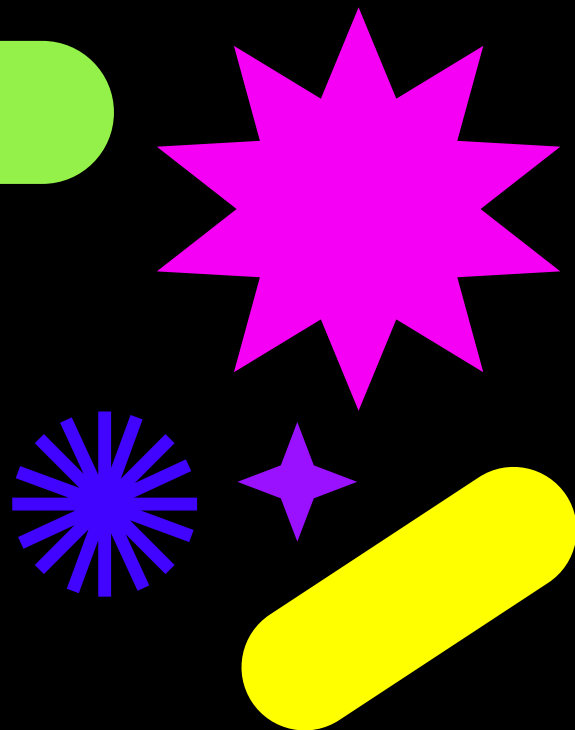


יסודות מדעי המחשב

מחרוזות

דפנה לוי רשתי



בתוכניות שכתבנו עד כה השתמשנו בטיפוסים שונים המוגדרים בשפת java: **שלם**, **ממשי**, **תו** ו**בוליאני**. יכולנו להגדיר משתנים מטיפוסים אלו ולבצע עליהם פעולות שונות (קלט, פלט, חישובים וכו').

עם זאת מרבית התוכניות שכתבנו התייחסו גם **למחרוזות**, **סדרות של תווים**. עד עתה השתמשנו במחרוזות כאשר רצינו להדפיס הודעות למשתמש ותחמנו אותן בגרשיים, למשל בהוראה:

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

ההודעה "Enter two numbers:" היא מחרוזת.

הנה דוגמאות נוספות למחרוזות בשפת Java:

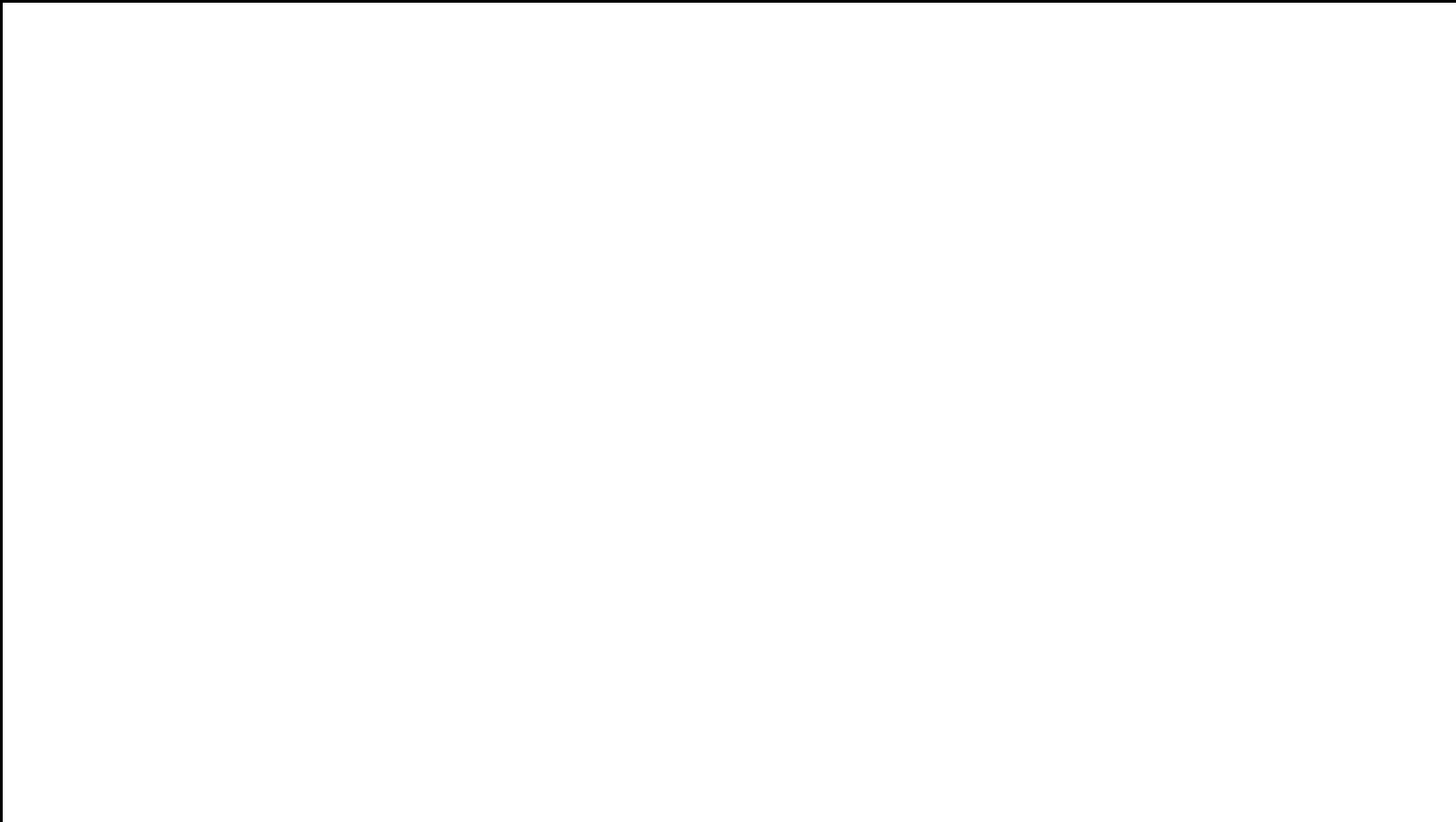
"How Are You?" , "Hello" , "453" , "J" , "" , ""

(שתי האחרונות: "" היא מחרוזת ריקה, שאינה מכילה אף תו,
" " מחרוזת שיש בה תו רווח)

תו ומחרוזת

char: תו - סימן בודד (אות, ספרה, סימן פיסוק, רווח, 😊 ועוד)
String: מחרוזת - רצף של תווים

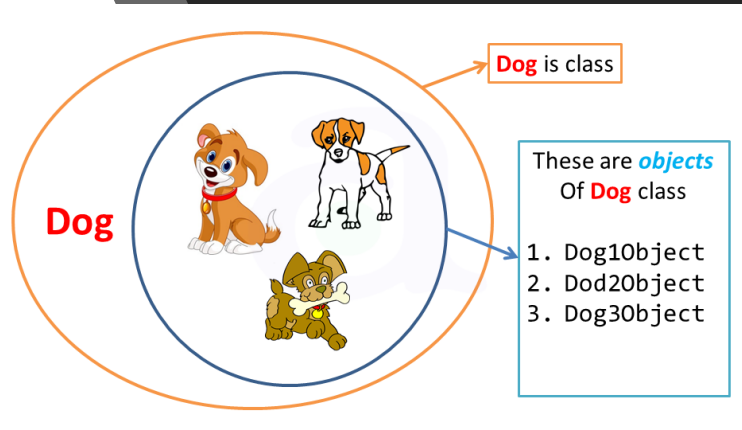
תו מוגדר באמצעות הטיפוס char
מחרוזת מוגדרת באמצעות הטיפוס String





המחלקה בספריה של שרותים

- ניתן לראות במחלקה ספריה של שרותים, מודול, אוסף של פונקציות עם מכנה משותף.
- רוב המחלקות ב Java, נוסף על היותן ספריה, משמשות גם כטיפוס נתונים. ככאלו הן מכילות רכיבים נוספים פרט לשרותי מחלקה.
- קיימות מחלקות המשמשות כספריות בלבד כמו `java.lang.System`
- ספריות הכלולות בחבילה `java.lang` מיובאות אוטומטית



המחלקה כטיפוס נתונים

- טיפוס מוכר לנו בהקשר של סוג הערך של משתנים :
int, double, boolean, char
אלו הם טיפוסים פרימיטיביים.
- הטיפוסים נבדלו בטווח הערכים שלהם ובפעולות שניתן לבצע עליהם
- הכרנו גם טיפוסים מורכבים יותר :
String, Scanner
- הטיפוסים המורכבים הם מחלקות ב-Java.
- כל טיפוס מגדיר את המשתנים הנחוצים לו ואת הפעולות שניתן לבצע על כל אחד מהמופעים של הטיפוס - כך למשל כל עצם ממחלקת Scanner מסוגל לטפל בקלט.

המחלקה הראשית - מחלקה שיש בה את הפעולה הראשית main

```
import java.util.Scanner
```

```
public class Main {
```

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

```
Scanner scan = new Scanner(System.in);
```

```
int first, second;
```

```
String str = "Please enter ";
```

```
System.out.print (str + "1st number:");
```

```
first = scan.nextInt();
```

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

```
System.out.print (str + "2nd number:");
```

```
second = scan.nextInt();
```

```
System.out.println (first + second);
```

```
}
```

```
}
```

הפעולה הראשית main - מבצעת את האלגוריתם של התוכנית

על מנת להשתמש במחלקת Scanner, עשינו import למחלקה

הכרזה על משתנים מטיפוסים שונים. int טיפוס פרימיטיבי, String ו Scanner הם מטיפוס מחלקה. שימו לב לאות הגדולה

```

import java.util.Scanner

public class Main {

    public static void main (String[] args) {

        Scanner scan = new Scanner(System.in);

        int first, second;

        String str = "Pleae enter ";

        System.out.print (str + 1st number:");

        first = scan.nextInt();

        System.out.println();

        System.out.print (str + "2nd number:");

        second = scan.nextInt();

        System.out.println (first + second);

    }
}

```

על מנת להשתמש ב
Scanner בנינו עצם
חדש מטיפוס המחלקה

לא בנינו עצם מטיפוס String

לא עשינו import ל System ולא
ל String

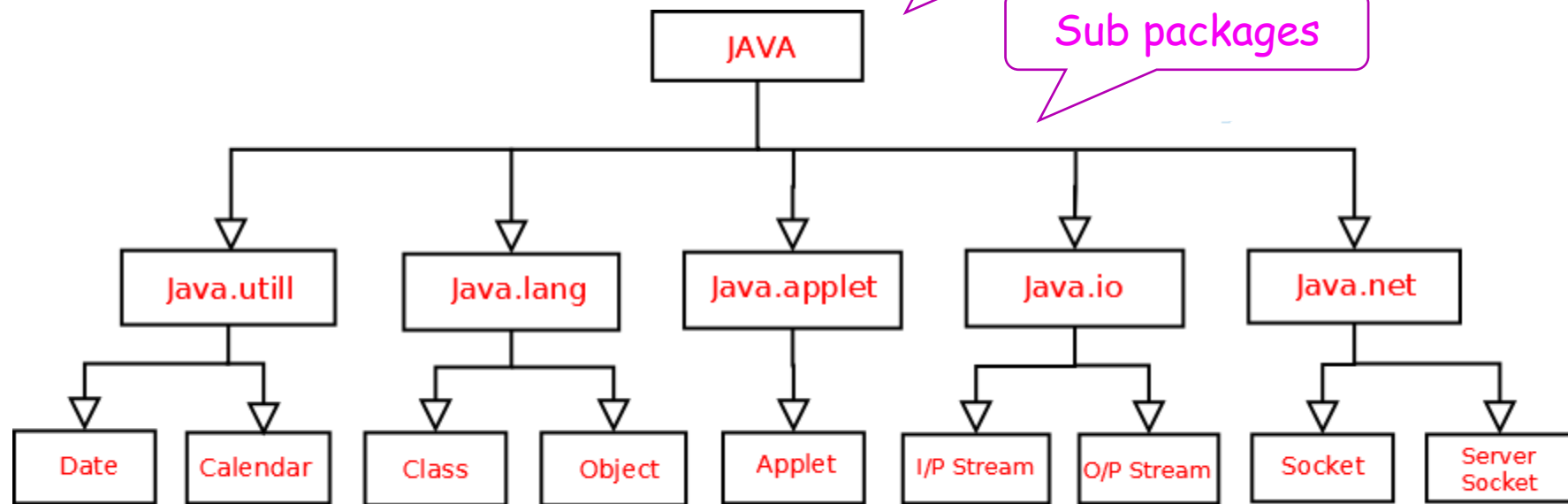
השתמשנו בפעולת המחלקה על
העצם מטיפוס המחלקה

השתמשנו בפעולת המחלקה ע"י שם
המחלקה

ג'אווה בנוייה בצורה היררכית ויש בה
חבילות הכוללות מחלקות מובנות מראש

Java package

Sub packages



Classes

המחלקה מחרוזת

ב- java מוגדרת המחלקה (Class) מחרוזת

משתנה מטיפוס מחרוזת אינו משתנה פשוט אלא **עצם (מופע) של המחלקה String** - שימו לב לאות S !

הטיפול בערכים שהם עצמים שונה מהטיפול בערכים מן הטיפוסים הבסיסיים (כמו char ו- int)

למחרוזת מאפיינים ייחודיים (פעולה בונה, שרשור ע"י +)
בכל מחלקה, גם למחלקה זו יש הגדרת תכונות ופעולות, שאת חלקן נלמד.

ניתן גם לבנות מחרוזת תוך שימוש בפעולות בונות הקיימות במחלקה, כמו :

```
String str1 = new String("abc");  
String str2 = new String ();
```

<https://docs.oracle.com/javase/10th.gnirtS/gnal/avaj/ipa/scod/8>

Field Summary

Fields

Modifier and Type	Field and Description
static <code>Comparator<String></code>	CASE_INSENSITIVE_ORDER A Comparator that orders String objects as by compare

Constructor Summary

Constructors

Constructor and Description
String() Initializes a newly created String object so that it represents an empty character
String(byte[] bytes) Constructs a new String by decoding the specified array of bytes using the platf
String(byte[] bytes, Charset charset) Constructs a new String by decoding the specified array of bytes using the spec
String(byte[] ascii, int hibyte) Deprecated. This method does not properly convert bytes into characters. As of JDK 1.1, the p charset.
String(byte[] bytes, int offset, int length) Constructs a new String by decoding the specified subarray of bytes using the p

Method Summary

All Methods	Static Methods	Instance Methods	Concrete Methods	Deprecated Methods
Modifier and Type		Method and Description		
char		charAt(int index) Returns the char value at the specified index.		
int		codePointAt(int index) Returns the character (Unicode code point) at the specified index.		
int		codePointBefore(int index) Returns the character (Unicode code point) before the specified index.		
int		codePointCount(int beginIndex, int endIndex) Returns the number of Unicode code points in the specified text range of this String.		
int		compareTo(String anotherString) Compares two strings lexicographically.		
int		compareToIgnoreCase(String str) Compares two strings lexicographically, ignoring case differences.		
String		concat(String str) Concatenates the specified string to the end of this string.		
boolean		contains(CharSequence s) Returns true if and only if this string contains the specified sequence of char values.		
boolean		contentEquals(CharSequence cs) Compares this string to the specified CharSequence.		
boolean		contentEquals(StringBuffer sb) Compares this string to the specified StringBuffer.		

משתנה פשוט / משתנה שהוא הפנייה לעצם

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 איתחולו.

עבור משתנים רגילים, הקצאת מקום בזיכרון נעשית אוטומטית עם ההצהרה עליהם ואילו עבור עצמים **הקצאת הזיכרון והאתחול מתבצעים באמצעות הפעולה new**.

מכיוון שהשימוש במחרוזות נפוץ כל כך, שפת java מאפשרת ליצור ולאתחל מחרוזת בצורה ישירה ללא שימוש ב- new.

הגדרה של משתנה מחרוזת –

שם
המשתנה

הצהרה ללא
מתן ערך

```
String name;  
name = "Dani Lev";
```

השמה של ערך –
בין גרשיים

```
String str = "Hello";
```

הצהרה ומתן
ערך התחלתי

```
String str = "";
```

הצהרה ומתן ערך
התחלתי של
מחרוזת ריקה

• השמת ערך מהקלט למשתנה מטיפוס מחרוזת:

```
String name;
```

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



A straight pink arrow points from the `input.next()` expression to the right, and a curved pink arrow points from the `name` variable back to the assignment statement.

זוהי פעולת `next()` על
העצם `input` מטיפוס `Scanner`
היא מחזירה מחרוזת של תווי הקלט עד לתו רווח הראשון או
`enter`
יש השמה של המחרוזת
למשתנה `name`

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



A straight pink arrow points from the `input.nextLine()` expression to the right.

זוהי פעולת `nextLine()` של המחלקה
העצם `input` מטיפוס `Scanner`
היא מחזירה מחרוזת של תווי הקלט עד
ל `enter` (ירידת שורה)

פעולה מוכרת: שרשור מחרוזות

כיצד נחבר את "sa" ל"ra" כדי לקבל "sara"?
כיצד חיברנו: "The winner is" למשתנה name?

שרשור מחרוזות נעשה על ידי "+"

```
String str = "sa" + "ra";    → "sara"
```

```
str = str + str;    → "sarasara"
```

```
str += "le";    → "sarasarale"
```



פעולה מוכרת: שרשור מחרוזות

הערות:

כאשר משרשרים למחרוזת ערך שאינו מחרוזת, הוא מומר תחילה למחרוזת ואז מתבצעת פעולת השרשור.
אין צורך בפעולת המרה. ההמרה נעשית אוטומטית.

```
int num = 8;
```

```
String str = "sa" + num;      → "sa8"
```

פעולה מוכרת: שרשור מחרוזות



חשוב:

פעולת השרשור אינה "מדביקה" את המחרוזות המקוריות אלא יוצרת מחרוזת חדשה, עצם חדש, שעונה על תנאי השרשור.

פעולת השרשור יוצרת מחרוזת חדשה ומקצה עבודה מקום. היא אינה משפיעה על המחרוזות המקוריות.

פעולה מוכרת: שרשור מחרוזות

פעולת השרשור יוצרת מחרוזת חדשה ומקצה עבודה מקום. היא אינה משפיעה על המחרוזות המקוריות.

```
String str1 = "sara";
```

```
String str2 = "-maya";
```

```
String str = str1 + str2;
```

```
System.out.println(str1); → "sara"
```

```
System.out.println(str2); → "-maya"
```

```
System.out.println(str); → "sara-maya"
```

פעולה מוכרת: שרשור מחרוזות

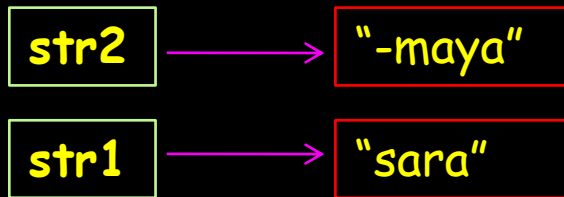
פעולת השרשור יוצרת מחרוזת חדשה ומקצה עבורה מקום. היא אינה משפיעה על המחרוזות המקוריות.

```
String str2 = "-maya";
```

```
String str1 = "sara";
```

```
str1 = str1 + str2;
```

"sara-maya"



חידה – אתחלו את i כך שהלולאה תהיה אין-סופית

```
while(i != i + 0){  
    System.out.println(i);  
}
```



אורך מחרוזת – הפעולה length()

```
String s="ABCD";  
System.out.println(s.length());
```

4

לכל מחרוזת יש אורך
הפעולה length() מחזירה את
אורך המחרוזת.

```
String s="";  
System.out.println(s.length());
```

0

```
String s;  
System.out.println(s.length());
```

Error: variable s might
not have been initialized

```
String s=null;  
System.out.println(s.length());
```

NullPointerException

מחרוזות

מחרוזות הן אובייקטים המכילים רצף של תווים.

```
String s = "Hello";
```

index	0	1	2	3	4
character	H	e	l	l	o

כל אלמנט במחרוזת הוא מסוג char.

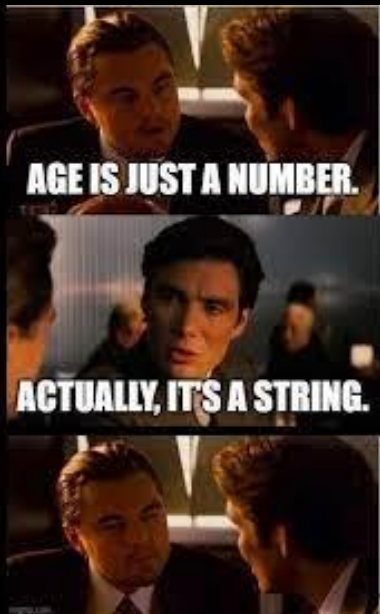
האינדקס של התו הראשון הוא 0.

אורך המחרוזת מוחזר ע"י הפונקציה length()

שרשור מחרוזות נעשה ע"י האופרטור +

```
String s2 = s + " World" + 5 // "Hello World5"
```

```
String s3 = (6 + 5) + " World" // "11World"
```



Methods — length, charAt

`int length();`

- Returns the number of characters in the string

`char charAt(i);`

- Returns the char at position i.



Character positions in strings are numbered starting from 0 – just like arrays.

Returns:

`"Problem".length();`

`"Window".charAt (2);`

7

'n'

לכל תו במחרוזת יש מספר סידורי מ-0 ועד length-1
ניתן "לפנות" לתו בודד באמצעות הפעולה `charAt(pos)`
פנייה לתו שאינו בתחום המחרוזת תגרום לחריגה בזמן ריצה.

דוגמה:

```
String str = "hello world!";
```

```
char c1,c2,c3;
```

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

```
c1=str.charAt(0);
```

```
c2=str.charAt(8);
```

len=12

The diagram illustrates the memory layout of the string "hello world!". It consists of a 2x12 grid of cells. The top row contains the characters 'h', 'e', 'l', 'l', 'o', a space character, 'w', 'o', 'r', 'l', 'd', and '!'. The bottom row contains the corresponding indices from 0 to 11. Two yellow arrows point to the first and ninth cells (indices 0 and 8) from the code above.

h	e	l	l	o		w	o	r	l	d	!
0	1	2	3	4	5	6	7	8	9	10	11

Immutability

לאחר היצירה, לא ניתן לשנות מחרוזת: אף אחת מהשיטות שלה לא משנה את המחרוזת.

אובייקטים כאלה נקראים בלתי ניתנים לשינוי.

אובייקטים בלתי ניתנים לשינוי נוחים מכיוון שמספר הפניות יכולות להצביע על אותו אובייקט בבטחה: אין סכנה לשנות אובייקט באמצעות הפניה אחת מבלי שהאחרים יהיו מודעים לשינוי.

שימו ♥



לא ניתן לעשות שינוי בתווים של מחרוזת קיימת.

הדרך היחידה לשנות ערך של מחרוזת היא ע"י השמה של ערך מחרוזתי חדש.

כלומר, השמה זו היא חוקית:

`str = str + " abc"`

מכיוון שהצבנו ב- `str` ערך חדש.

מעבר על תווי מחרוזת

```
public static void main(String []args){  
    String str = "abcabcdd";  
    char ch = 'a';  
  
    for(int i = 0; i < str.length(); i++){  
        System.out.println(str.charAt(i));  
  
        int i = 0;  
        boolean found = false;  
        while (!found && i < str.length()){  
            found = str.charAt(i) == 'c';  
            i++;  
        }  
        System.out.println("c in Str? " + found);  
    }  
}
```

כיוון שלתווי המחרוזת יש סדר ויש אינדקס, המתחיל ב-0 וערכיו ידועים (קטנים מאורך המחרוזת), ניתן להשתמש בלולאות כדי לרוץ על תווי המחרוזת על פי האינדקס.

```
a  
b  
c  
a  
b  
c  
d  
d  
c in Str? true
```

סריקה של מחרוזת

מנייה של מספר המופעים של תו כלשהו בתוך מחרוזת:

```
public static void main(String[] args){  
    String str = "abc xabc dda";  
    char ch = 'a';  
    int count = 0;  
    for(int i = 0; i < str.length(); i++)  
        if (str.charAt(i) == ch)  
            count++;  
    System.out.println(count);  
}
```

כתבו פעולה המקבלת מחרוזת שהיא משפט,
ומחזירה את מספר המילים במשפט:

```
String str = "I Love java"
```

מספר המילים הוא 3

```
String str = "Wow."
```

מספר המילים הוא 1

```
String str = ""
```

מספר המילים הוא 0



סריקה של מחרוזת

```
public static void main(String[] args){  
    String str = "Separate one page or a whole set for easy conversion into  
        independent PDF files.";  
    System.out.println(countWords(str));  
}  
  
public static int countWords(String str) {  
  
}
```


שימו ♥

הפנייה לתו במחרוזת מחזירה char ולא
String. לכן ' ' ולא " "

סריקה של מחרוזת

```
public static void main(String[] args){  
    String str = "Separate one page or a whole set for easy conversion into  
        independent PDF files.";  
    System.out.println(countWords(str));  
}
```

```
public static int countWords(String str) {  
    if (str == " ") return 0;  
    int count = 0;  
    for (int i=0; i < str.length(); i++)  
        if (str.charAt(i)== ' ') count ++;  
    return count+1;  
}
```



שימו ♥
הפנייה לתו במחרוזת מחזירה char ולא
String. לכן ' ' ולא "

תרגיל

כתבו פעולה המקבלת מחרוזת ומחזירה אמת אם
המחרוזת היא פלינדרום ושקר אחרת.



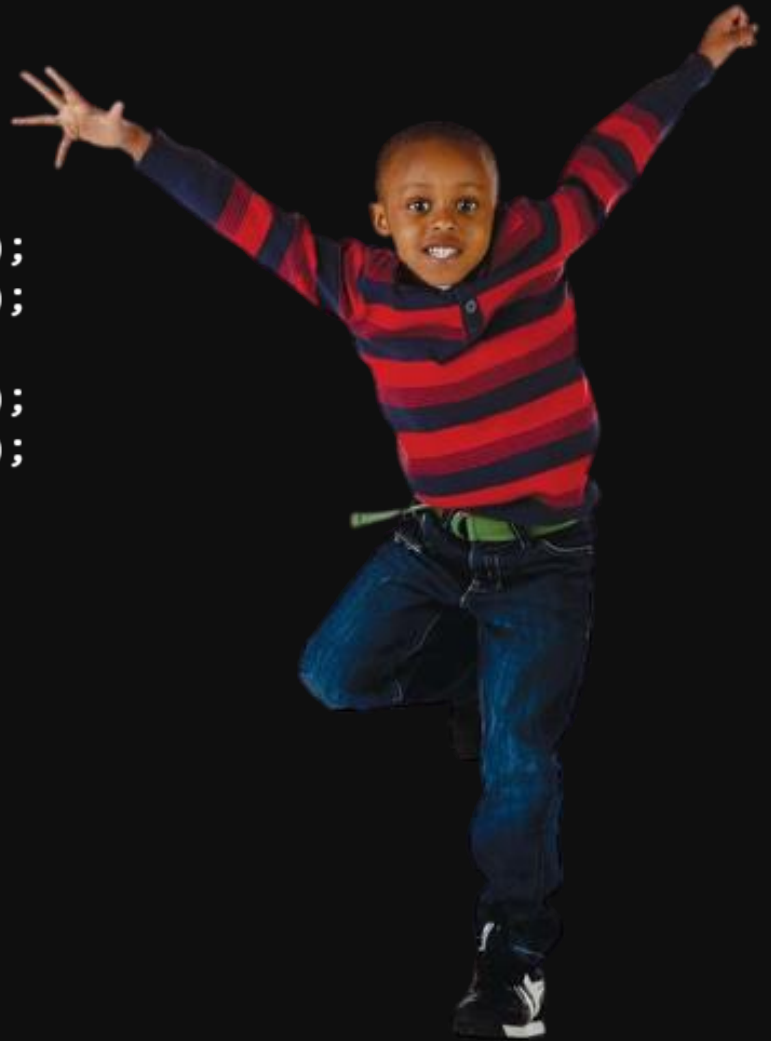
Palindrome

←
WON NOW
→


```
public static boolean isPalindrome1(String str) {  
    for (int i=0, j = str.length()-1; i < j ; i++, j--)  
        if (str.charAt(i) != str.charAt(j))  
            return false;  
    return true;  
}  
  
public static boolean isPalindrome2(String str) {  
    for (int i=0; i < str.length() / 2; i++)  
        if (str.charAt(i) != str.charAt(str.length()-i-1))  
            return false;  
    return true;  
}
```

```
public static void main(String[] args){  
    String str1 = "נטולגלוטנ";  
    System.out.println(isPalindrome1(str1));  
    System.out.println(isPalindrome2(str1));  
    String str2 = "נטול גלוטנ";  
    System.out.println(isPalindrome1(str2));  
    System.out.println(isPalindrome2(str2));  
}
```

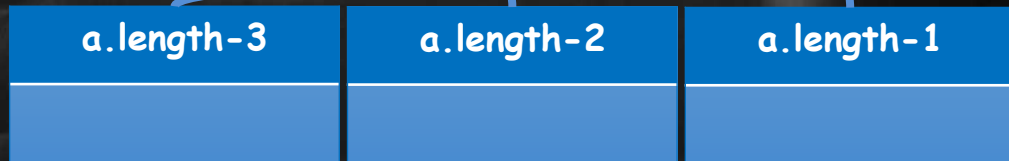
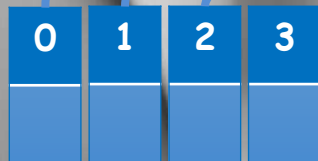
true
true
false
false



$$0 + a.length() - 1 = a.length() - 1$$

$$1 + a.length() - 2 = a.length() - 1$$

$$1 + a.length() - 2 = a.length() - 1$$

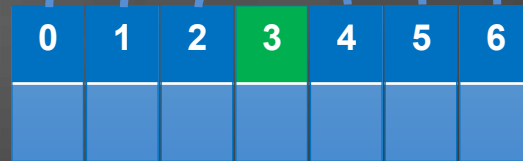


$$\begin{aligned} 0 + 5 &= 5 \\ 1 + 4 &= 5 \\ 2 + 3 &= 5 \end{aligned}$$



$$a.length() / 2$$

$$\begin{aligned} 0 + 6 &= 6 \\ 1 + 5 &= 6 \\ 2 + 5 &= 6 \end{aligned}$$



$$a.length() / 2$$

תרגיל

כתבו פעולה המקבלת מחרוזת ומחזירה את
המחרוזת ההפוכה.



Palindrome

←
WON NOW
→

```
public static void main(String[] args){  
    String str = "$$**##@";  
    String str2 = revString(str);  
    System.out.println(str2);  
}
```

@##**\$\$

```
public static String revString(String str) {  
  
}  
}
```

אין צורך בהמרה - כל מה
שמסורשר למחרוזת, מומר
אוטומטית למחרוזת



```
public static void main(String[] args){  
    String str = "$$**##@";  
    String str2 = revString(str);  
    System.out.println(str2);  
}
```

@##**\$\$

```
public static String revString(String str) {  
    String strNew="";  
    for (int i = str.length() - 1; i >= 0; i--)  
        strNew = strNew + str.charAt(i);  
    return strNew;  
}
```

אין צורך בהמרה - כל מה
שמשורשר למחרוזת, מומר
אוטומטית למחרוזת



```
public static void printStairsFromEnd(String str){
    // הפעולה מקבלת מחרוזת
    // הפעולה מדפיסה את המחרוזת בצורה מדורגת מהסוף להתחלה

}

public static void printStairsFromStart(String str){
    // הפעולה מקבלת מחרוזת
    // הפעולה מדפיסה את המחרוזת בצורה מדורגת מההתחלה לסוף

}
```

```
e
te
ite
hite
white
```

```
hello
ello
llo
lo
o
```



```
public static void printStairsFromEnd(String str){  
    // הפעולה מקבלת מחרוזת  
    // הפעולה מדפיסה את המחרוזת בצורה מדורגת מהסוף להתחלה  
    for (int i = str.length() - 1; i >= 0; i--){  
        System.out.println(str.substring(i));  
    }  
}  
  
public static void printStairsFromStart(String str){  
    // הפעולה מקבלת מחרוזת  
    // הפעולה מדפיסה את המחרוזת בצורה מדורגת מההתחלה לסוף  
    for (int i = 0; i < str.length(); i++){  
        System.out.println(str.substring(i));  
    }  
}
```

```
e  
te  
ite  
hite  
white
```

```
hello  
ello  
llo  
lo  
o
```

```
public static String twiceCharacters(String str){  
    // הפעולה מקבלת מחרוזת  
    // הפעולה מחזירה מחרוזת בה התווים מוכפלים  
  
}
```

```
public static String reverseString(String str){  
    // הפעולה מקבלת מחרוזת  
    // הפעולה מחזירה את המחרוזת ההפוכה  
  
}
```

```
public static String twiceCharacters(String str){  
    // הפעולה מקבלת מחרוזת  
    // הפעולה מחזירה מחרוזת בה התווים מוכפלים  
    String newStr = "";  
    for (int i = 0; i < str.length(); i++)  
        newStr += str.charAt(i) + str.charAt(i);  
    return newStr;  
}
```

```
public static String reverseString(String str){  
    // הפעולה מקבלת מחרוזת  
    // הפעולה מחזירה את המחרוזת ההפוכה  
    String newStr = "";  
    for (int i = str.length() - 1; i >= 0; i--)  
        newStr += str.charAt(i);  
    return newStr;  
}
```

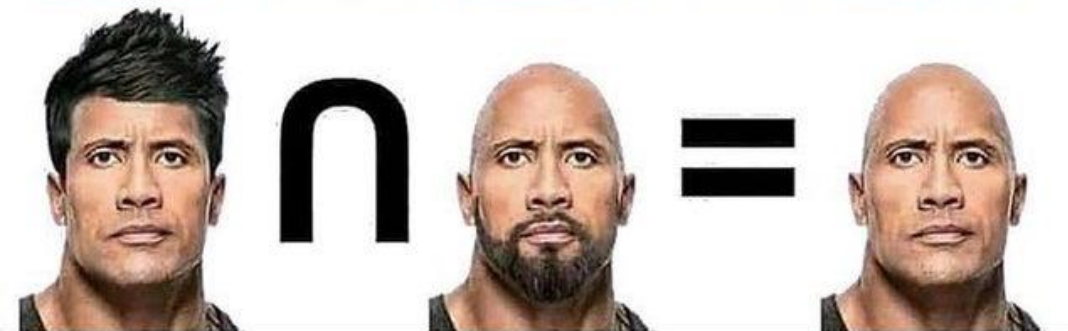
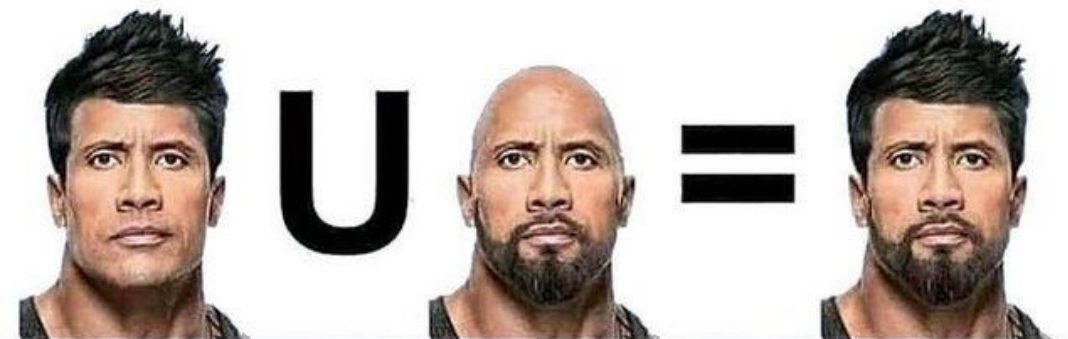
```
public static boolean palindromeStr(String str){
    // הפעולה מקבלת מחרוזת
    // הפעולה מחזירה אמת אם המחרוזת היא פלינדרום ושקר אחרת
}
```



```
public static String withoutDuplicates(String str){  
    // הפעולה מקבלת מחרוזת  
    // הפעולה מחזירה מחרוזת בה אין תווים כפולים  
    String newStr = "";  
    for (int i = 0; i < str.length(); i++)  
        if (newStr.indexOf(str.charAt(i)) < 0) newStr += str.charAt(i);  
    return newStr;  
}
```

```
public static boolean palindromeStr(String str){  
    // הפעולה מקבלת מחרוזת  
    // הפעולה מחזירה אמת אם המחרוזת היא פלינדרום ושקר אחרת  
    for (int i = 0; i < (str.length() - 1) / 2; i++)  
        if (str.charAt(i) != str.charAt(str.length() - 1 - i)) return false;  
    return true;  
}
```

Union Vs Intersection



חיתוך –
שייך לקבוצה א וגם לקבוצה ב

איחוד –
שייך לקבוצה א או לקבוצה ב


```
public static String intersection(String str1, String str2){  
    // הפעולה מקבלת שתי מחרוזות  
    // הפעולה מחזירה את מחרוזת החיתוך  
    String intersectionStr = "";  
    for (int i = 0; i < str1.length(); i++){  
        if (str2.indexOf(str1.charAt(i)) > -1 &&  
            intersectionStr.indexOf(str1.charAt(i)) == -1)  
            intersectionStr += str1.charAt(i);  
    }  
    return intersectionStr;  
}
```



```
public static String union(String str1, String str2){
```

// הפעולה מקבלת שתי רשימות

// הפעולה מחזירה את מחרוזת האיחוד

}

```
public static String union(String str1, String str2){  
    // הפעולה מקבלת שתי רשימות  
    // הפעולה מחזירה את מחרוזת האיחוד  
    String unionStr = "";  
    for (int i = 0; i < str1.length(); i++){  
        if (unionStr.indexOf(str1.charAt(i)) == -1)  
            unionStr += str1.charAt(i);  
    }  
    for (int i = 0; i < str2.length(); i++){  
        if (unionStr.indexOf(str2.charAt(i)) == -1)  
            unionStr += str2.charAt(i);  
    }  
    return unionStr;  
}
```

מעקבים

```
public static String secret(String st, int n) {  
    String result = "";  
    int i = 0;  
  
    while (i < n/2) {  
        result = result + st.charAt(i);  
        result = result + st.charAt(st.length()/2 + i);  
        i++;  
    }  
  
    if (n % 2 != 0)  
        result = result + st.charAt(n - 1);  
    return result;  
}
```

מה יודפס לאחר ביצוע קטע הקוד הבא:

```
String s = "abcabc";
```

```
System.out.println (secret(s, s.length()));
```

```

public static String secret(String st, int n) {
    String result = "";
    int i = 0;
    while (i < n/2) {
        result = result + st.charAt(i);
        result = result + st.charAt(st.length()/2 + i);
        i++;
    }
    if (n % 2 != 0)
        result = result + st.charAt(n - 1);
    return result;
}

```

0	1	2	3	4	5
a	b	c	a	b	c

result	i	i < 3	charAt(i)	charAt(3+i)	n%2!=0

```

public static String secret(String st, int n) {
    String result = "";
    int i = 0;
    while (i < n/2) {
        result = result + st.charAt(i);
        result = result + st.charAt(st.length()/2 + i);
        i++;
    }
    if (n % 2 != 0)
        result = result + st.charAt(n - 1);
    return result;
}

```

0	1	2	3	4	5
a	b	c	a	b	c

result	i	i < 3	charAt(i)	charAt(3+i)	n%2!=0
""	0	true	a	a	
"aa"	1	true	b	b	
"aabb"	2	true	c	c	
"aabbcc"	3	false			
					false

```

public static String secret(String st, int n) {
    String result = "";
    int i = 0;
    while (i < n/2) {
        result = result + st.charAt(i);
        result = result + st.charAt(st.length()/2 + i);
        i++;
    }
    if (n % 2 != 0)
        result = result + st.charAt(n - 1);
    return result;
}

```

0	1	2	3	4
a	a	b	b	a

result	i	i < 2	charAt(i)	charAt(2+i)	n%2!=0
""	0	true	a	b	
"ab"	1	true	a	b	
"abab"	2	false			
"ababa"					true

```
String s = "?????";
```

```
System.out.println (secret(s, s.length()));
```

תנו דוגמה למחרוזות תווים כך שאם נעביר אותן לקטע הקוד, במקום המסומן ב-????,

הפלט לאחר ביצוע קטע הקוד יהיה :

input	output
	fedcba
	abcde

```
String s = "?????";
```

```
System.out.println (secret(s, s.length()));
```

תנו דוגמה למחרוזות תווים כך שאם נעביר אותן לקטע הקוד, במקום המסומן ב-????,

הפלט לאחר ביצוע קטע הקוד יהיה :

input	output
fdbeca	fedcba
acbde	abcde


```

public static String secret(String st, int n) {
    String result = "";
    int i = 0;
    while (i < n/2) {
        result = result + st.charAt(i);
        result = result + st.charAt(st.length()/2 + i);
        i++;
    }
    if (n % 2 != 0)
        result = result + st.charAt(n - 1);
    return result;
}

```

אם נשנה את השיטה secret שכתובה לעיל, כך שבמקום התנאי:

`if (n % 2 != 0)`

יהיה התנאי:

`if (n % 2 == 0)`

מה יודפס לאחר ביצוע קטע הקוד הבא:

`String s = "abcabc";`

`System.out.println (secret(s, s.length()));`

aabbccc

נתונה המחלקה MyString

```
public class MyString {
```

```
    private String _st;
```

```
    public MyString() { this._st = "";}
```

ריקה

```
    public MyString(String s) { this._st = s;}
```

```
    public void removeChar(char ch) {
```

את כל המופעים

```
    public void appendChar(char ch) {
```

בסוף

```
    public void addAtBeginning(char ch) {
```

```
    public char maxChar() {
```

אחרי בא"ב

```
    public boolean isEmpty() { return _st.length() == 0;}
```

```
    public String toString() { return _st; }
```

ריקה

```
    public static MyString secret(MyString ms) {
```

```
    public static MyString secret2(MyString ms) {
```

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

```
}
```

```
public static MyString secret(MyString ms) {
```

```
    MyString str = new MyString();
```

```
    char ch;
```

```
    while (!ms.isEmpty()) {
```

```
        ch = ms.maxChar();
```

```
        ms.removeChar(ch);
```

```
        str.addAtBeginning(ch);
```

```
    }
```

```
    return str;
```

```
}
```

```

public static MyString secret(MyString ms) {
    MyString str = new MyString();
    char ch;
    while (!ms.isEmpty()) {
        ch = ms.maxChar();
        ms.removeChar(ch);
        str.addAtBeginning(ch);
    }
    return str;
}

```

```

MyString ms1 = new MyString("ababab");
System.out.println(secret(ms1));

```

ms	str	!ms.isEmpty()	ch	ms.remove	str.add
"ababab"	"				

```

public static MyString secret(MyString ms) {
    MyString str = new MyString();
    char ch;
    while (!ms.isEmpty()) {
        ch = ms.maxChar();
        ms.removeChar(ch);
        str.addAtBeginning(ch);
    }
    return str;
}

```

```

MyString ms1 = new MyString("ababab");
System.out.println(secret(ms1));

```

ms	str	!ms.isEmpty()	ch	ms.remove	str.add
"ababab"	"	true	b	"aaa"	"b"
"aaa"	"b"	true	a	"	"ab"
"	"ab"	false			

```

public static MyString secret(MyString ms) {
    MyString str = new MyString();
    char ch;
    while (!ms.isEmpty()) {
        ch = ms.maxChar();
        ms.removeChar(ch);
        str.addAtBeginning(ch);
    }
    return str;
}

```

```

MyString ms2 = new MyString("xyzxyyyzyz");
System.out.println(secret(ms2));

```

ms	str	!ms.isEmpty()	ch	ms.remove	str.add
"xyzxyyyzyz"	""	true	z	"xyxyyy"	"z"
"xyxyyy"	"z"	true	y	"xxx"	"yz"
"xxx"	"yz"	true	x	""	"xyz"
""	"xyz"	false			

```
public static MyString secret(MyString ms) {  
    MyString str = new MyString();  
    char ch;  
    while (!ms.isEmpty()) {  
        ch = ms.maxChar();  
        ms.removeChar(ch);  
        str.addAtBeginning(ch);  
    }  
    return str;  
}
```

```
public static MyString secret(MyString ms) {  
    MyString str = new MyString();  
    char ch;  
    while (!ms.isEmpty()) {  
        ch = ms.maxChar();  
        ms.removeChar(ch);  
        str.appendChar(ch);  
    }  
    return str;  
}
```

מה יקרה עם השינוי?

```

public static char fun1(String s) {
    while (s.length() > 1) {
        if(s.charAt(1) < s.charAt(0))
            s = s.substring(1);
        else
            s= s.charAt(0) + s.substring(2);
    }
    return s.charAt(0);
}

```

```

System.out.println(fun1("medicine"));

```

0	1	2	3	4	5	6	7	c1<c0	
m	e	d	i	c	i	n	e	e<m	t
e	d	i	c	i	n	e		d<e	t
d	i	c	i	n	e			i<d	f
d	c	i	n	e				c<d	t
c	i	n	e					i<c	f
c	n	e						n<c	f
c	e							e<c	f
c									

```

public static String fun2(String s, char c) {
    int i = 0;
    while (i < s.length() && s.charAt(i) != c) {
        i++;
    }
    if(i < s.length())
        s = s.substring(0,i) + s.substring(i+1);
    return s;
}

```

```

System.out.println(fun2("medicine", 'i'));

```

0	1	2	3	4	5	6
m	e	d	i	c	i	n

i	i < 8	
0	true	m != i true
1	true	e != i true
2	true	d != i true
3	true	i != i false

0	1	2	3	4	5	6
m	e	d	i	c	i	n

medicine


```

public static String something(String s) {
    String ans = "";
    while(s.length() > 0) {
        char a = fun1(s);
        s = fun2(s, a);
        ans = ans + a;
    }
    return ans;
}

```

```

System.out.println(something("medicine"));

```

s	ans	len>0	a
medicine	""	true	c
mediine	c	true	d
meiine	cd	true	e
miine	cde	true	e
miin	cdee	true	i
min	cdeei	true	i
mn	cdeei	true	m
n	cdeeiim	true	n
""	cdeeiimn	false	

פעולות על מחרוזות

קיימות פעולות המוגדרות במחלקת `String`

את הפעולות **מפעילים על עצם מטיפוס `String`** באמצעות סימן הנקודה (תחביר הפעלת פעולת המחלקה על העצם מהטיפוס).

דוגמה:

```
String str1="have a nice day", str2;  
str2 = str1.substring(5);
```

הפעלת הפעולה `substring` על המחרוזת `str1`. השמת תוצאה ב- `str2`

פעולות עיקריות



equals
compareTo
indexOf
substring
replace
insert/remove

לכל פעולה יש לבדוק - מהם הפרמטרים,
מה הפעולה מבצעת ומה הפעולה מחזירה

equals(String s)

דוגמאות		טיפוס הערך המוחזר	תיאור הפעולה	הפעולה
הערך המוחזר	הפעולה			
false	s1.equals(s2) כאשר ב-s1 נמצאת המחרוזת "love" וב-s2 נמצאת המחרוזת "Love"	בוליאני	פעולה המקבלת מחרוזת, ומחזירה true אם המחרוזת שעליה הופעלה הפעולה והמחרוזת שהתקבלה שוות זו לזו בדיוק. אחרת, היא מחזירה .false	equals(String s)
true	s1.equals(s2) כאשר ב-s1 נמצאת המחרוזת "love" וב-s2 נמצאת המחרוזת "love"			

הפעולה equals

בדיקה האם שתי מחרוזות שוות (כל התווים שווים)
ב java אין לבדוק באמצעות == (ב C# אפשר)
הפעולה מחזירה true או false

```
String s1, s2;  
s1= input.next();  
s2= input.next();  
if (s1.equals(s2))  
    System.out.println (" Strings are equals ");  
else  
    System.out.println (" Strings are not equals ");
```

הפעלת הפעולה על
s1, s2 המחרוזות
פרמטר לפעולה

מחרוזות - השוואה

נניח ונרצה להשוות שתי מחרוזות (לבדוק האם הן שוות).

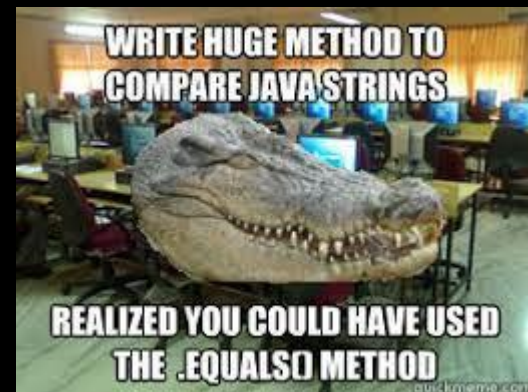
```
public static void main(String[] args) {  
    String s1 = new String("hello");  
    String s2 = new String("hello");  
    System.out.println(s1.equals(s2));  
    System.out.println(s1 == s2);  
}
```

true

false

מה יודפס למסך? למה?

כדי להשוות שתי מחרוזות מבחינת תוכן יש להשתמש בפונקציה `equals()` ולא באופרטור `==` שבודק אם מדובר באותו אובייקט.

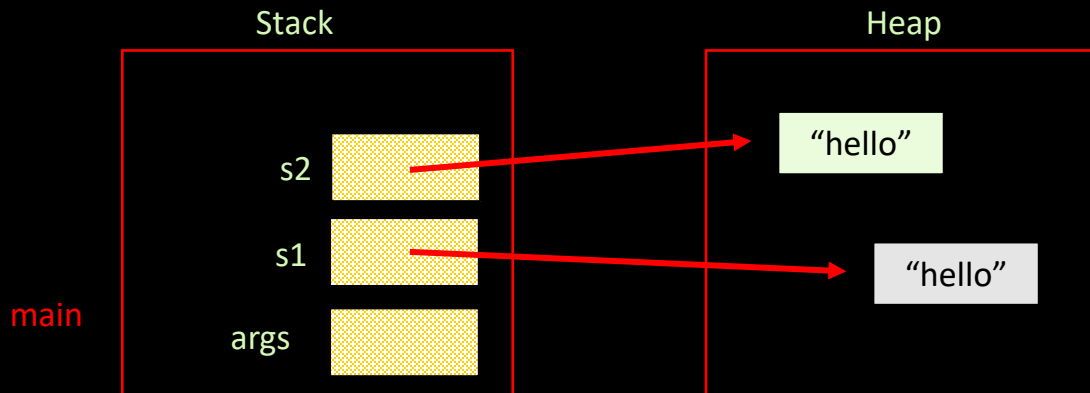


מה יהיה פלט התכנית הבאה?

```
public static void main(String[] args) {  
    String s1 = new String("hello");  
    String s2 = new String("hello");  
    System.out.println(s1.equals(s2));  
    System.out.println(s1 == s2);  
}
```

true

false



```
public class Main {  
    public static void main( String args[] ) {  
        String s1 = "TAT";  
        String s2 = "TAT";  
        String s3 = new String("TAT");  
        String s4 = new String("TAT");  
  
        System.out.println(s1 == s2);  
        System.out.println(s1.equals(s2));  
        System.out.println(s3 == s4);  
        System.out.println(s3.equals(s4));  
        System.out.println(s1 == s3);  
        System.out.println(s1.equals(s3));  
    }  
}
```

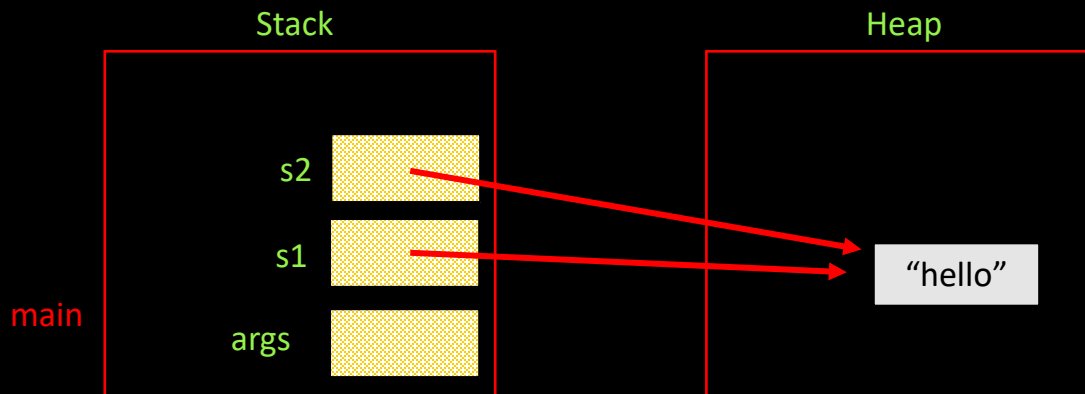
```
true  
true  
false  
true  
false  
true
```

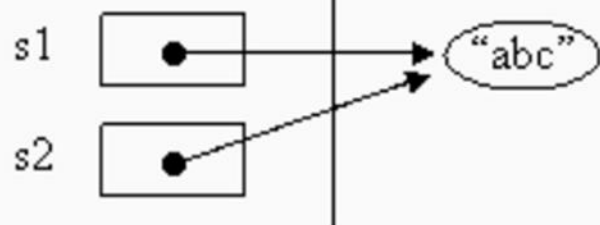

מה יהיה פלט התכנית הבאה?

```
public static void main(String[] args) {  
    String s1 = "hello";  
    String s2 = "hello";  
    System.out.println(s1.equals(s2));  
    System.out.println(s1 == s2);  
}
```

true

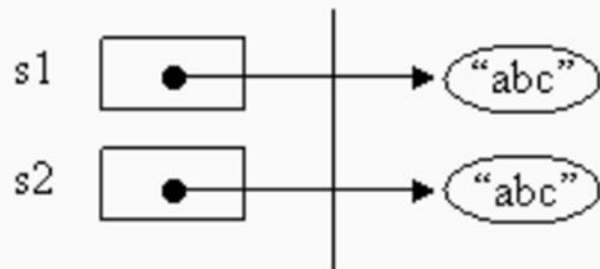
true





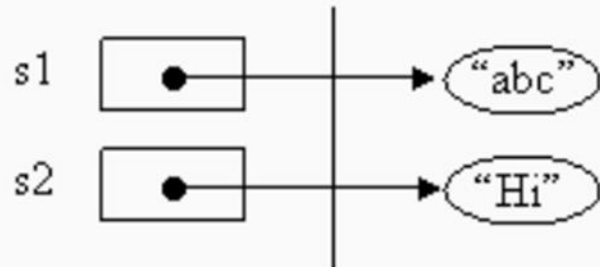
`s1 == s2`

(also implies `s1.equals(s2)` is true)



`s1 != s2`, but

`s1.equals(s2)` is true



`s1 != s2`, and

`s1.equals(s2)` is false

equals()

vs

==

**in Java
String**

מחרוזות – פונקציות בדיקה

Method	Description
<code>equals(str)</code>	whether two strings contain the same characters
<code>equalsIgnoreCase(str)</code>	whether two strings contain the same characters, ignoring upper vs. lower case
<code>startsWith(str)</code>	whether one contains other's characters at start
<code>endsWith(str)</code>	whether one contains other's characters at end
<code>contains(str)</code>	whether the given string is found within this one

compareTo(String s)

דוגמאות		טיפוס הערך המוחזר	תיאור הפעולה	הפעולה
הערך המוחזר	הפעולה			
מספר שלילי כלשהו	<code>s1.compareTo(s2)</code> כאשר ב-s1 נמצאת המחרוזת "aa" וב-s2 נמצאת המחרוזת "ab"	שלם	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.	<code>compareTo(String s)</code>

הפעולה compareTo

השוואה מילונית בין שתי מחרוזות.

הפעולה מופעלת על מחרוזת אחת ומקבלת את המחרוזת השנייה שימוש בפעולה, לדוגמה:

```
num=s1.compareTo(s2);
```

הפעולה מחזירה מספר (!)

אפס - כאשר המחרוזות שוות

מספר שלילי - s1 "קטנה" מ- s2

מספר חיובי - s1 "גדולה" מ- s2

"אבא" לפני "אמא"

"אבא" אחרי "אב"

השוואת מחרוזות

שימו ♥ - ניתן לבדוק האם מחרוזות שוות:

`s1==s2` – הפעולה מחזירה ערך בוליאני (האם אותו אובייקט)

`s1.equals(s2)` – הפעולה מחזירה ערך בוליאני

`s1.compareTo(s2)==0` – הפעולה מחזירה ערך מספרי

שימו ♥ - ניתן לבדוק האם מחרוזות שונות:

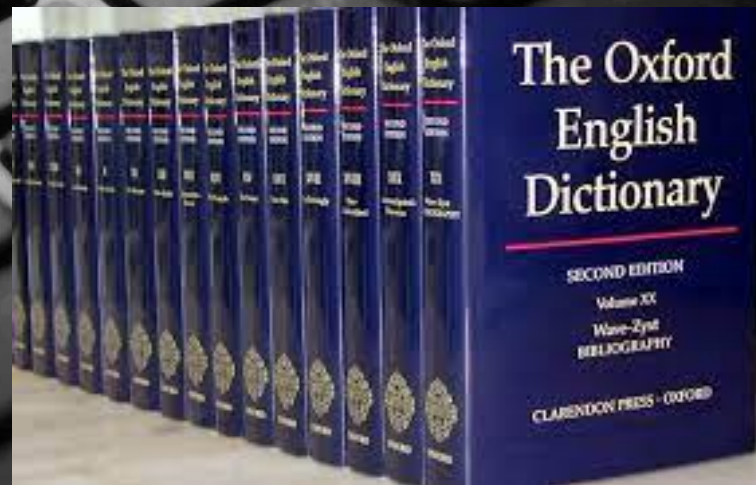
`s1!=s2` – הפעולה מחזירה ערך בוליאני (האם אובייקטים שונים)

`!s1.equals(s2)` – הפעולה מחזירה ערך בוליאני

`s1.compareTo(s2)!=0` – הפעולה מחזירה ערך מספרי

תרגיל

כתבו קטע תוכנית הקולטת 10 שמות מחרוזות
ומדפיסה את המחרוזת שתופיע אחרונה ברשימה
ממוינת אלפביתית של מחרוזות.



```
public static void main(String[] args){  
    String str, strMax;  
    System.out.println("Enter 10 Strings: ");  
    strMax = "";  
    for (int i = 0; i < 10; i++) {  
        str = input.next();  
        if (str.compareTo(strMax) > 0)  
            strMax = str;  
    }  
    System.out.println();  
    System.out.println("The last String is " + strMax);  
}
```

Enter 10 Strings:

banana
apple
cherry
avokado
mango
orange
nectarine
apricot
plum
melon

The last String is plum



indexOf(String s) – indexOf(char c)

דוגמאות		טיפוס הערך המוחזר	תיאור הפעולה	הפעולה
הערך המוחזר	הפעולה			
3	s1.indexOf("pl") כאשר ב-s1 נמצאת המחרוזת "people"	שלם	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.	indexOf(String s)
2	s1.indexOf('o') כאשר ב-s1 נמצאת המחרוזת "people"			indexOf(char c)

הפעולה indexOf

הפעולה מחפשת מיקום של תת-מחרוזת בתוך מחרוזת

הפעולה מחזירה מספר שלם:

המיקום הראשון מתחילה תת-המחרוזת.

אם לא קיימת תת-מחרוזת במחרוזת יוחזר -1

זימון הפעולה:

```
int num=st.indexOf("abc"); → num = 1
```

```
int num=st.indexOf("abc", 3); → num = 5
```

אפשרות נוספת:

חיפוש תת מחרוזת החל ממופע מסוים. יש להעביר 2

פרמטרים: תו/מחרוזת ומיקום התחלת חיפוש.

לחפש את תת-מחרוזת החל מהתו ה-3.

כתבו פעולה המקבלת מחרוזת שהיא סיסמה ומחרוזת של תווים שאסור שיהיו בסיסמה, הפעולה תחזיר true אם המחרוזת לא כוללת אף תו מהתווים האסורים ואחרת תחזיר false

true
false

```
public static void main(String []args){  
    System.out.println(inStr("trt67jh$$", " ~()[]"));  
    System.out.println(inStr("trt67~jh$$", " ~()[]"));  
}  
  
public static boolean inStr(String pass, String notInPass) {  
  
}
```



true
false

```
public static void main(String []args){  
    System.out.println(inStr("trt67jh$$", " ~()[]"));  
    System.out.println(inStr("trt67~jh$$", " ~()[]"));  
}  
  
public static boolean inStr(String pass, String notInPass) {  
    for (int i = 0; i < pass.length(); i++)  
        if (notInPass.indexOf(pass.charAt(i)) > 0)  
            return false;  
    return true;  
}
```



הפעולה lastIndexOf

הפעולה מקבלת מחרוזת ומחזירה את המיקום הראשון שלה בסריקה מסוף המחרוזת עליה היא פועלת.

כמו הפעולה indexOf - הפעולה תחזיר מספר.

דוגמה:

```
String st = "abacacab";
```

```
int num = st.lastIndexOf("ab");    →    num = 6
```

```
int num = st.lastIndexOf("ac");    →    num = 4
```

גם את הפעולה `lastIndexOf` אפשר לזמן גם עם שני פרמטרים. במקרה הזה הסריקה היא מהמיקום המתואר ע"י הפרמטר
דוגמה:

```
String st = "abacacab";
```

```
int num = st.lastIndexOf("ab", 5); → num = 0
```

```
int num = st.lastIndexOf("ac", 3); → num = 2
```

כתבו פעולה המקבלת מחרוזת שהיא כתובת email ומחזירה true אם בכתובת יש בדיוק @ שטרודל (ובעברית: כרוכית) אחד.



שְטְרוּדֵל (בעברית: כְרוּכִית; מגרמנית: (Strudel) הוא סוג של מאפה מרכז-אירופי, שמקורו במטבח האוסטרי, ומכאן כינויו המקובל "שטרודל וינאי". מקור המילה הוא כנראה מגרמנית עתיקה, שבה פירוש המילה "שטרודל" הוא "מערבולת".

הסוגים הנודעים ביותר הם שטרודל תפוחי עץ (, (Apfelstrudel טופפן-שטרודל) טופפן היא סוג של גבינת שמנת עשירה) ושטרודל דובדבנים חמוצים (, (Weichselstrudel יש גם שטרודלים מלוחים המכילים מילוי של תרד, כרוב כבוש וכדומה.


```
public static boolean onlyOneStrudel (String email) {  
    if (email.indexOf('@') < 1) return false; // אין אף אחד או בהתחלה  
    if (email.indexOf('@') != email.lastIndexOf('@'))  
        return false; // יש יותר מאחד  
    return true;  
}
```

```
return (email.indexOf('@') >= 1) &&  
    (email.indexOf('@') != email.lastIndexOf('@'));
```



substring(int k, int s) - substring(int k)

דוגמאות		טיפוס הערך המוחזר	תיאור הפעולה	הפעולה
הערך המוחזר	הפעולה			
המחרוזת החדשה "Bye"	<code>s1.substring(4)</code> כאשר ב-s1 נמצאת המחרוזת "GoodBye"	מחרוזת	פעולה שיוצרת מחרוזת זהה לתת-מחרוזת המתחילה מהמקום ה-k של המחרוזת שעליה הפעולה מופעלת ועד סופה. הפעולה מחזירה את המחרוזת החדשה.	<code>substring(int k)</code>
המחרוזת החדשה "Bye Is"	<code>s1.substring(4,10)</code> כאשר ב-s1 נמצאת המחרוזת "GoodBye Israel"	מחרוזת	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.	<code>substring(int k, int s)</code>

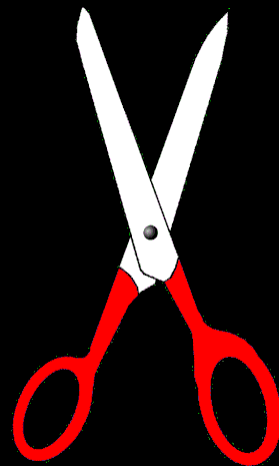
הפעולה substring(int k)

הפעולה מקבלת מספר שלם - מיקום התחלת תת מחרוזת רצויה
הפעולה מחזירה מחרוזת ממיקום זה:

```
String str1 = "good morning";
```

```
String str2 = str1.substring(5); → str2="morning"
```

```
String str3 = str1.substring(9); → str2="ing"
```



הפעולה substring(int k, int s)



הפעולה מקבלת שני מספרים שלמים

המספר הראשון - מיקום התחלה

המספר השני - מיקום סוף (עד, לא כולל)

הפעולה מחזירה מחרוזת חדשה באורך s ממיקום k.

```
String str1 = "good morning";
```

```
String str2 = str1.substring(5,8);
```

```
String str3 = str1.substring(9,3);
```

→ str2="mor"

→ שגיאה, חריגה

תרגיל

כתבו קטע תוכנית המקבל כקלט מחרוזת – אם מספר
התווים במחרוזת אי-זוגי יודפסו שני התווים המרכזיים, אחר
יודפסו שלושת התווים האמצעיים
הנחה: במחרוזת לפחות שני תווים

"hello"
"Sunday"

```
System.out.println("Enter a String ....");  
String line = input.next();  
int x = line.length() ;  
  
if (x % 2 == 0)  
    System.out.println(line.substring(x/2-1, x/2+1));  
else  
    System.out.println(line.substring(x/2-1, x/2+2));
```

כתבו פעולה המקבלת שתי מחרוזות ומחזירה את תת
המחרוזת הזוה הארוכה ביותר שהיא תחילתה של
המחרוזת הראשונה וסופה של המחרוזת האחרת. הפעולה
תדפיס את המחרוזת.

"ab cd fgrt"
"uytccffab cd"

```
public static int startAndEnd(String st1, String st2){  
  
    int len = Math.min(st1.length(), st2.length());  
    boolean found = false;  
    while(!found && len > 0){  
  
        String sub1 = st1.substring(0, len);  
        String sub2 = st2.substring(st2.length() - len);  
  
        if (sub1.equals(sub2)){  
            found = true;  
            System.out.println(sub1);  
        }  
        else  
            len--;  
    }  
  
    return len;  
}
```


כתבו פעולה המקבלת שתי מחרוזות ומחזירה כמה פעמים
מופיעה המחרוזת השנייה במחרוזת הראשונה.

"ab cdffab t ftr ab"
"ab"

```
public static int countSubString(String str, String sub) {  
    int count = 0, i=0;  
    while (i < str.length()-sub.length()) {  
        if (str.substring(i, i+sub.length()).equals(sub)) {  
            count++;  
        }  
        i++;  
    }  
    return count;  
}
```

```
System.out.println(countSubString("aaaabaaabaab","aa"));
```

פעולות נוספות ממחלקת String

המחרוזת החדשה "peace"	<code>s1.toLowerCase()</code> כאשר ב-s1 נמצאת המחרוזת "Peace"	מחרוזת	פעולה שיוצרת מחרוזת זהה למחרוזת שעליה היא מופעלת, ובה כל האותיות מוחלפות באותיות קטנות. הפעולה מחזירה את המחרוזת החדשה.	<code>toLowerCase()</code>
המחרוזת החדשה "PEACE"	<code>s1.toUpperCase()</code> כאשר ב-s1 נמצאת המחרוזת "Peace"	מחרוזת	פעולה שיוצרת מחרוזת זהה למחרוזת שעליה היא מופעלת, ובה כל האותיות מוחלפות באותיות גדולות. הפעולה מחזירה את המחרוזת החדשה.	<code>toUpperCase()</code>

המרת מחרוזת למספר

```
String s1 = "123";  
int n1 = Integer.valueOf(s1) + 1;  
System.out.println(n1);  
int n2 = Integer.parseInt(s1) - 1;  
System.out.println(n2);  
  
String s2 = "123.124";  
double n3 = Double.valueOf(s2) + 1;  
System.out.println(n3);  
double n4 = Double.parseDouble(s2) - 1;  
System.out.println(n4);
```

124
122
124.124
122.124

תזכורת char

```
char c = 'a';
```

```
if ('a' < 'b').....;
```

```
char c1 = c++;
```

```
char c2 = (char)(c1+2);
```

```
if (c >='a' && c <='z')...
```

פעולות רלבנטיות ממחלקת Character

```
boolean b = Character.isLetter(c);           // true if letter
b = Character.isDigit(c);                     // true if digit
b = Character.isLetterOrDigit(c);             // true if letter or digit
b = Character.isWhitespace(c);                // true if white space
b = Character.isLowerCase(c);                 // true if lowercase
b = Character.isUpperCase(c);                 // true if uppercase
char c1 = Character.toLowerCase(c);           // equivalent lowercase
c1 = Character.toUpperCase(c);                 // equivalent uppercase
```