע עליו איטרציות בצורה הבאה:

```
for (Color pixel: image.pixels()){
//       write you code here
        }
```

<u>כעת תורכם: ממשו שיטה העוברת על כל הפיקסלים של התמונה, מחשבת את הערך הממוצע של גוון הפיקסל ומחזירה ערך זה.</u>

הערות:

- כדי לחשב את הגוון של הפיקסל הצבעוני, המירו את הפיקסל לגוון אפור
- סכמו את כל גווני האפור של הפיקסלים וחלקו בסך הפיקסלים ובציון ה-RGB המקסימאלי (255). החזירו ערך זה.

<u>יצירת תמונת ascii</u> – כעת, נרצה לחלק את התמונה הגדולה למספר תתי תמונות. לכל תת-תמונה נחשב את הבהירות שלה ונחליף אותה בתו עם רמת הבהירות הקרובה ביותר. ה-API של החבילה image מבטיח לכם שלכל מופע מסוג Image יש שיטה בשם:

```
Iterable<Image> squareSubImagesOfSize(int pixel)
```

שיטה זו מקבלת מספר פיקסלים, מחלקת את התמונה לריבועים בגודל pixelsXpixels ומחזירה איטרטור על תתי-תמונות. כלומר, נוכל לרוץ באמצעות לולאה על כל תתי התמונות.
<u>עכשיו תורכם: ממשו שיטה עם החתימה הבאה:</u>

```
  private char[][] convertImageToAscii(for you to decide)
```

השיטה תחזיר את המערך הדו ממדי של תווי ה-ASCII אחרי ההמרה. היא תחלק את התמונה למספר תתי-תמונות הנדרש, תחשב את רמת הבהירות עבור כל תת-תמונה ותתאים לה את תו ה-ASCII עם רמת הבהירות הקרובה ביותר. את התו הזה היא תשמור במיקום המתאים במערך.

בדיקת שפיות: עבור התמונה bored.jpeg ותוים m ו-0 והערך numCharsInRow=2, אתם אמורים לקבל מערך בגדול 2*2 של:

m,o

o,m

עכשיו נותר רק לממש את השיטה:

```
public char[][] chooseChars(int numCharsInRow, Character[] charSet)
```

1. חשבו לכל תו את הבהירות שלו וסדרו את המערכים כך שהתו הכהה יהיה ראשון.
2. בצעו מתיחה לינארית לערכי הבהירות של התווים.
3. חשבו את מספר הריבועים שאותו אתם צריכים לחלק את התמונה. כמו כן, חשבו עבור כל ריבוע את התו המתאים והחליפו את הריבוע בתו. לבסוף, החזירו את הטבלה הדו מימדית של התווים.

בדיקת שפיות:

הריצו את הקוד הבא:

```
Image img = Image.fromFile("path\bored.jpeg");
BrightnessImgCharMatcher charMatcher = new BrightnessImgCharMatcher(img, "Ariel");
var chars = charMatcher.chooseChars(2, {'m','o'});
System.out.println(Arrays.deepToString(chars));
```

ובדקו שהודפס:

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

עד כאן חלק א'. בקבצי העזר הוספנו לכם גם את החבילה ascii_output אשר מייצאת את התמונה שלכם או לקונסול או לקובץ HTML. צרפנו לכם אותן כדי שתוכלו להשתעשע עם תמונות שונות ולייצא אותן בצורה נוחה. מוזמנים להמיר תמונות חמודות ולצרף לנו ל-README.

<u>לסיכום – מה אנחנו חייבים לראות בתרגיל שלכם:</u>

במחלקה Driver ממשו שיטת main.

במחלקה `CharRanderer`, ממשו את השיטה בעלת החתימה:

```
public static void printBoolArr(boolean[][] arr)
```

יצירת המחלקה `BrightnessImgCharMatcher` (ממשו לה constructor כראות עיניכם)
המחלקה חייבת להחזיק את השיטות בעלות החתימות הבאות:

```
public char[][] chooseChars(int numCharsInRow, Character[] charSet)
```

```
private char[][] convertImageToAscii(for you to decide)
```

שימו לב שחל איסור להשתמש ב Collections בעת כתיבת תרגיל זה. תוכלו להשתמש ב-Comparator אשר נלמד בעבר בכיתה.

<u>הוראות הגשה:</u>
עליכם להגיש קובץ jar בשם ex4_1.jar המכיל את המודול ascii_art וכן קובץ README המכיל הסברים על אופן מימוש ההתאמה בין רמת הבהירות לפיקסל.

אין צורך להגיש את קבצי העזר שלא שיניתם.

המון הצלחה!