

# תוכנה 1 – חורף 2019/20

## תרגיל מספר 3

### מערכים ומחרוזות

#### 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 ומספר התרגיל (לדוגמא, עבור המשתמש stav1 יקרא הקובץ stav1\_hw1.zip). קובץ ה-zip יכיל:
  - א. קובץ פרטים אישיים בשם details.txt המכיל את שמכם ומספר ת.ז.
  - ב. את תיקיית **src** ובתוכה את היררכיית התיקיות כפי שקיבלתם בקובץ הזיפ:  
src\il\ac\tau\cs\sw1\hw3  
כאשר בתיקיית hw3 יהיו שני הקבצים שקיבלתם: ArrayUtils.java ו-StringUtils.java. אין לצרף קבצים או תיקיות נוספות.
- הקפידו כי את תיקיית ה-src "רואים" מיד כשפותחים את הזיפ, ולא צריך קודם להיכנס לתיקיה אחרת עם שם הזיפ או הפרוייקט, למשל.
- כמו כן, נא לא להשתמש אף פעם במתודה System.exit. היא משבשת את הבדיקות האוטומטיות.

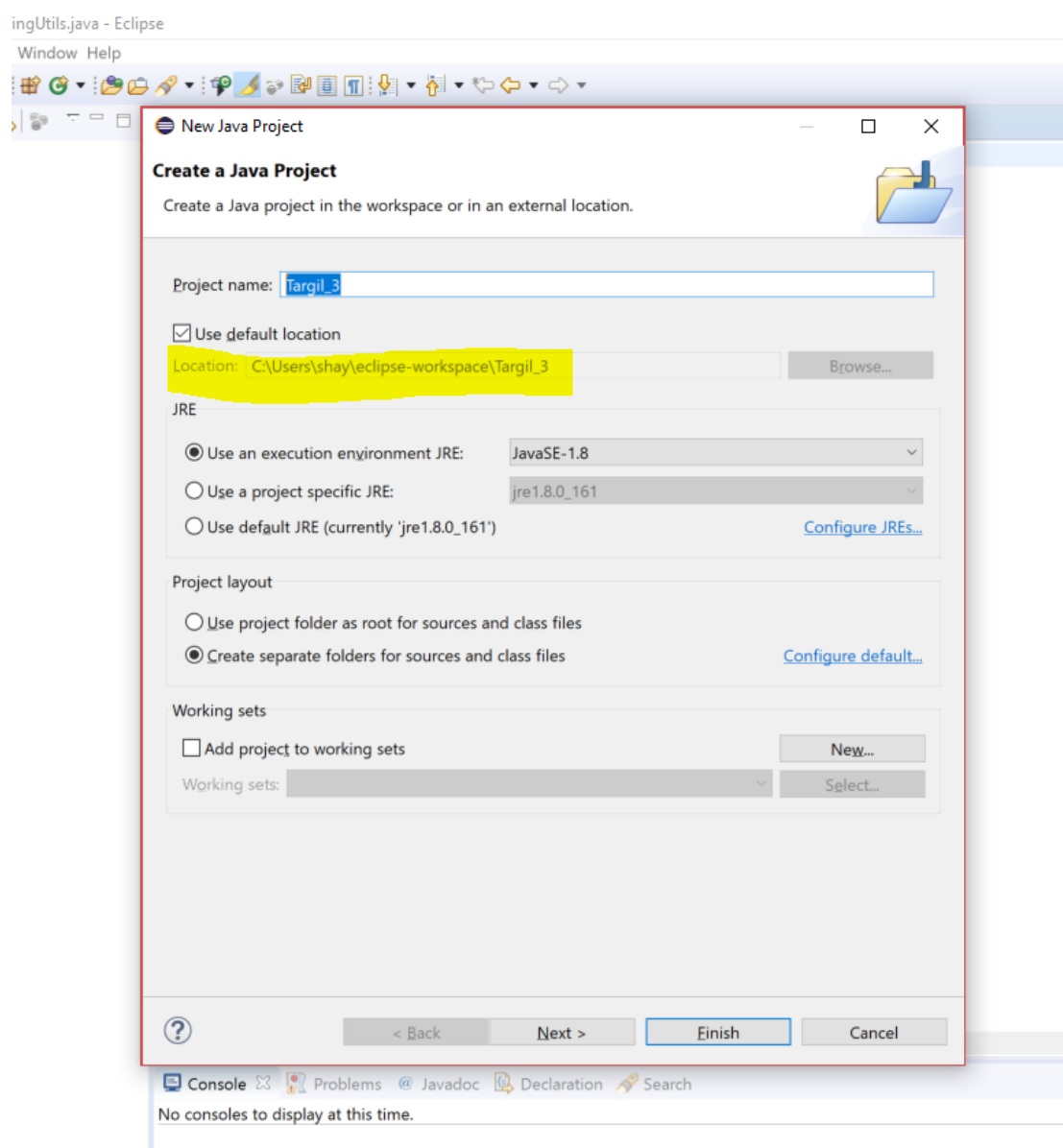
---

#### הערות כלליות:

- הקפידו שחתימות המתודות תהיינה **זרות** לאלו המצוינות בשאלה (אין לשנות את החתימות אשר ניתנו בשלד). כמובן שאת משפטי ה-return שמצויים בשלד אפשר לשנות.
- ניתן ומומלץ להוסיף מתודות עזר (מותר ומומלץ גם, אך לא חובה, שהן יהיו private).
- בתרגיל זה אין צורך לטפל במקרים בהם מערכים/מחרוזות הקלט שווים ל-null אלא אם צוין אחרת.
- **בתרגיל זה עליכם להגיש שתי מחלקות, ולהשלים את הקוד בשלד הנתון. המחלקות לא כוללות מתודת main, ואין להגיש אותן עם מתודת main.**
- כדי לבדוק את עצמכם, כתבו מחלקה נפרדת, בה הוסיפו מתודת main, ובדקו את המחלקות והמתודות בה. **את המחלקה אשר בניתם לצורך בדיקה אין להגיש.**
- כמו כן, לכל תרגיל מצורפות דוגמאות אשר מדגימות מהו הפלט הרצוי. עם זאת, בדיקת התרגיל תעשה על קלטים נוספים/אחרים.
- ניתן להוסיף פקודות import לתחילת הקבצים (אך מומלץ מאד שתמחקו לפי ההגשה פקודות import מיותרות – האקליפס מסמן אותן עם קו צהוב ואזהרה).

## הנחיות ליצירת פרוייקט חדש ב-Eclipse וייבוא השלד של תרגיל 3:

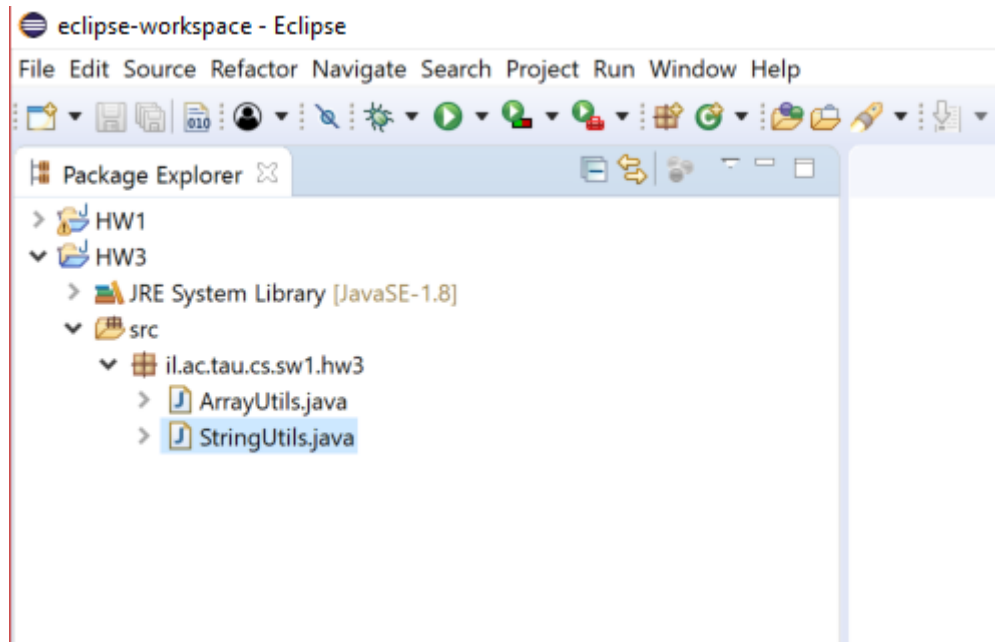
יש ליצור פרוייקט חדש ב-Eclipse ע"י בחירה ב- File-> New -> Java Project. רשמו את שם התוכנית (השם הוא לבחירתכם) בחלון שמופיע, וודאו שמסומנת הבחירה Use default location, ושימו לב שמתחתיה מופיע המיקום של ה-workspace בו ה-Eclipse שומר את התוכנית שלכם (מסומן בצהוב בצילום מסך המצורף). זה המיקום בו נמצאת התיקיה של התוכנית ובתוכה תיקיית ה-src שצריך להגיש (לאחר סיום כתיבת התוכנית, כמובן). כמו כן, וודאו באותו החלון כי הגירסה של ה-JRE היא 1.8 (כלומר ג'אוה 8). לחצו על Finish.



כעת, נווטו (מחוץ לאקליפס) אל התיקייה של הפרוייקט שיצרתם ב-workspace, והינסו בתוכה לתיקיית ה-src (שאמורה להיות ריקה). העתיקו את תוכן תיקיית ה-src (רק את התוכן, לא כולל התיקיה עצמה) שחילצתם מקובץ הזיפ אל תוך תיקיית ה-src המקומית ב-workspace.

לסיום, חזרו לאקליפס, לחצו על הפרוייקט שלכם לחיצה ימנית, ובחרו refresh. כעת שתי המחלקות שסופקו לכם אמורות להופיע בפרוייקט באקליפס.

מצורף צילום נוסף להמחשה (התעלמו מכך ששם הפרוייקט לא תואם לשם שנקבע בצילום מסך הקודם).



## חלק א' – מערכים (50 נק')

ממשו מחלקה בשם **ArrayUtils** שתכיל את המתודות הסטטיות הבאות:

### בחלק זה מבנה הנתונים היחידי בו מותר להשתמש הינו מערכים.

1. [10 נק'] ממשו מתודה בשם **transposeMatrix** המקבלת מערך דו-מימדי שמייצג מטריצה המכילה מספרים שלמים ומחזירה את המטריצה המשוחלפת שלה (כלומר, אתם מתבקשים לבצע **transpose**). האינדקס הראשון מציין את מספר השורה במטריצה (הערך של האינדקס עבור השורה הראשונה הוא אפס), והאינדקס השני את מספר העמודה. ניתן להניח כי המטריצה המתקבלת אינה null, אין בה אף שורה שהיא null, והינה מכילה רק מספרים שלמים. במידה ואחד המימדים של המטריצה הוא אפס, מחזירים את המטריצה ללא שינוי. שימו לב, כי ניתן (וצריך) לבצע transpose גם למטריצות שאינן ריבועיות. ניתן להניח שהמטריצה המתקבלת היא חוקית במובן שלכל שורה, כלומר תת-מערך, יש את אותו האורך. בתרגיל זה מותר להשתמש רק במשתנים מקומיים מסוג int. המטריצה שתחזירו יכולה להיות המערך המקורי שקיבלתם עם שינויים שביצעתם, או מערך חדש שיצרתם ומילאתם בהתאם.

```
public static int[][] transposeMatrix(int[][] m)
```

דוגמא:

```
transposeMatrix ([[1, 2, 3], [4,5,6], [7,8,9]]) -> [[1, 4, 7], [2,5,8], [3,6,9]]
```

```
transposeMatrix ([[ -1, 8], [7, -3]]) -> [[ -1, 7],[8, -3]]
```

```
transposeMatrix ([[5,3,2]]) -> [[5],[3],[2]]
```

```
transposeMatrix ([[1,2,3],[4,5,6]]) -> [[1,4],[2,5],[3,6]]
```

transposeMatrix ([]) -> [] // the input is an array (of arrays) of size zero, which is also the output.

למקרה שהסימון בדוגמה האחרונה מבלבל, נציין שהכוונה היא למערך באורך אפס (שהוא טכנית עדיין מערך של מערכים, אבל יש בו אפס כאלה).

## 2. [10 נק'] ממשו מתודה בשם *shiftArrayCyclic* המקבלת מערך המכיל מספרים שלמים

ומחזירה את המערך המקורי בו איברי מערך הקלט מוזזים לכיוון המצוין כמספר הפעמים שצוין. אם הפרמטר direction שווה ל-'R', המערך יוזז ימינה פעמים, כך שבכל פעם האיבר האחרון מוזז למקום הראשון במערך (ושאר האיברים זזים מקום אחד ימינה כמובן). ואילו כאשר הפרמטר direction שווה ל-'L', באופן אנלוגי לחלוטין, המערך יוזז שמאלה פעמים, כך שבכל פעם האיבר הראשון מוזז למקום האחרון במערך. ניתן להניח כי המערך המתקבל אינו null.

אם הכיוון המצוין אינו 'R' או 'L' או אם מספר ההזזות (move) אינו חיובי, המתודה תחזיר את המערך ללא שינוי. שימו לב כי מספר ההזזות, move, יכול להיות יותר מגודל המערך (במקרה כזה, שמופיע גם בדוגמאות בהמשך, אפשר לחשב את מספר ההזזות כמודולו אורך המערך, שכן כאשר מספר ההזזות שווה לאורך המערך, אנחנו חוזרים למצב המקורי). **יש להחזיר את המערך המקורי, לאחר שערכי התאים בו השתנו. מותר ליצור מערכי עזר.**

חתימת המתודה:

```
public static int[] shiftArrayCyclic(int[] array, int move, char direction)
```

דוגמא:

```
shiftArrayCyclic ([1, 2, 3, 4, 5],-1, 'R') -> [1,2,3,4,5]
```

```
shiftArrayCyclic ([1, 2, 3, 4, 5], 1, 'R') -> [5,1,2,3,4]
```

```
shiftArrayCyclic ([1, 2, 3, 4, 5], 1, 'r') -> [1,2,3,4,5]
```

```
shiftArrayCyclic ([1, 2, 3, 4, 5], 1, 'g') -> [1,2,3,4,5]
```

```
shiftArrayCyclic ([1, 2, 3, 4, 5], 3, 'L') -> [4,5,1,2,3]
```

```
shiftArrayCyclic ([0, 8, 9, 5, 6], 6, 'L') -> [8, 9, 5, 6,0]
```

shiftArrayCyclic ([],3, 'R') -> []

3. **[15 נק']** נגדיר **סכום מתחלף** באופן הבא: בהינתן שני אינדקסים:  $i, j$ , הסכום המתחלף הינו האיבר במקום ה- $i$ , פחות האיבר במקום ה- $i+1$ , ועוד האיבר במקום ה- $i+2$ , וכו' (עד האינדקס ה- $j$ ). כלומר סכום כל האיברים בין שני האינדקסים הנתונים, כאשר כל איבר שני מחוסר מהסכום. אורך סדרת הסכום הזאת יכול להיות גם אי-זוגי כולל באורך 1. וסדרת סכום יכולה להיות גם באורך אפס כאשר הסכום עצמו מוגדר להיות אפס גם כן.

ממשו מתודה בשם **alternateSum המקבלת מערך המכיל מספרים שלמים, ומחשבת את הסכום המתחלף המקסימלי**. כלומר המתודה בודקת את כל הסכומים המתחלפים האפשריים, ומחזירה את המקסימלי מביניהם. שימו לב שהסכום הזה הוא תמיד אי-שלילי, כי תמיד יש סדרה של אפס איברים (שסכומה מוגדר להיות אפס). ניתן להניח כי המערך המתקבל אינו null. שימו לב כי הסכום המתחלף מוגדר להיות אך ורק בין איברים עוקבים. ניתן להניח כי אורך המערך הינו לפחות 2 או מערך ריק, כלומר אין צורך להתמודד עם מערך באורך 1, אם כי מספר האיברים בסכום המתחלף עצמו יכול להיות 1. כמו כן, **לא** ניתן "לסובב" את המערך, כלומר להתחיל את הסכום בסוף המערך ואז להמשיך לתחילתו. מותר, אך לא חובה, להשתמש במערכי עזר.

חתימת המתודה:

```
public static int alternateSum(int[] array)
```

דוגמא:

```
alternateSum([1, -2, 3, 4, 5]) -> 7 (1-(-2)+3-4+5)
```

```
alternateSum([1, 2, -3, 4, 5]) -> 9 (2-(-3)+4)
```

```
alternateSum([]) -> 0 //if the array is empty, then the return value is 0
```

הערה: מותר להוסיף מתודות עזר במידת הצורך. נסו לחשוב על פיתרון יעיל כמה שניתן (ביעילות הכוונה היא בעיקר לסיבוכיות אסימפטוטית כתלות ב  $n$ , ופחות לדברים כמו לצמצם מ-3 פעולות פר איטרציה ל-2 פעולות פר איטרציה על חשבון קוד מסורבל. וגם אין הכרח שהפתרון יהיה אופטימלי מבחינת יעילות כדי לקבל את כל הנקודות, אך בכל זאת מומלץ להשקיע מעט זמן לוודא שהפתרון לא סובל מבזבזים בולטים מבחינת מספר הפעולות הכולל. הכי חשוב הוא שלפתרון לא תהיה סיבוכיות אקספוננציאלית).

**רמז:** האם ניתן להשתמש בחישובים שכבר חישבנו?

4. **[15 נק']** ממשו מתודה בשם **findPath** המקבלת מערך דו מימדי אשר מייצג מטריצת שכנויות של גרף מכוון. כלומר הכניסה ה- $[i][j]$  מכילה את הספרה 1 אם קיימת קשת מכוונת מהקודקוד ה- $i$  לקודקוד ה- $j$ , ו-0 אחרת. זה אומר, במילים אחרות, שיש מסלול באורך 1 מקודקוד  $i$  לקודקוד  $j$ . שימו לב, כיוון שהגרף מכוון, לא בהכרח מתקיים שהכניסה ה- $[i][j]$  שווה לכניסה ה- $[j][i]$ . אלכסון המטריצה מכיל אחדות, כי אנו מניחים שלכל קודקוד יש קשת עצמית (מעצמו לעצמו). שימו לב, כי המטריצה הזאת היא ריבועית (כלומר מספר התת-מערכים שווה לאורכו של כל מערך, והאורך הזה הוא מספר הצמתים בגרף).

המתודה מקבלת בנוסף שני אינדקסים  $i, j$  ומחזירה 1 אם קיים בגרף מסלול באורך כלשהו שמתחיל ב- $i$  (הפרמטר השני שמקבלת המתודה) ומסתיים ב- $j$  (הפרמטר השלישי שמקבלת המתודה), ו-0 אחרת. ניתן להניח כי המטריצה הינה מטריצה ריבועית אשר עומדת בדרישות, האינדקסים מהווים קלט חוקי וכי המטריצה שונה מ-null.

**הערה: מותר להוסיף מתודות עזר במידת הצורך. נסו לחשוב על פיתרון יעיל כמה שניתן (ראה הערה על סיבוכיות בסוף ההנחיות לשאלה הקודמת).** מותר, כמובן, אך לא חובה, להשתמש ברקורסיה.

אמנם אין על כך איסור רשמי, אך מומלץ לא לשנות את המטריצה המקורית במהלך הפתרון, כי זה יכול בקלות להוביל לשגיאות בחישוב.

חתימת המתודה:

```
public static int findPath(int[][] m, int i, int j)
```

דוגמא:

```
findPath ([[1,0,0],[0,1,0],[0,0,1]],1,1) -> 1
```

```
findPath ([[1,0,0,1],[0,1,0,1],[0,0,1,0],[1,1,0,1]],0,1) -> 1
```

```
findpath([[1,1,0],[0,1,1], [0,1,1]],0,2) -> 1
```

```
findpath([[1,1,0],[0,1,1], [0,1,1]],2,0) -> 0
```

```
findPath ([[1,0,0,1],[1,1,0,1],[0, 1,1,0],[1,1,0,1]],0,2) -> 0
```

## חלק ב' – מחרוזות (50 נק')

**בחלק זה המבנה היחיד בו מותר להשתמש הינו מערכים (למען הסר ספק, מחרוזות לא נחשבות בתור מבני נתונים וניתן להשתמש בהן).**

**מותר, אך לא חובה, להשתמש במתודות המובנות במחלקות String ו-Arrays, אם לא נאמר אחרת.**

ממשו מחלקה בשם **StringUtils** שתכיל את המתודות הסטטיות הבאות:

5. **[10 נק']** ממשו מתודה בשם **findSortedSequence** המקבלת מחרוזת קלט הכוללת אותיות אנגליות קטנות ורווחים בלבד, כאשר בין כל שתי מילים יש בדיוק רווח יחיד (ניתן להניח את תקינות הקלט), ומחזירה את תת-המחרוזת הארוכה ביותר שלה (מבחינת מספר המילים בה) בה המילים ממוינות לפי סדר לקסיקוגרפי עולה (עם רווחים בין המילים). אם יש כמה תת-מחרוזות ממוינות בעלות אותו אורך (שוב – מבחינת מספר המילים), יש להחזיר את זו שקרובה יותר לסוף מחרוזת הקלט. שימו לב שמיון לקסיקוגרפי זה אותו סדר בו המילים מופיעות במילון, ולא מדובר ב"מיון פנימי" כך שהאותיות במילה עצמה ממוינות. בין היתר, נובע שאם מילה A היא רישא של מילה B, אז במיון לקסיקוגרפי A תופיע בהכרח לפני B. שתי מילים (או יותר) זהות שמופיעות ברצף נחשבות ממוינות לקסיקוגרפית. במקרה בו אין במחרוזת שתי מילים רצופות שממוינות בסדר לקסיקוגרפי עולה (כלומר המחרוזת כולה ממוינת בסדר יורד) יש להחזיר רק את המילה האחרונה (זה לא כלל נוסף, אלא הבהרה למקרה הקצה שנובעת מההנחיה הכללית). אם מחרוזת הקלט היא המחרוזת הריקה, יש להחזיר מחרוזת

ריקה גם כן (אך לא null!). מחרוזת שיש בה רק רווחים שקולה למחרוזת ריקה (יש להחזיר מחרוזת ריקה).  
מותר, אך כלל לא חובה, להשתמש במתודות מיון "מוכנות" מהספריות הסטנדרטיות של ג'אווה (כמו ב- java.util.Arrays). מותר, אך לא חובה, גם להשתמש במתודת join של String וגם במתודות השוואה של מחרוזות שנלמדו בתרגול.  
תזכורת: שוויון בין מחרוזות יש לבדוק עם המתודה equals ואף פעם לא עם ==.  
ניתן להניח כי מחרוזת הקלט שונה מ-null. וודאו שאין במחרוזת המוחזרת רווחים לפני המילה הראשונה או אחרי האחרונה, ושיש רווח אחד בדיוק בין כל שתי מילים עוקבות.

✓ רמז: היעזרו בפקודה split של המחלקה String

חתימת המתודה:

```
public static String findSortedSequence (String str)
```

דוגמא (שימו לב שלא צריך גרשיים בפלט, והם מופיעים כאן רק כדי לסמן שמדובר במחרוזת):

```
findSortedSequence ("to be or not to be") -> "not to"
```

```
findSortedSequence ("my mind is an empty zoo") -> "an empty zoo"
```

```
findSortedSequence ("") -> ""
```

```
findSortedSequence ("andy bought candy") -> "andy bought candy"
```

```
findSortedSequence ("life is not not not fair") -> "is not not not"
```

```
findSortedSequence ("art act") -> "act"
```

6. **20 נק'** כתבו מתודה בשם: *parityXorStrings* המקבלת שתי מחרוזות המורכבות מאותיות אנגליות קטנות בלבד. המתודה תחזיר מחרוזת חדשה, המכילה רק את האותיות המופיעות מספר אי-זוגי של פעמים במחרוזת הראשונה וזוגי בשניה. 0 מופעים נחשב מספר זוגי של מופעים.

אין צורך לבדוק את תקינות הקלט. במידה ואין תווים אשר עומדים בדרישה, יש להחזיר את המחרוזת הריקה.

סידור התווים במחרוזת המוחזרת יהיה לפי סדר הופעתם במחרוזת הראשונה (כולל חזרות, כלומר אם תו מסוים הופיע 3 פעמים במחרוזת הראשונה, הוא יופיע גם 3 פעמים במחרוזת המוחזרת, ולא בהכרח ברצף). ניתן להניח כי שתי המחרוזות שונות מ-null.

חתימת המתודה:

```
public static String parityXorStrings (String a, String b)
```

דוגמאות:

```
parityXorStrings ("dog","god") -> ""
```

```

parityXorStrings ("catcatcat","tacotaco") -> "catcatcat"

parityXorStrings ("cat","jeffjeff") -> "cat"

parityXorStrings ("catcat","jeffjeff") -> ""
parityXorStrings ("catcat","jeff") -> ""

parityXorStrings ("cat","cajeffjefft") -> ""

parityXorStrings ("jeff","catff") -> "je"

parityXorStrings ("izoi","oziizo") -> "zo"

parityXorStrings ("fireman","maniac") -> "firea"

```

7. [20 נק'] אנגרמה היא שעשוע לשון שבו יוצרים מילה חדשה מערבוב אותיותיה של מילה קיימת, או משפט חדש מערבוב אותיות של משפט קיים. לקריאה נוספת: [אנגרמה](#).

כתבו מתודה בשם *isAnagram* אשר מקבלת שתי מחרוזות, ומחזירה האם אחת המחרוזות מתקבלת על ידי שיכול אותיות של השנייה, כלומר האם המחרוזת השנייה הינה אנגרמה של המחרוזת הראשונה. תווים יכולים לחזור על עצמם ובאנגרמה כל תו צריך להופיע אותו מספר פעמים (לא כולל רווחים שיכולים להשתנות במספרם). בתרגיל זה אנו נבדיל בין אות קטנה לגדולה, כלומר האות a אינה נחשבת זהה לאות A.

ייתכן שיהיו כמה רווחים ברציפות, אך למספר הרווחים אין שום השפעה לגביי שאלה ההכרעה – האם מדובר באנגרמה. נובע מההגדרה שאם (ורק אם) שתי מחרוזות הן אנגרמה, אז אם מסדרים כל אחת מהן בשורה כך שהאותיות שלה (כולל חזרות) ממוינות בסדר לקסיקוגרפי (ומוחקים את כל הרווחים), ונניח למשל שאותיות גדולות מופיעות לפני קטנות בסדר לקסיקוגרפי, בשני המקרים נקבל בדיוק את אותה מחרוזת. ניתן להניח כי כל התווים בשתי המחרוזות הן אותיות אנגלית קטנות או גדולות, או רווחים. מותר, אך לא חובה, להשתמש במתודה Arrays.sort.

חתימת המתודה:

```
public static boolean isAnagram(String a, String b)
```

דוגמאות:

```

isAnagram("mothEr in law","hitlEr woman") -> true

isAnagram("ListeN","SileNt") -> false

isAnagram("software","jeans") -> false

isAnagram("Funeral","real Fun") -> true

isAnagram("Aa","aA") -> true

isAnagram("", " ") -> true

```

---

**הערה:** בכל השאלות אין לערבב בין הקבצים – כלומר אין לקרוא מקובץ אחד למתודה (גם לא מתודת עזר) מקובץ אחר. אם אתם רוצים להשתמש באותה מתודת עזר, יש לרשום אותה פעם אחת בכל



קובץ (זה נועד עבור פשטות הבדיקה, ובפרויקט אמיתי היינו נמנעים עקרונית משכפול קוד). בתוך אותה מחלקה, ניתן בהחלט להשתמש באותה מתודת עזר לתרגילים שונים.

**בהצלחה !**