פרק 9 – המחלקה מחרוזת (String)

בתוכניות שכתבנו עד כה השתמשנו בטיפוסים שונים המוגדרים בשפת Java: שלם, ממשי, תווי ובוליאני. יכולנו להגדיר משתנים מטיפוסים אלו ולבצע עליהם פעולות שונות (קלט, פלט, חישובים וכו'). עם זאת מרבית התוכניות שכתבנו התייחסו גם ל**מחרוזות**, סדרות של תווים. עד עתה השתמשנו במחרוזות כאשר רצינו להדפיס הודעות למשתמש ותחמנו אותן בגרשיים, למשל בהוראה:

```
System.out.println("Enter two numbers:");
```

ההודעה "Enter two numbers:" היא מחרוזת.

הנה דוגמאות נוספות למחרוזות בשפת Java: "Hello, How Are You?", "453", "J", "" (האחרונה היא מחרוזת ריקה, שאינה מכילה אף תו).

בנוסף להדפסה נרצה לעתים להפעיל על מחרוזות פעולות נוספות. בפרק זה נראה כיצד ניתן לקלוט מחרוזות מהמשתמש, ולעבד אותן. אם עד עתה קלטנו מספרים או תווים בודדים בלבד, עכשיו נוכל לבקש מהמשתמש להקיש את שמו, את כתובתו וכדומה ולבצע פעולות שונות על המחרוזות הנקלטות.

9.1 היכרות ראשונית עם המחלקה String

מחלקה (**class**) מגדירה טיפוס חדש, ועצם (**object**) הוא משתנה מהטיפוס. עצם אינו משתנה פשוט כמו int או כמו char אלא **מופע** (**instance**) של מחלקה שאפשר להפעיל עליו פעולות המוגדרות במחלקה.

בשפת Java מוגדרת **מחלקה** בשם `String` (מחרוזת). הצהרה על משתנה מטיפוס מחרוזת מתבצעת כך:

```
String s1;
```

אמנם הצהרה על עצמים דומה להצהרה על משתנים מטיפוסים רגילים. אבל בשונה ממשתנים מטיפוסים רגילים, עלינו ליצור עצמים. יצירת עצם כוללת הקצאת שטח בזיכרון עבור העצם ואת אתחולו. עבור משתנים רגילים, הקצאת מקום בזיכרון נעשית אוטומטית עם ההצהרה עליהם ואילו עבור עצמים הקצאת הזיכרון והאתחול מתבצעים באמצעות הפעולה **new**, למשל כך:

```
s1 = new String();
```

הפעולה **new** מקצה זיכרון עבור עצם בשם `s1` המפנה לזיכרון שהוקצה. במקרה זה העצם `s1` מאותחל במחרוזת ריקה "". אם נרצה לאתחל את העצם בערך כלשהו, נוכל לעשות זאת בציון הערך הרצוי בזמן היצירה, למשל כך:

```
s1 = new String("abc");
```

כעת העצם `s1` מפנה לשטח בזיכרון המכיל את המחרוזת "abc".

מכיוון שהשימוש במחרוזות נפוץ כל כך, שפת Java מאפשרת ליצור מחרוזת ולאתחל אותה בצורה ישירה ללא שימוש בפעולה **new**. למשל, כך:

```
s1 = "xyz";
```

יצירת מחרוזת בצורה זו מקצה זיכרון עבור המחרוזת `s1`, ומאתחל אותה ב-"xyz".

בדומה למשתנים פשוטים, אפשר לאחד את ההצהרה ואת היצירה כך:

```
String s1 = new String("hello");
```

או כך:

```
String s2 = "good morning";
```

בהמשך הפרק נכיר פעולות שונות שאפשר לבצע על עצמים מטיפוס String.

בשפת Java **מחרוזת** היא עצם של המחלקה String.

הצהרה על מחרוזות מתבצעת באופן דומה להצהרה על משתנים רגילים, למשל:

```
String s1;
```

כדי שניתן יהיה להשתמש בעצם מהמחלקה מחרוזת, יש ליצור את העצם. **יצירת העצם** כוללת **הקצאת שטח בזיכרון ואתחול העצם**.

במקרה של עצם מסוג String, אפשר ליצור אותו באמצעות הפעולה **new**:

```
String s1 = new String("hello");
```

או באמצעות השמה פשוטה של סדרת תווים בגרשיים:

```
String s2 = "good morning";
```

מצ'ה 1

מטרת הבעיה ופתרונה: הצגת קלט של מחרוזת.

פתחו וישמו אלגוריתם שיקבל כקלט את שמכם ויציג :Your name is ומיד אחר כך את השם שנקלט.

בחירת משתנים

? באיזה טיפוס נשתמש כדי לשמור את השם שנקלט?

כיוון שעלינו לקלוט מילה נשתמש במחלקה מחרוזת. אם כך רשימת המשתנים תכלול את העצם name מהמחלקה מחרוזת.

האלגוריתם:

1. *קלוט מחרוזת ב-*name
2. *הדפס כפלט* "Your name is: "
3. *הדפס כפלט את* name

יישום האלגוריתם

עד כה ראינו כיצד לקלוט ערכים מטיפוסים פשוטים – שלמים, ממשיים ותווים. קליטת מחרוזת נעשית בצורה דומה. את הוראה 1 נוכל לממש במשפטים:

```
Scanner in = new Scanner(System.in);
name = in.nextLine();
```

הפעולה `nextLine` קולטת את כל התווים עד סוף השורה ומחזירה עצם מסוג מחרוזת. למשל אם המשתמש הקליד:

```
my name is Moshe
```

העצם name יכיל את המחרוזת: "my name is Moshe"

התוכנית המלאה

```
/*
קלט: שם, הנקלט כמחרוזת
פלט: השם, מלווה בהודעה מקדימה
*/
import java.util.Scanner;
public class YourName
{
    public static void main (String [] args)
    {
        String name;    // הצהרה על עצם מהמחלקה מחרוזת
        Scanner in = new Scanner(System.in);
        // הוראת קלט + הקצאת מקום בזיכרון
        System.out.print("Enter your name: ");
        name = in.nextLine();
        // פלט
        System.out.println("your name is: " + name);
    } // main
} // class YourName
```

שימו ♥: כמו פעולות קלט לערכים מטיפוסים פשוטים, גם פעולת קלט למחרוזת היא למעשה פעולה המחזירה את הערך הנקלט. הפעולה קולטת מחרוזת מהקלט, ויוצרת עצם חדש מהמחלקה מחרוזת. פעולת הקלט מקצה עבורו מקום בזיכרון ושומרת בו את מחרוזת הקלט. הפעולה in.nextLine מחזירה הפניה למחרוזת החדשה שהוקצתה.

בעת ביצוע ההוראה:

```
name = in.nextLine();
```

אנו מבצעים השמה של המחרוזת הנקלטת במשתנה name. בעקבות ההשמה מַפְנֶה name אל אותו שטח זיכרון שהוקצה על ידי פעולת הקלט ולכן אין צורך להקצות במפורש שטח זיכרון עבור העצם name.

סוף פתרון בעיה 1

בפתרון בעיה 1 הראינו כיצד קולטים מחרוזת באמצעות הפעולה nextLine של המחלקה Scanner. פעולה זו קולטת מהמשתמש מחרוזת המסתיימת בסוף השורה.

קיימת פעולה נוספת בשם next הקולטת מחרוזת-חלקית שסופה מצוין ברווח הראשון שהמשתמש מקליד. למשל כתוצאה מביצוע ההוראה הבאה:

```
name = in.next();
```

אם המשתמש מקליד:

```
Moshe Cohen
```

העצם name יכיל את המחרוזת: "Moshe". במקרה שהקלט לא כולל תו רווח, המחרוזת תסתיים בסוף השורה.

ניתן **לקלוט מחרוזת** באמצעות הפעולות in.nextLine או in.next. פעולות הקלט מקצות שטח בזיכרון עבור המחרוזת החדשה.

9.2 ביצוע פעולות על מחרוזות

כדי לבצע פעולות על עצמים יש לרשום את שם העצם, אחריו את סימן הנקודה ולאחר מכן את שם הפעולה. למשל, כדי להגריל מספר הפעלנו את הפעולה nextInt על עצם מסוג Random כך:

```
Random rnd = new Random();
int num;
num = rnd.nextInt(100);
```

במקרה זה הפעולה nextInt מחזירה מספר אקראי בין 0 ל-99 הנשמר במשתנה num. באופן דומה, אפשר להפעיל פעולות שונות על עצמים מסוג מחרוזת. אחת הפעולות המוגדרות עבור מחרוזות היא הפעולה `length`, המחזירה את אורך המחרוזת שעליה היא מופעלת. למשל, כדי לשמור את אורך המחרוזת `s1` במשתנה len נוכל לכתוב:

```
int len;
len = s1.length();
```

הסוגריים שאחרי שם הפעולה נועדו להעברת ערכים (פרמטרים) שהפעולה נזקקת להם . במקרה של הפעולה `length` אין צורך בהעברת פרמטרים ולכן הסוגריים נותרים ריקים. בהמשך הפרק נכיר פעולות נוספות אשר מצפות לקבל פרמטרים לצורך הפעלתן.

תרגיל 2

מטרת הבעיה ופתרונה: הדגמת השימוש בפעולות של מחרוזות. הכרת הפעולות `length` ו-`charAt`.

פתחו וישמו אלגוריתם אשר יקבל כקלט מחרוזת. אם אורך המחרוזת הוא זוגי יוצג כפלט התו הראשון במחרוזת ואם אורך המחרוזת הוא אי-זוגי יוצג התו האחרון במחרוזת.
למשל, עבור הקלט shalom יוצג התו: 's'. ועבור הקלט dog יוצג התו: 'g'.

שימו ♥: בשפת Java מיקום התווים במחרוזת מתחיל ב-0. כלומר, התו הראשון במחרוזת נמצא במקום 0, התו השני במחרוזת הוא במקום 1, השלישי במקום 2 וכן הלאה.

? אם התו הראשון נמצא במקום 0, באיזה מקום נמצא התו האחרון?

נתבונן למשל במחרוזת "abcd" שאורכה 4. התו האחרון בה הוא 'd' והוא נמצא במקום 3.

מיקום התווים במחרוזת מתחיל ב-0.
בכל מחרוזת, אם נסמן את אורך המחרוזת ב-len אז התו האחרון במחרוזת נמצא במקום len-1.

בחירת משתנים

נשתמש במשתנים הבאים:

str – עצם מהמחלקה מחרוזת
letter – האות המבוקשת, מטיפוס char
len – לשמירת אורך המחרוזת, מטיפוס שלם

האלגוריתם

1. *קלוט מחרוזת* str-*ב*
2. *השם את אורכה של המחרוזת* str *ב*-len
3. *אם* len *זוגי*
   3.1 *השם ב*-letter *את התו הנמצא במקום 0 ב*-str
4. *אחרת*
   4.1 *השם ב*-letter *את התו הנמצא במקום 1*-len *ב*-str
5. *הדפס כפלט את ערכו של* letter

יישום האלגוריתם

? כדי לבדוק את אורכה של המחרוזת נשתמש בפעולה `length`, באיזו פעולה נשתמש כדי לדעת מהו התו שנמצא במקום מסוים במחרוזת?

במחלקה מחרוזת מוגדרות פעולות נוספות ואחת מהן היא הפעולה `.charAt(k)` פעולה זו מקבלת כפרמטר (בתוך הסוגריים) מספר שלם `k`, ומחזירה את התו שנמצא במקום ה-`k` במחרוזת שעליה הופעלה הפעולה. לכן למשל כדי לדעת מהו התו שנמצא במקום 0 במחרוזת `str` נכתוב:

```
letter = str.charAt(0)
```

שימו ♥: השתמשנו פעמים רבות בפעולה זו כאשר קלטנו תו יחיד. למעשה קלטנו מחרוזת וחילצנו ממנה את התו הראשון (התו שנמצא במקום האפס).

התוכנית המלאה

```
/*
קלט: מחרוזת
פלט: התו הראשון או האחרון במחרוזת, בהתאם לאורכה (זוגי או אי-זוגי)
*/
import java.util.Scanner;
public class IsEven
{
    public static void main (String[] args)
    {
        String str;
        int len;
        char letter;
        Scanner in = new Scanner(System.in);
        System.out.print("Enter a string: ");
        str = in.nextLine();
        len = str.length();
        if (len % 2 == 0)
            letter = str.charAt(0);
        else
            letter = str.charAt(len - 1);
        System.out.println("The letter is " + letter);
    } // main
} // class IsEven
```

סוף התרגול בשיעה 2

בסוף הפרק תוכלו למצוא טבלה המפרטת את הפעולות השכיחות המוגדרות ב-Java עבור עצמים מהמחלקה String.

העמודה הימנית בטבלת הפעולות מתארת את שם הפעולה ואת רשימת הפרמטרים שהיא מצפה לקבל (בתוך סוגריים). כפי שראינו זה עתה, הפעולה הראשונה בטבלה `length` אינה מצפה לקבל ערך כלשהו, ולכן הסוגריים ריקים. לעומתה, הפעולה השנייה `charAt` מצפה לקבל פרמטר אחד מטיפוס שלם. הפעולה השלישית `indexOf` מצפה לקבל עצם מהמחלקה מחרוזת או לקבל תו.

העמודה השלישית מתארת את טיפוס הערך המוחזר כתוצאה מהפעלת הפעולה. בדומה לפעולות המתמטיות שלמדנו בפרק 4, גם כאן יש לשים לב לטיפוס הערך המוחזר ולוודא התאמה בינו ובין הפעולות שנבצע עליו. כלומר כאשר נשים בתוך משתנה ערך שמוחזר מפעולה, עלינו לוודא התאמה בין טיפוס הערך המוחזר לטיפוס המשתנה שההשמה תתבצע בו.

מחלקות מגדירות אוסף של **פעולות** שאפשר להפעיל על עצמים מהמחלקה.

הפעלת פעולה נכתבת כך (משמאל לימין):

```
שם העצם.שם הפעולה(פרמטרים)
```

יש לבדוק היטב בתיאור הפעולה אילו ערכים היא מצפה לקבל ומה טיפוס הערך שהיא מחזירה.

שאלה 9.1

פתחו וישמו אלגוריתם שמקבל כקלט רשימה של 100 מחרוזות. האלגוריתם מחשב בכמה מחרוזות מתוך ה-100 התו הראשון שווה לתו האחרון, ומציג את הערך שחישב.

שאלה 9.2

פתחו אלגוריתם שיקבל כקלט מחרוזת ותו. פלט האלגוריתם יהיה מספר ההופעות של התו הנתון בתוך המחרוזת הנתונה. ישמו את האלגוריתם בשפת Java.

הדרכה: מאחר שניתן לברר את אורך המחרוזת הנתונה ניתן להשתמש בהוראה לביצוע-חוזר שמספר הסיבובים בה ידוע מראש (לולאת **for**).

מעי'ה 3

מטרת הבעיה ופתרונה: הדגמת השימוש בפעולה להשוואת מחרוזת במחלקה String, ושילוב מחרוזות בתבנית מנייה.

בסיום בחינות הקבלה למקהלת הזמר העירונית יוצא הבוחן הראשי וקורא את רשימת המועמדים שהתקבלו על פי סדר הא״ב. מועמד אשר שומע את שמו יודע כי התקבל ללהקה. מועמד ששמו עדיין לא נקרא, אשר שומע שם אחר שמופיע אחריו בסדר הא״ב, יודע כי לא התקבל ללהקה.

פתחו אלגוריתם אשר הקלט שלו הוא שם מועמד, ולאחר מכן רשימת שמות שקורא הבוחן על פי סדר הא״ב. פלט האלגוריתם הוא אם המועמד התקבל, ומספר השמות שהיה עליו לשמוע לפני שהגיע למסקנה אם התקבל או לא. ישמו את האלגוריתם בשפת Java.

פירוק הבעיה לתת-משימות

למעשה האלגוריתם צריך להשוות מחרוזות ובנוסף לבצע מנייה. אלגוריתם זה דומה לאלגוריתמים שפיתחנו בפרקים קודמים. אם כך, נפרק את הבעיה לתת-משימות הבאות:

1. קליטת שם המועמד
2. קליטת שמות המועמדים שהתקבלו והשוואתם לשם המועמד
3. מניית מספר המחרוזות שנקלטו
4. הצגה אם המועמד התקבל והצגת מספר המחרוזות שהיה עליו לשמוע

תת-משימה 3 תבוצע בדומה למשימות מנייה אחרות, ההבדל הוא כמובן **בטיפוס** הערכים הנמנים – מחרוזת במקרה זה – ובאופן ביצוע הפעולות עליהם. בשלב יישום האלגוריתם נראה כיצד לעשות זאת.

בחירת משתנים

name – שם המועמד, עצם מהמחלקה מחרוזת
str – השם התורן מהקלט, עצם מהמחלקה מחרוזת
counter – למניית מספר המחרוזות, מטיפוס שלם

האלגוריתם

1. *אתחל את* counter *ל-*1
2. *קלוט מהמקלדת ל-*name
3. *קלוט מהמקלדת ל-*str
4. *כל עוד* name > str *(בסדר מילוני) בצע:*
   4.1. *הגדל את ערכו של* counter *ב-*1
   4.2. *קלוט מהמקלדת ל-*str
5. *אם* name = str
   5.1. *הצג כפלט "המועמד* <name> *התקבל"*
6. *אחרת*
   6.1. *הצג כפלט "המועמד* <name> *לא התקבל"*
7. *הצג את ערכו של* counter

יישום האלגוריתם

❓ כיצד נבדוק אם שם המועמד גדול על פי **סדר מילוני** מהמחרוזת הנקלטת?

אחת הפעולות המוגדרות עבור מחרוזות היא הפעולה `CompareTo` המשווה מחרוזת למחרוזת נוספת. אם `s1` הוא עצם מהמחלקה מחרוזת, ו-`s2` הוא עצם נוסף מהמחלקה מחרוזת, אז הביטוי `s1.CompareTo(s2)` מפעיל את הפעולה `CompareTo` המשווה את `s1` למחרוזת נוספת (`s2`) שהתקבלה כפרמטר.

הפעולה `CompareTo` מחזירה את תוצאת ההשוואה כערך שלם באופן הבא:

- אם המחרוזות שוות יוחזר הערך 0.
- אם המחרוזת `s1` שמופעלת עליה הפעולה, **קודמת** למחרוזת `s2` שהתקבלה כפרמטר בסדר מילוני, יוחזר מספר שלם **שלילי**.

- אם המחרוזת `s1` שמופעלת עליה הפעולה, מופיעה **אחרי** המחרוזת `s2` שהתקבלה כפרמטר בסדר מילוני, יוחזר מספר שלם **חיובי**.

לכן, כדי לבדוק אם שם המועמד `name` גדול מהמחרוזת הנקלטת `str`, נוכל להשתמש בפעולה `name.compareTo(str)`. אם הפעולה תחזיר ערך חיובי נדע ששם המועמד גדול מהמחרוזת הנקלטת , ואם היא תחזיר ערך שלילי נדע ששם המועמד קטן מהמחרוזת הנקלטת. אם הפעולה תחזיר 0, נדע שהמחרוזות שוות.

❓ כיצד נבדוק אם המחרוזת האחרונה שנקלטה שווה לשם המועמד?

אפשר להשתמש שוב בפעולה `compareTo` ולבדוק אם היא מחזירה את הערך 0. לחילופין אפשר לבדוק שוויון של שתי מחרוזות באמצעות הפעולה `equals` הבודקת שוויון של המחרוזת שעליה היא מופעלת למחרוזת המתקבלת כפרמטר. למשל, הביטוי `name.equals(str)` יחזיר **`true`** אם המחרוזת `str` שווה למחרוזת `name`, אחרת יוחזר **`false`**.

שימו ♥: את המונה נאתחל ב-1 כיוון שמחוץ ללולאה אנו קולטים את השם הראשון שנקרא.

התוכנית המלאה

```
/*
קלט: שם מועמד ורשימה ממוינת של שמות
פלט: האם המועמד התקבל ומספר השמות שהיה עליו לשמוע
*/
import java.util.Scanner;
public class NameFinder
{
    public static void main (String[] args)
    {
        // הגדרת משתנים ואתחולם
        String name;
        String str;
        int counter = 1;
        // קלט שם מבוקש
        Scanner in = new Scanner(System.in);
        System.out.print("Enter the candidate name: ");
        name = in.nextLine();
        // לולאת זקיף
        System.out.print("Enter first winner name: ");
        str = in.nextLine();
        // כל עוד לא עברנו את שם המועמד
        while (name.compareTo(str) > 0)
        {
            counter++;
            System.out.print("Enter next winner name: ");
            str = in.nextLine();
        } // while
        if (name.equals(str))
             System.out.println(name + " is accepted");
        else
             System.out.println(name + " is not accepted");
        System.out.println(counter + " Names");
    } // main
 } // class NameFinder
```

שימו ♥: לולאת ה-while ממשיכה כל עוד מחרוזות הקלט קטנות משם המועמד. קליטת מחרוזת שווה לשם המועמד או גדולה ממנה גורמת ליציאה מהלולאה. לכן, אחרי הלולאה עלינו לבדוק את סיבת היציאה מהלולאה: האם המחרוזת האחרונה שנקלטה שווה לשם המועמד או גדולה ממנה.

סוף פתרון בעיה 3

שאלה 9.3

פתחו אלגוריתם המקבל כקלט רשימת מחרוזות המסתיימת במחרוזת "***". פלט האלגוריתם יהיה אורך המחרוזת הארוכה ביותר ברשימה. ישמו את האלגוריתם בשפת Java.

שאלה 9.4

פתחו אלגוריתם שיקבל כקלט שתי מחרוזות ויציג אותן לפי סדר האי"ב. ישמו את האלגוריתם בשפת Java.

שאלה 9.5

פתחו וישמו אלגוריתם שיקבל כקלט מילה באנגלית ויציג אותה פעמיים, פעם באותיות גדולות ופעם באותיות קטנות. למשל עבור הקלט Memory הפלט יהיה: MEMORY ומיד אחר-כך memory. חפשו פעולות מתאימות בטבלת הפעולות המופיעה בסוף הפרק.

שרשור מחרוזות

פעולה נוספת ושימושית מאוד על מחרוזות היא פעולת **הִשְׁרְשׁוּר**. פעולה זו מקבלת שתי מחרוזות ויוצרת מחרוזת חדשה, המורכבת מהמחרוזת הראשונה ואחריה מוצמדת המחרוזת השנייה. הפעולה אינה משנה את המחרוזות המקוריות.

בשפת Java מתבצע שרשור באמצעות סימן הפעולה +. כלומר, אם `s1` ו-`s2` הן מחרוזות, אז כדי לשרשר אותן זו לזו נכתוב את הביטוי `s1+s2`. למשל, אם ב-`s1` נמצאת המחרוזת `"he"` וב-`s2` נמצאת המחרוזת `"llo"`, אז `s1+s2` היא המחרוזת החדשה `"hello"`.

שימו ♥: פעולת השרשור אינה פעולה של המחלקה מחרוזת ולכן אופן הפעלתה שונה מזה של פעולות כמו `length`, כמו `charAt` או כמו פעולות אחרות השייכות למחלקה. היא אינה מופעלת על עצם מהמחלקה, ולכן אין שימוש בסימן הנקודה. אופן הפעלת פעולת השרשור על מחרוזות דומה לאופן הפעלת פעולות מתמטיות על טיפוסים רגילים בשפה.

למעשה כבר השתמשנו פעמים רבות בפעולת השרשור על מחרוזות. כזכור, עד כה השתמשנו במחרוזות בהוראות הדפסה כגון:

```
System.out.println("hello");
```

לעתים, פקודות ההדפסה היו מורכבות מעט יותר. למשל:

```
System.out.println("The number is: " + number) ;
```

או אפילו:

```
System.out.println("There are " + bags + " bags, " + left +
                                                  " are left over.");
```

בכל ההוראות האלו הפעולה `println` מקבלת מחרוזת להדפסה. בדוגמה הראשונה המחרוזת היא `"hello"`. בדוגמאות האחרות מועברת לפעולה `println` מחרוזת שהתקבלה כתוצאה

מהפעלת פעולת שרשור של מחרוזת ושל משתנה מספרי – תוצאת הפעולה מחזירה מחרוזת חדשה. כך למשל המחרוזת ש-`println` קיבלה בכל אחת משתי הדוגמאות האחרות היא בעצם מחרוזת שהתקבלה כתוצאה מהפעלת פעולת השרשור + (פעם אחת בדוגמה השנייה, וארבע פעמים בשלישית).

למשל, אם ערכו של המשתנה `number` הוא 3, אז המחרוזת שהועברה להדפסה בדוגמה השנייה היא שרשור של המחרוזת `"3"` למחרוזת `"The number is: "`, כלומר המחרוזת `"The number is: 3"`.

שימו ♥: מאחר שבמשתנה `number` יש ערך מטיפוס שלם יש להמיר אותו למחרוזת. ואכן טרם השרשור התבצעה פעולת המרה של הערך המספרי למחרוזת המתאימה לו באופן אוטומטי. כך קורה תמיד כאשר נשרשר ערך מטיפוס שאינו מחרוזת למחרוזת. לכן הפעולה `println` מקבלת למעשה מחרוזת אחת שהיא תוצאת השרשור.

הפעולה + משמשת ל**שרשור מחרוזות** באופן הבא: `מחרוזת1 + מחרוזת2`
פעולת השרשור יוצרת מחרוזת חדשה ומקצה עבורה מקום. היא אינה משפיעה על המחרוזות המקוריות.

כאשר משרשרים למחרוזת ערך שאינו מחרוזת, הוא מומר תחילה למחרוזת ואז מתבצעת פעולת השרשור.

שאלה 9.6

פתחו אלגוריתם שיקבל כקלט 100 מחרוזות. עבור כל זוג מחרוזות תוצג כפלט מחרוזת שהיא שרשור של שתי המחרוזות עם הסימן '@' ביניהן (כלומר בסך הכול תוצגנה 50 מחרוזות). ישמו את האלגוריתם בשפת Java.

ומה בפנים? פעולות המסתכלות אל תוך הקנקן

המחלקה מחרוזת מגדירה פעולות שונות המאפשרות להסתכל פנימה לתוך המחרוזת ולהתייחס לתווים המרכיבים אותה. פעולות כאלו, כמו הפעולה `charAt` שכבר הכרנו, מסייעות מאוד בפתרון בעיות שונות, כפי שמדגימה הבעיה הבאה:

בעיה 4

מטרת הבעיה ופתרונה: הדגמת השימוש בפעולות המחלצות מידע מתוך מחרוזת.

פתחו אלגוריתם שהקלט שלו הוא מחרוזת המהווה משפט באנגלית, והפלט הוא האות **האחרונה** של המילה **הראשונה** במשפט ומספר המילים במשפט.
למשל, עבור הקלט: Welcome to Israel and have a nice day! הפלט המתאים הוא e 8.
ישמו את האלגוריתם בשפת Java.
אפשר להניח שהמשפט מכיל לפחות מילה אחת.

ניתוח הבעיה בעזרת דוגמאות

נתבונן במשפט שניתן כדוגמה: Welcome to Israel and have a nice day!.

❓ כיצד אפשר לדעת כמה מילים יש במשפט הנתון?

כיוון שלפני כל מילה פרט למילה הראשונה יש רווח נוכל למנות את מספר הרווחים במחרוזת ולהוסיף 1. למשל, במשפט Welcome to Israel and have a nice day! יש 7 רווחים, ולכן 8 מילים. (זאת בהנחה שבין מילה למילה יש רווח אחד בלבד!)

❓ מהי האות האחרונה במילה הראשונה?

במקרה שהמשפט מכיל מילה אחת בלבד, האות האחרונה במילה הראשונה היא האות האחרונה במחרוזת. בכל שאר המקרים, האות האחרונה במילה הראשונה היא האות שנמצאת מיד לפני הרווח הראשון.

פירוק הבעיה לתת-משימות

לשם פתרון הבעיה נפתור את התת-משימות הבאות:

1. קליטת המחרוזת
2. מניית מספר הרווחים במחרוזת שנקלטה
3. חישוב מספר המילים במחרוזת
4. מציאת התו האחרון של המילה הראשונה במחרוזת
5. הצגה של מספר המילים במחרוזת ושל התו שנמצא בתת-משימה 4

בחירת משתנים

sentence – מחרוזת לשמירת המשפט שנקלט
numOfSpaces –שלם, מונה את מספר הרווחים במשפט
numOfWords – שלם, שומר את מספר המילים במשפט
placeOfLastLetter – שלם, שומר את מיקומו של התו האחרון במילה הראשונה

יישום האלגוריתם

כדי למנות את מספר הרווחים במחרוזת, נעבור על כל תו במחרוזת ונבדוק אם הוא שווה לרווח. לצורך בדיקת התו במיקום ה-i נשתמש בפעולה `charAt` המחזירה את התו במקום המבוקש:

```
if (sentence.charAt(i)==' ')
```

שימו ♥: הפעולה `charAt` מחזירה ערך מטיפוס `char` ולא מטיפוס מחרוזת, ולכן יש להשוות את הערך המוחזר לתו רווח ' ' (ולא למחרוזת המכילה את התו רווח " ").

כדי למצוא את התו האחרון של המילה הראשונה נבדוק את מספר המילים במחרוזת. אם המחרוזת מורכבת ממילה אחת בלבד נחלץ, באמצעות `charAt`, את התו במקום האחרון, אחרת נחלץ את התו שנמצא לפני הרווח הראשון.

כדי למצוא את מיקומו של הרווח הראשון אפשר להשתמש בפעולה `indexOf` המקבלת תו ומחזירה מספר שלם המייצג את מקומו במחרוזת:

```
placeOfLastLetter = sentence.indexOf(' ') - 1;
```

טבלת הפעולות בסוף הפרק מתארת גרסה נוספת של פעולה זו המקבלת מחרוזת ומחזירה את מיקומה בתוך המחרוזת שעליה מופעלת הפעולה.

שימו ♥: יש תמיד לדאוג להתאמה בין טיפוסי הערכים שאנו משתמשים בהם ובין אלו המתוארים בכותרת הפעולה.

התוכנית המלאה

```
/*
קלט: משפט באנגלית
פלט: האות האחרונה במילה הראשונה ומספר המילים במשפט
*/
import java.util.Scanner;
public class HowManyWords
{
    public static void main (String[] args)
    {
        String sentence;
        int placeOfLastLetter;
        int numOfSpaces = 0;
        int numOfWords;
        Scanner in = new Scanner(System.in);
        System.out.println("Enter a sentence, with exactly one " +
                                              "space between words:");
        sentence = in.nextLine();
        // מניית הרווחים במשפט
        for (int i = 0 ; i < sentence.length() ; i++)
            if (sentence.charAt(i) == ' ')
                numOfSpaces++;
        numOfWords = numOfSpaces + 1;
        // מציאת מקומו של הרווח הראשון
        if (numOfWords == 1)
           placeOfLastLetter = sentence.length()- 1;
        else
           placeOfLastLetter = sentence.indexOf(' ') - 1;
        System.out.println("The last letter of the first word is: " +
                                  sentence.charAt(placeOfLastLetter));
        // כמה מילים במשפט
        System.out.println("There are " + numOfWords + " words");
    }//main
}// class HowManyWords
```

סוף פתרון בעיה 4

שאלה 9.7

פתחו וישמו אלגוריתם שהקלט שלו הוא תו ומחרוזת והפלט הוא הודעה אם התו נמצא במחרוזת או לא.

שאלה 9.8 (מתוך בגרות 1995)

פתחו וישמו אלגוריתם הקולט מחרוזת. האלגוריתם מציג כפלט כל תו במחרוזת פעמיים, פרט לתו '*' (כוכבית). גם אם התו כלול במחרוזת הוא לא מוצג כלל. הפלט מוצג בשורה אחת.
למשל עבור הקלט: AB*3B*?? הפלט יהיה AABB33BB??.

שאלה 9.9

פתחו וישמו אלגוריתם הקולט מחרוזת. האלגוריתם מציג כפלט את המחרוזת ללא אותיות זהות צמודות. למשל עבור הקלט: apple הפלט יהיה: aple, עבור הקלט: Yellow balloon הפלט יהיה: Yelow balon ועבור הקלט: abbba הפלט יהיה: aba.

שאלה 9.10

פתחו אלגוריתם שהקלט שלו הוא שלוש מחרוזות, והפלט שלו הוא שלוש המחרוזות לפי סדר מילוני. ישמו את האלגוריתם בשפת Java.

למשל עבור הקלט: apple today good הפלט המתאים הוא: apple good today.

9.3 הוראת השמה במחרוזות

עד כה ראינו כיצד לבצע פעולות קלט ופלט של מחרוזות, פעולת שרשור ופעולות שונות שמחזירות מידע על המחרוזת. בעיבוד של משתנים רגילים, אחת מהפעולות השימושיות ביותר היא הוראת השמה. בסעיף זה נדון בהשמה עבור מחרוזות, כלומר בהשמה של מחרוזת במשתנה שטיפוסו מחרוזת.

למעשה, כפי שכבר אמרנו, גם בעת קליטת מחרוזת אנו מבצעים השמה של המחרוזת הנקלטת, בעצם מסוג מחרוזת. למשל, בהוראה:

```
str = in.nextLine();
```

הפעולה `in.nextLine` מחזירה הפניה לשטח הזיכרון החדש שהוקצה עבור מחרוזת הקלט. בעקבות ההשמה, המשתנה `str` מפנה אל אותו שטח זיכרון.

בסעיף זה נראה דוגמאות נוספות להשמה של מחרוזות.

מצ'ה 5

מטרת הבעיה הבאה: הצגת השמת מחרוזת למחרוזת והדגמה נוספת של השימוש בפעולות של המחלקה String.

נאמר שכתובת דואר אלקטרוני היא חוקית אם היא מקיימת את התנאים הבאים:

מתחילה באות אנגלית, מורכבת מרצף לא ריק של תווים ואחריו מגיע התו '@', אחריו שוב רצף לא ריק של תווים, אחריו התו '.' ולאחריו עוד רצף לא ריק של תווים. כתובת ישראלית מסתיימת במחרוזת "il."

בקביעת חוקיות של כתובת דואר אין משמעות להבדל בין אותיות גדולות לקטנות.

למשל, המחרוזת d@ccv.hhh היא כתובת דואר אלקטרוני חוקית, וכך גם t@יי.IL, שהיא כתובת ישראלית. לעומתן, המחרוזות הבאות אינן כתובות חוקיות: ח@יח.cc (לא מתחילה באות אנגלית), tr@.il (רצף התווים שבין התו '@' לבין התו '.' הוא ריק), rt@jjj (לא מכילה את התו '.') sda.asd@sad (התו '.' מופיע לפני התו '@').

פתחו אלגוריתם המקבל כקלט מחרוזת. ניתן להניח כי במחרוזת המתקבלת התו '.' לא מופיע יותר מפעם אחת וכך גם התו '@'. האלגוריתם מציג כפלט את המחרוזת שנקלטה, בצירוף הודעה המבהירה אם המחרוזת מהווה כתובת דואר אלקטרוני חוקית, ואם כן, אם זוהי כתובת ישראלית.

ישמו את האלגוריתם בשפת התכנות Java.

ניתוח הבעיה בעזרת דוגמאות

הכתובת tr@xxx.IL היא חוקית וישראלית. גם הכתובת tr@xxx.il היא חוקית וישראלית. אין משמעות לשימוש באותיות קטנות או גדולות – ההבדל ביניהן אינו משפיע על קביעת חוקיות המחרוזת וגם לא על סיווגה ככתובת ישראלית. אבל הפלט אינו לגמרי זהה בשני המקרים: אמנם בשני המקרים נציג הודעה כי המחרוזת חוקית וישראלית, אך במקרה הראשון נציג את המחרוזת tr@xxx.IL ובמקרה השני את המחרוזת tr@xxx.il.

כיוון שבשביל לבדוק את חוקיות המחרוזת וכדי לקבוע אם היא ישראלית או לא אין משמעות להבדל בין אותיות גדולות לקטנות, נוכל לפשט את הבדיקה אם ראשית נמיר את המחרוזת למחרוזת זהה המורכבת מאותיות גדולות או קטנות בלבד, ורק אז נבדוק. נחליט כי המחרוזת האחידה תהיה כולה אותיות קטנות. את מחרוזת הקלט המקורית נשמור כמו שהיא, כדי שנוכל להציגה כפלט.

קל לבדוק אם התו הראשון הוא אות אנגלית. ברגע שבדיקה זו הצליחה, כבר ברור שהמילה אינה מתחילה בתו '@', כלומר שהרצף שלפני התו '@' אינו ריק.

פירוק הבעיה לתת-משימות

1. קליטת מחרוזת
2. יצירת מחרוזת זהה לזו שנקלטה ובה כל האותיות הן אותיות קטנות בלבד
3. בדיקה שהתו הראשון הוא אות אנגלית
4. בדיקה שהתווים '@' ו-'.' מופיעים במחרוזת
5. בדיקה שהתו '.' מופיע אחרי התו '@', אך לא מיד אחריו ולא בסוף המחרוזת
6. עבור מחרוזת חוקית: בדיקה אם היא ישראלית
7. הצגה של המחרוזת ושל הודעה מלווה מתאימה

בחירת משתנים

בשאלה מתוארים כמה תנאים שמחרוזת חוקית צריכה לעמוד בהם. מספיק שאחד מהם לא מתקיים כדי לקבוע שהמחרוזת אינה חוקית. נוכל להיעזר במשתנה בוליאני: כל עוד לא מצאנו בעיה בכתובת הדואר האלקטרוני ערכו של המשתנה יהיה **true**. כאשר נמצא שגיאה באחד התנאים נציב בו **false**. ההודעה לפלט תוצג לפי ערכו של המשתנה.

בנוסף נזדקק לשתי מחרוזות: האחת לשמירת מחרוזת הקלט המקורית, והשנייה לשמירת המחרוזת האחידה.

כדי לבדוק את חוקיות המחרוזת נשתמש בשני משתנים שלמים שישמרו את מיקום התווים '.' ו-'@'.

לבסוף, נקדיש משתנה גם לאורך של המחרוזת שנקלוט. כך נוכל לחשב את האורך פעם אחת, ולהשתמש בערך המשתנה בכל הפעמים שנזדקק לאורך המחרוזת.

isLegal – משתנה בוליאני ("דגל"), שיעיד אם הכתובת היא חוקית או לא
str – מחרוזת לשמירת הכתובת הנקלטת
lowerCaseStr – מחרוזת זהה למחרוזת הקלט ובה כל האותיות הן אותיות קטנות
atPlace – מספר שלם, לשמירת מיקומו של התו '@'
dotPlace – מספר שלם, לשמירת מיקומו של התו '.'

len – אורך מחרוזת הקלט

האלגוריתם

1. *אתחל את שדה* isLegal *ב-***true**
2. *קלוט כתובת ב-*str
3. *בנה מחרוזת חדשה, ריקה ל-*str *ובה כל האותיות הגדולות מהמחרוזת בקטנות, והשם אותה ב-*lowerCaseStr
4. *אם האות הראשונה היא לא אות אנגלית*
   4.1. *שנה את שדה* isLegal *ל-***false**
5. *אם התו '.' או התו '@' אינם מופיעים במחרוזת*
   5.1. *שנה את שדה* isLegal *ל-***false**
6. *אחרת*
   6.1. *אם התו '.' מופיע לפני התו '@' או מיד אחריו או בסוף המחרוזת*
      6.1.1. *שנה את שדה* isLegal *ל-***false**
7. *אם ערכו של* isLegal *שווה ל-***true**
   7.1. *הדפס כפלט הודעה כי הכתובת תקינה*
   7.2. *אם שלושת התווים האחרונים הם המחרוזת "*.il*"*
      7.2.1. *הדפס הודעה כי הכתובת ישראלית*
   7.3. *אחרת*
      7.3.1. *הדפס הודעה כי הכתובת אינה ישראלית*
8. *אחרת*
   8.1. *הדפס כפלט הודעה כי הכתובת אינה תקינה*

יישום האלגוריתם

עלינו לבדוק אם התווים '.' ו-'@' מופיעים במחרוזת `str`, ואם כן – אם מיקומם תקין. לשם כך נשתמש בפעולה `str.indexOf` המקבלת תו, ומחזירה את מיקומו במחרוזת `str`.

כדי ליצור ממחרוזת הקלט (השמורה ב-`str`) מחרוזת חדשה הזהה לה ובה כל האותיות הן קטנות, נשתמש בפעולה `str.toLowerCase`. כמו כל פעולה הפועלת על מחרוזות, גם הפעולה `toLowerCase` אינה משנה את המחרוזת שעליה היא מופעלת, אלא יוצרת מחרוזת חדשה. כמו כל פעולה שיוצרת מחרוזת חדשה, הפעולה `toLowerCase` מקצה מקום עבור המחרוזת החדשה.

אנו מעוניינים שהמשתנה `lowerCaseStr` יפנה למחרוזת החדשה. לכן נשתמש במשפט השמה, ונשים את הערך המוחזר מהפעולה `str.toLowerCase` במשתנה `lowerCaseStr`, כך:

```
lowerCaseStr = str.toLowerCase();
```

בעקבות הוראה זאת העצם `lowerCaseStr` מפנה אל שטח בזיכרון, ששמורה בו מחרוזת חדשה, הזהה למחרוזת הקלט ובה כל האותיות הן קטנות. במחרוזת המקורית השמורה ב-`str` לא חל כל שינוי.

לא היינו צריכים להקצות במפורש שטח זיכרון לעצם `lowerCaseStr`, מפני שהפעולה `toLowerCase` הקצתה בעצמה שטח עבור המחרוזת. לאחר פעולת ההשמה העצם `lowerCaseStr` מפנה לשטח זה.

התוכנית המלאה

```
/* התוכנית מקבלת כקלט מחרוזת, מציגה אותה כפלט, ומציגה גם הודעה המבהירה
   אם זו מחרוזת דואר אלקטרוני חוקית, ואם כן, אם היא כתובת ישראלית */
import java.util.Scanner;
public class EmailAddress
{
    public static void main (String[] args)
    {
        String str;               // מחרוזת הקלט
        String lowerCaseStr;      // מחרוזת כמחרוזת הקלט
                                  // ובה כל האותיות קטנות
        int atPlace;              // שומר מיקום התו '@'
        int dotPlace;             // שומר מיקום התו '.'
        boolean isLegal = true;   // דגל חוקיות הכתובת
        int len;                  // אורך המחרוזת שנקלטה
        Scanner in = new Scanner(System.in);
        // קלט מחרוזת ושמירת אורכה
        System.out.print("Enter a valid e-mail address: ");
        str = in.nextLine();
        len = str.length();
        // יצירת המחרוזת החדשה ושמירתה
        lowerCaseStr = str.toLowerCase();
        // בדיקה שהתו הראשון הוא אות לועזית
        if (!(lowerCaseStr.charAt(0) >= 'a' &&
                                    lowerCaseStr.charAt(0) <= 'z'))
            isLegal = false;
        // '@'-ו '.' מציאת מקום התווים
        atPlace = lowerCaseStr.indexOf('@');
        dotPlace = lowerCaseStr.indexOf('.');
        if ((dotPlace == -1)||(atPlace == -1)) // אחד משני התווים חסר
            isLegal = false;
        else
            if ((dotPlace < atPlace) || (dotPlace == atPlace + 1)
                                        || (dotPlace == len - 1))
                // הסדר בין התווים שגוי או שהם מופיעים ברצף
                // או שהנקודה מופיעה בסוף
                isLegal = false;
        // הפלט
        if (isLegal)
        {
            System.out.println(str + " is a legal Email address");
            // בדיקת שלושת התווים האחרונים במחרוזת, האם שווים ל-il.
            if (lowerCaseStr.indexOf(".il") == len - 3)
                System.out.println("Email address is Israeli");
            else
                System.out.println("Email address is not Israeli");
        }
        else
            System.out.println(str +" is not a valid Email address");
    } // main
} // EmailAddress
```

סוף פתרון בעיה 5

השמת מחרוזות נכתבת כהוראת השמה רגילה:

```
s1 = s2;
```

בעקבות ביצוע ההשמה, מפנה `s1` אל אותו שטח זיכרון שאליו מפנה `s2`. לכן אין צורך לבצע קודם הקצאת זיכרון עבור `s1`.

שאלה 9.11

פתחו אלגוריתם המקבל כקלט רשימת מחרוזות המסתיימת במחרוזת "stop". פלט האלגוריתם יהיה המחרוזת הארוכה ביותר ברשימה. ישמו את האלגוריתם בשפת Java.

שאלה 9.12

פתחו אלגוריתם המקבל מחרוזת כקלט ומציג אותה בסדר הפוך. למשל עבור הקלט university הפלט יהיה: ytisrevinu. אין צורך ליצור מחרוזת הפוכה אלא רק להציג את התווים שלה בסדר הפוך. ישמו את האלגוריתם בשפת Java.

תרגיל 6

מטרת הבעיה ופתרונה: הדגמת בניית מחרוזת בשלבים.

פתחו אלגוריתם אשר הקלט שלו הוא מחרוזת. הפלט יהיה מחרוזת חדשה, הזהה למחרוזת המקורית, פרט לכך שמופיעה בה האות c בכל מקום שהופיעה האות b או B במחרוזת המקורית. למשל עבור המחרוזת "abcd" הפלט יהיה המחרוזת "accd".

שימו ♥: פרט לשינוי המתואר, המחרוזת החדשה צריכה להיות זהה למקורית. לא ניתן להפך בה אותיות גדולות לקטנות או להפך.

פירוק הבעיה לתת-משימות

1. קליטת מחרוזת
2. בניית המחרוזת החדשה
3. הדפסת המחרוזת החדשה

את תת-משימה 2 נוכל לפרק באופן הבא:

2.1. יצירת עצם מסוג מחרוזת ואתחולו במחרוזת ריקה
2.2. בנייה הדרגתית של המחרוזת החדשה, תו אחר תו

את תת-משימה 2.2 נוכל ליישם בלולאת **for** ומספר הפעמים לביצועה נקבע לפי אורך המחרוזת.

בחירת משתנים

str – המחרוזת שמתקבלת כקלט
newStr – המחרוזת החדשה

יישום האלגוריתם

אתחול העצם `newStr` במחרוזת ריקה (תת-משימה 2.1) אפשר לבצע יחד עם הצהרת המחרוזת:

```
String newStr = new String("");
```

את תת-משימה 2.2 נוכל ליישם בלולאת **for**, שמספר הפעמים לביצועה נקבע לפי אורך המחרוזת. בכל פעם נבדוק את התו הבא במחרוזת באמצעות הפעולה charAt. ונבנה את המחרוזת החדשה באמצעות פעולת השרשור, למשל כך:

```
newStr = newStr + 'c';
```

שימו ♥: פעולת השרשור היא פעולה שמחזירה מחרוזת. כמו כל פעולה שמחזירה מחרוזת, פעולת השרשור אינה משנה את המחרוזות שהיא פועלת עליהן, אלא יוצרת מחרוזת חדשה (כמובן אחרי שהקצתה מקום עבורה), המתקבלת משרשור המחרוזות המקוריות.

נניח למשל שב-newStr נמצאת המחרוזת "ad". מה קורה בעת ביצוע הוראת ההשמה שלעיל?

פעולת השרשור מקבלת שתי מחרוזות (במקרה זה את newStr ואת המחרוזת "c", אחרי המרת התו 'c' למחרוזת) ויוצרת תוך הקצאת מקום מתאים בזיכרון מחרוזת חדשה שמכילה את שרשור שתי המחרוזות, כלומר את המחרוזת "adc". היא מחזירה את המחרוזת החדשה.

בפעולת ההשמה המחרוזת חדשה מושמת ב-newStr. כלומר במקום ש-newStr יַפנה אל המחרוזת המקורית ("ad"), הוא מפנה כעת אל השטח החדש בזיכרון. מה קרה למחרוזת המקורית? ניתן לומר שהיא הלכה לאיבוד, מכיוון שאין הפניות אליה. בכך פעולת השמה של מחרוזות אינה שונה מפעולת השמה רגילה של משתנים: גם כאשר אנו מבצעים השמה כגון x = y למשתנים x ו-y שהם משתנים רגילים, אז ערכו המקורי של x אובד והערך החדש של y מחליף אותו.

כאשר נשרשר למחרוזת תו נוסף, אותו תהליך יחזור על עצמו: הקצאת שטח למחרוזת חדשה שתהיה השרשור של המחרוזת המקורית לתו החדש, והשמה הגורמת ל-newStr להפנות אל השטח בזיכרון של המחרוזת החדשה.

אם כך בבנייה הדרגתית של מחרוזת כפי שתואר לעיל, לא יהיה מדויק לומר ש**אותה** מחרוזת גדלה כל פעם בתו נוסף. למעשה, נוצרת סדרה של מחרוזות: בכל פעם נוצרת מחרוזת חדשה, ארוכה בתו אחד, והיא תופסת את מקומה של המחרוזת הקודמת.

התוכנית המלאה

```
/*
קלט: מחרוזת
פלט: מחרוזת זהה למקורית, ובמקום כל 'B' או 'b' מופיע 'c'
*/
import java.util.Scanner;
public class ReplaceCForB
{
    public static void main (String[] args)
    {
        String str;            // מחרוזת הקלט
        String newStr = new String("");    // המחרוזת החדשה
                                           //כעצם המאותחל במחרוזת ריקה
        // קלט
        Scanner in = new Scanner(System.in);
        System.out.print("Enter a string: ");
        str = in.nextLine();
        // בניית המחרוזת החדשה
        for(int i = 0; i < str.length(); i++)
        //סריקת מחרוזת הקלט תו אחר תו
        {
```

```
            if ((str.charAt(i) == 'b') || (str.charAt(i) == 'B'))
                newStr = newStr + 'c'; // 'c'-ב 'b' או 'B' החלפת
            else
                newStr = newStr + str.charAt(i); // העתקת התו המקורי
        } // for
        // פלט
        System.out.println("The new String is: " + newStr);
    } // main
} // class ReplaceCForB
```

סוף פתרון בעיה 6

כאשר אנו נדרשים **לבנות מחרוזת חדשה בשלבים, תו אחרי תו** ניתן לעשות זאת באופן הבא: ליצור עצם שיהיה מחרוזת חדשה ולאתחל אותו במחרוזת ריקה (""); באמצעות הוראה לביצוע-חוזר, לבצע בכל פעם שרשור של המחרוזת הקיימת לתו החדש, והחלפת המחרוזת הקיימת במחרוזת החדשה שהתקבלה מפעולת השרשור.

שאלה 9.13

פתחו וישמו אלגוריתם שהקלט שלו הוא מחרוזת המהווה משפט משובש: במקום כל רווח מופיע הסימן $. האלגוריתם מייצר מחרוזת **חדשה** ובה המשפט התקני, ומציג אותו כפלט. ישמו את האלגוריתם בשפת Java.

שאלה 9.14

אורי ונעמי המציאו שפה מוצפנת. המשפטים הניתנים להצפנה כוללים מילים ורווחים בלבד, בהינתן שכל מילה נכתבת רק באותיות אנגליות קטנות.
ההצפנה מתבצעת באופן הבא: כל רווח מוחלף בסימן '@', ולאחר כל אות מופיעה האות האנגלית הגדולה המתאימה לה.
למשל, המשפט good morning מוצפן כך gGoOoOdD@mMoOrRnNiInNgG.
פתחו אלגוריתם המקבל משפט כקלט ומציג את המשפט המוצפן המתאים לו כפלט. ישמו את האלגוריתם בשפת Java.

שאלה 9.15

פתחו אלגוריתם שמקבל כקלט מחרוזת, מייצר מחרוזת חדשה ובה סדר התווים הפוך למחרוזת המקורית, ומציג את המחרוזת החדשה כפלט. בנוסף האלגוריתם בודק אם שתי המחרוזות (המקורית וההפוכה) שוות זו לזו. במילים אחרות האלגוריתם בודק אם המחרוזת המקורית היא פלינדרום. האלגוריתם מציג כפלט הודעה מתאימה לתוצאת הבדיקה.
למשל: עבור הקלט: dog יהיה הפלט: god – not a palindrome.
עבור הקלט: aba יהיה הפלט: aba – a palindrome.
ישמו את האלגוריתם בשפת Java.

שאלה 9.16

תלמידי הכיתה התעניינו לדעת למי יש סבתא בשם הארוך ביותר. פתחו אלגוריתם אשר יקבל כקלט את מספר התלמידים בכיתה, ולאחר מכן רשימה של שמות הסבתות של תלמידי הכיתה כמחרוזות. אורך הרשימה כמספר תלמידי הכיתה. פלט האלגוריתם יהיה השם הארוך ביותר.
ישמו את האלגוריתם בשפת Java.

שאלה 9.17

כתובת אתר אינטרנט של חברה מסחרית בינלאומית בנויה בדרך כלל מ-3 חלקים המופרדים בנקודות:

www.שם החברה.com

פתחו אלגוריתם המקבל כקלט כתובת של אתר של חברה מסחרית בינלאומית, במבנה שתואר לעיל, ומציג כפלט את שם החברה בלבד. ישמו את האלגוריתם בשפת Java.

הדרכה: השתמשו בפעולה `substring` שבטבלת הפעולות הנמצאת בסוף הפרק.

שאלה 9.18

כתובת אתר אינטרנט של חברה מסחרית שאינה בינלאומית בנויה בדרך כלל מ-3 חלקים המופרדים בנקודות:

www.שם החברה.סיומת המדינה

פתחו אלגוריתם המקבל כקלט רשימת כתובות של אתרי חברות מסחריות כאלה (כל אחת מהן במבנה שתואר לעיל). הרשימה תסתיים במחרוזת "end". האלגוריתם יציג כפלט עבור כל חברה את שמה ואת סיומת המדינה שלה. ישמו את האלגוריתם בשפת Java.

שאלה 9.19

יעל ועומרי המציאו משחק. כל אחד בתורו רושם משפט המתחיל במילה האחרונה של המשפט מהתור הקודם. המשפט הראשון במשחק יתחיל במילה "start". למשל:

start the game
game is one of the best ways to kill time
time is money

פתחו אלגוריתם המסייע למשחקים: האלגוריתם מקבל כקלט את מספר התורות המבוקש. בתחילת כל תור הוא מציג את המילה שצריך להתחיל בה המשפט הבא, ולאחר מכן הוא מקבל כקלט את המשפט החדש. המשחק מסתיים כאשר מספר התורות הסתיים, או כאשר המשפט שבחר אחד השחקנים מתחיל במילה שגויה. ישמו את האלגוריתם בשפת Java.

סיכום

בפרק זה הכרנו את המחלקה String ולמדנו כיצד לעבד מחרוזות.

בשפת Java מחרוזת אינה טיפוס פשוט, כמו **`int`** או **`char`**. הטיפוס החדש מוגדר בשפה כ**מחלקה בשם String**, ומחרוזות הן **עצמים** של מחלקה זו.

למעשה כבר הכרנו עצמים כאשר השתמשנו במחלקה Random ליצירת מספרים אקראיים. עצמים הם משתנים של הטיפוס החדש ואפשר להפעיל עליהם פעולות המוגדרות במחלקה. אופן השימוש בעצמים שונה מהשימוש במשתנים מהטיפוסים הפשוטים המוכרים לנו.

הצהרה על עצם ממחלקה מסוימת דומה להצהרה על משתנה מטיפוס סטנדרטי (כגון **`int`** או **`char`**). למשל:

```
String str;
```

בניגוד להצהרה על משתנים מטיפוסים סטנדרטיים, אחרי הצהרה על עצם צריך גם ליצור אותו באמצעות הפעולה **`new`**. יצירת העצם כוללת **הקצאת שטח זיכרון ואתחול העצם**.

למשל, ההוראה הבאה מקצה מקום עבור `str` ומאתחלת אותו כך שישמור את המחרוזת `"ab"`:

```
str = new String("ab");
```

מכיוון שהשימוש במחרוזות נפוץ כל כך, שפת Java מאפשרת ליצור עצמים מסוג מחרוזת ולאתחל אותם בצורה ישירה ללא שימוש בפעולה **new**. למשל כך:

```
String s1 = "xyz";
```

מיקום התווים במחרוזת מתחיל ב-0. בכל מחרוזת, אם נסמן את אורך המחרוזת ב-len, אז התו האחרון במחרוזת נמצא במיקום len-1, והתו הראשון נמצא במיקום ה-0.

במחלקה String מוגדרות **פעולות שאפשר לבצע על מחרוזות**, כלומר על עצמים של המחלקה.

לכל פעולה מוגדרים סוגי הערכים (הפרמטרים) שהיא מצפה לקבל, וכן מוגדר טיפוס הערך המוחזר ממנה. חשוב לוודא התאמה של טיפוסי הפרמטרים המועברים לפעולה לאלה שהיא מצפה לקבל, ושל טיפוס הערך המוחזר ממנה לביטוי שהוא משולב בו. למשל, אם אנו מבצעים השמה של הערך המוחזר מפעולה במשתנה, יש לוודא התאמה של טיפוס המשתנה לטיפוס הערך המוחזר. יש פעולות שלא מצפות לקבל פרמטרים ופעולות שלא מחזירות ערך.

כדי לציין ביצוע פעולה של עצם מהמחלקה משתמשים **בסימון הנקודה**: ראשית נכתוב את שם העצם, אחריו נכתוב נקודה, לאחר מכן נכתוב את שם הפעולה לביצוע ומיד אחר כך מפורטים בסוגריים הערכים המועברים לפעולה. למשל:

```
len = str.length();
letter = str.charAt(5);
```

עבור עצמים של המחלקה String מוגדרות פעולות רבות, המשמשות אותנו ביישום אלגוריתמים הקשורים למחרוזות. למחלקה זו פעולות רבות נוספות פרט לאלו שהוצגו בפרק. תוכלו למצוא הרחבה על כל אחת מהפעולות בקישור הזה:
http://java.sun.com/j2se/1.5/docs/api/java/lang/String.html
ביחידה זו נשתמש רק בפעולות שהוצגו בפרק זה.

הפעולות שמחזירות מחרוזת, אינן משנות את המחרוזת שהופעלו עליה. הן יוצרות מחרוזת חדשה ומקצות שטח עבורה.

לביצוע **קלט של מחרוזת** השתמשנו בפעולה `nextLine` או `next` של המחלקה Scanner. גם פעולה זו מקצה שטח זיכרון עבור המחרוזת שנקלטה.

בעקבות ביצוע **השמה של מחרוזת** אל מחרוזת, `s1 = s2`, מפנה `s1` אל אותו שטח זיכרון שאליו מפנה `s2`. לכן אין צורך לבצע קודם הקצאת זיכרון עבור `s1`.

בנוסף לפעולות המוגדרות במחלקה String למדנו בפרק זה על **פעולת השרשור**. פעולה זו מקבלת שתי מחרוזות ויוצרת מחרוזת חדשה והיא השרשור של שתי המחרוזות. זוהי פעולה שימושית מאוד לצורך הדפסה. פעולת השרשור אינה פעולה של המחלקה `String` ולכן בכתיבתה אין שימוש בסימון הנקודה. סימן פעולת השרשור הוא הסימן `+`.

בפרקים הבאים נלמד להגדיר מחלקות בעצמנו, ולא רק להשתמש במחלקות קיימות.

רשימת פעולות על מחרוזות

הפעולה	תיאור הפעולה	טיפוס הערך המוחזר	דוגמאות	
			הפעולה	הערך המוחזר
`length()`	פעולה המחזירה את אורכה של המחרוזת (מספר התווים במחרוזת).	שלם	`s1.length()` כאשר ב-`s1` נמצאת המחרוזת `"dog"`	3
			`s1.length()` כאשר ב-`s1` נמצאת המחרוזת `""`	0
`charAt(int index)`	פעולה המחזירה את התו הנמצא במחרוזת במיקום (אינדקס) הנתון.	תו	`s1.charAt(2)` כאשר ב-`s1` נמצאת המחרוזת `"shalom"`	'a'
`indexOf(String s)`	פעולה המקבלת מחרוזת או תו, ומחפשת בתוך המחרוזת שעליה מופעלת הפעולה את המיקום הראשון שבו מופיעה המחרוזת או התו שהתקבלו. הפעולה תחזיר את המיקום. היא תחזיר את הערך 1-, אם החיפוש נכשל.	שלם	`s1.indexOf("pl")` כאשר ב-`s1` נמצאת המחרוזת `"people"`	3
`indexOf(char c)`			`s1.indexOf('o')` כאשר ב-`s1` נמצאת המחרוזת `"people"`	2
`equals(String s)`	פעולה המקבלת מחרוזת, ומחזירה **`true`** אם המחרוזת שעליה הופעלה הפעולה והמחרוזת שהתקבלה שוות זו לזו בדיוק. אחרת, היא מחזירה **`false`**.	בוליאני	`s1.equals(s2)` כאשר ב-`s1` נמצאת המחרוזת `"love"` וב-`s2` נמצאת המחרוזת `"Love"`	**`false`**
			`s1.equals(s2)` כאשר ב-`s1` נמצאת המחרוזת `"love"` וב-`s2` נמצאת המחרוזת `"love"`	**`true`**
`compareTo(String s)`	פעולה המקבלת מחרוזת, ומשווה אותה למחרוזת שעליה הופעלה הפעולה. אם הן שוות זו לזו מוחזר הערך אפס. אם המחרוזת שעליה מופעלת הפעולה **קודמת** למחרוזת שהתקבלה בסדר **מילוני**, יוחזר מספר שלם שלילי. אם המחרוזת שעליה מופעלת הפעולה, מופיעה **אחרי** המחרוזת שהתקבלה בסדר מילוני, יוחזר מספר שלם חיובי.	שלם	`s1.compareTo(s2)` כאשר ב-`s1` נמצאת המחרוזת `"aa"` וב-`s2` נמצאת המחרוזת `"ab"`	מספר שלילי כלשהו

`toLowerCase()`	פעולה שיוצרת מחרוזת זהה למחרוזת שעליה היא מופעלת, ובה כל האותיות מוחלפות באותיות קטנות. הפעולה מחזירה את המחרוזת החדשה.	מחרוזת	`s1.toLowerCase()` כאשר ב-`s1` נמצאת המחרוזת `"Peace"`	המחרוזת החדשה `"peace"`
`toUpperCase()`	פעולה שיוצרת מחרוזת זהה למחרוזת שעליה היא מופעלת, ובה כל האותיות מוחלפות באותיות גדולות. הפעולה מחזירה את המחרוזת החדשה.	מחרוזת	`s1.toUpperCase()` כאשר ב-`s1` נמצאת המחרוזת `"Peace"`	המחרוזת החדשה `"PEACE"`
`substring(int k)`	פעולה שיוצרת מחרוזת זהה לתת-מחרוזת המתחילה מהמקום ה-`k` של המחרוזת שעליה הפעולה מופעלת ועד סופה. הפעולה מחזירה את המחרוזת החדשה.	מחרוזת	`s1.substring(4)` כאשר ב-`s1` נמצאת המחרוזת `"GoodBye"`	המחרוזת החדשה `"Bye"`
`substring(int k, int s)`	פעולה שיוצרת מחרוזת זהה לתת-מחרוזת המתחילה מהמקום ה-`k` של המחרוזת שעליה הפעולה מופעלת, ועד המקום ה-(`s-1`). הפעולה מחזירה את המחרוזת החדשה. שימו לב שעל ערכו של `s` להיות קטן או שווה לאורך המחרוזת.	מחרוזת	`s1.substring(4,10)` כאשר ב-`s1` נמצאת המחרוזת `"GoodBye Israel"`	המחרוזת החדשה `"Bye Is"`