מבוא לחישוב סיכומי שיעורים

שיעור 2 - 17/10/21

בcmd אם נרשום LS זה יראה לנו את כל התיקיות בשולחן עבודה,

בשביל לפתוח תוכנית מסויימת ניתן להעתיק את הpath שלה דרך המאפיינים ולהדביק בcmd

cd.. זה לחזור אחורה.

בשביל להפעיל תוכנית ג'אווה נרשום: javac Main.java (javac=java compailer) ברגע שנלחץ אישור אם לא הופיע שגיאה סימן שקובץ תקין ויופיע לנו בשולחן עבודה קובץ מקומפל כך שאם אחזור לcmd וארשום את שם התוכנית ("java Main") – יודפס לי הפלט של התוכנית.

טיפוסים פרימיטיביים:

Char – תו (כל תו), דורש 2 ביטים

Int – מספר שלם (חיובי או שלילי), דורש 4 ביטים, טווח ערכים: של -231 ועד 231-1

Double – מספר ממשי/עשרוני, דורש 8 ביטים, טווח ערכים: -1.7\*10308 ועד 1.7\*10308.

Boolean – מכיל true/false, דורש ביט 1.

משתנים מזהים:

משמשים כדי לתת שמות למשתנים, יכול להכיל אותיות, מספרים, קו תחתון וסימן דולר, אך לא יכול להכיל סימונים מתמטיים או להתחיל במספר.

הערה – אם נרשום את אותו שם למשתנה בהבדל של אותיות גדולות/קטנות – אלו יהיו משתנים שונים.

פעולות שאפשר לעשות על משתנים בוליאנים: not(!), or(||), and(&&).

\t הזזת טאב

\\ הדפסת \

'\ הדפסת גרש (שהתוכנה לא תחשוב שזה חלק ממחרוזת

\n ירידה לשורה חדשה

Ctrl+shift+f מיישר את השורות שיהיו קריאות יותר והזחות מתאימות.

קליטת ערכים:

כדי לקלוט ערכים יש לייבא קודם את הספרייה scanner ע"י: **import** java.util.Scanner;

ולאחר מכן לרשום: Scanner sc= **new** scanner(System.***in***);

כעת ניתן יהיה לקלוט כאשר המשתמש מפעיל את התוכנית ע"י שהוא יכתוב: x=sc.nextInt()

מה שזה בעצם אומר: קח את הקלט מהמשתמש sc, תמיר אותו לint ותיישם אותו בx.

וניתן לקלוט ישירות לתוך משתנה שהקמנו: **double** y = sc.nextDouble();

קבוע

סוג של משתנה שאני לא מעוניין שיהיה ניתן לשינוי (למשל אחוז המע"מ וכד'), איך נרשום?

**final** **int** TAX=(117/100); את המשתנה קבוע נהוג לכתוב באותיות גדולות.

ספריית math

כדי 'לחסוך' לנו פעולות חישוביות ניתן להשתמש בספרייה המובנית math (לא צריך לייבא אותה קודם).

כדי להשתמש בפעולות שלה נרשום Math.\_\_ ניתן לבצע פאי, קוסינוס, סינוס, המרת זווית ממעלות לרדיאן או הפוך (toradian)

למשל אם אנו רוצים לפתור משוואה ריבועית נרשום:

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

**double** a,b,c,d;

Scanner sc=**new** Scanner(System.***in***);

System.***out***.println("enter a:");

a=sc.nextDouble();

System.***out***.println("enter b:");

b=sc.nextDouble();

System.***out***.println("enter c:");

c=sc.nextDouble();

System.***out***.println(a+"x^2+2"+b+"x+"+c+"=0");

d=Math.*pow*(b, 2)-4\*a\*c; //pow = חזקה - power

**double** x1,x2;

x1=(-b+Math.*sqrt*(d))/(2\*a); //sqrt = שורש

x2=(-b-Math.*sqrt*(d))/(2\*a);

System.***out***.println("x1= "+ x1+ " x2 = "+x2);

המרה של משתנים מסוג מסוים לסוג אחר נבצע ע"י:

Answer=(str)num – ממיר מספר למחרוזת

תירגול 2 – 18.10.21

כדי לעשות חזקה צריך לעשות על משתנה double, או לחילופין להמיר לint: d=(int)Math.*pow*(b, 2)

לעגל תוצאה ניתן ע"י Math.ceil(9/2.) – מעגל למעלה, Math.round(9/2.) מעגל למטה.   
**דגש חשוב**- להוסיף את הנקודה, ללא הנקודה הפקודה 9/2 תיתן לנו 4 כי שניהם int אז יחזיר מספר שלם.

**הערות** – תמיד מומלץ להוסיף הערות כדי שדברים יהיו ברורים גם למי שירצה לעבוד על התוכנית שלנו אחר כך. הערות נסמן ב// - והתוכנית לא תיקרא את השורה הזו. אם אני רוצה לכתוב כמה פסקאות אפשר לכתוב בהתחלה \*/ ובסוף /\*, כל מה שיהיה באמצע לא ייקרא.

פונקציות מובנות על משתנים

נרשום באות גדולה ונקודה, למשל – מה הערך המקסימלי שמשתנה מסוג זה יכול לקבל:

**int** n1=Integer.***MAX\_VALUE***;

**short** n2 = Short.***MAX\_VALUE***;

**long** n3 = Long.***MAX\_VALUE***;

**double** n4 = Double.***MAX\_VALUE***.

**float** n5 = Float.***MAX\_VALUE***;

קיצור: במקום לרשום למשל a=a+1 ניתן לרשום a++, אותו דבר זה עם מינוס  
ההבדל בין זה לבין ++a, זה הקדימות בפעולות למשל בהדפסה.

System.***out***.println(a++); - תדפיס לי את a ולאחר מכן תעלה אותו ב1.

System.***out***.println(++a); - תעלה לי את a ב1 ואז תדפיס לי אותו.

לולאת do while ייכתב כפקודת do קודם, גם אם התנאי לא מתקיים, ואז נגיד לקוד להיכנס ללולאה

שיעור 3 – 24/10/2021

למדנו על או / וגם ע"י סימון && וגם, || או

כאשר הקוד רץ על שאילתת וגם – אם התנאי הראשון לא נכון, הוא כבר יפסיק ולא ירוץ על התנאי השני.

כנ"ל הפוך עם או – אם התנאי הראשון נכון – הוא יפסיק ולא ירוץ על התנאי השני.

אם אנו רוצים שהקוד בכל זאת ירוץ על התנאי השני גם אם הראשון כבר 'אומר' לי מה תהיה התוצאה נכתוב סימן אחד, במקום && נכתוב &, ובמקום || נכתוב |.

תנאי if מקוצר:

(x>0)?-x:x

לאחר התנאי נרשום סימן שאלה, ואז מה אם כן, נקודתיים, ואז מה אם לא.

נוכל להשתמש בזה גם בתוך הדפסה:

Int age = MyConsole.readInt("age:");

System.out.println( (age>18)?"you can enter the party" : "you can't enter the party")

שיעור 4 – 31/10/2021

מערך – כלי שבאמצעותו ניתן לאחסן נתונים, למעשה אנו מגדירים מערך ואומרים לתוכנית: לך לזיכרון ו"תתפוס מקום" בזיכרון כמות מקום לפי כמות המקומות שאגיד לך. (כל האיברים במערך צריכים להיות מאותו משתנה). הגדרת המערך: **int** [] arr = **new** **int** [9]; יצירת מערך בגודל 9 איברים

ברירת המחדל היא לשים בכולם 0, כעת נידרש לאתחל את המשתנים בו לאיברים שנרצה.

כדי לגשת/ לשנות איבר ספציפי במערך נרשום: arr[2]=8;

אנו נשתמש במערך כאשר יש לנו מספר משתנים שכל אחד מקבל ערך משל עצמו.

למשל: לכתוב תוכנית המקבלת רשימת ציונים, ומציגה מה הממוצע וכמה סטודנטים קיבלו מעל הממוצע:

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

**int** n=MyConsole.*readInt*("enter a number");

**int** [] arr = **new** **int** [n];

**int** sum=0;

**for** (**int** i=1;i<=n;i++) {

arr[i-1]=MyConsole.*readInt*("enter mark "+i);

sum+=arr[i-1];

}

**double** avg=sum/(n\*1.0); //\*1.0 to because n is int

**int** c=0;

**for**(**int** i=0; i<n;i++) {

**if**(arr[i]>avg)

c++;

}

System.***out***.println("the average is: "+avg+" there is "+c+" student above the avg");

שיעור 5 – 07/11/2021

אנו יכולים להגדיר מערך באופן ידני: **int** [] arr= {2,4,6,8,10,12,14,16,18,20};

דגש – אין חיתוך מחרוזות בג'אווה – אני לא יכול לגשת לאיבר האחרון למשל ע"י: arr[-1]

אם אני אצור מערך חדש וארצה שיופיעו בו אותם ערכים כמו בarr אסור לי לכתוב **int** [] b = arr;

כי אז הם יצביעו על אותו מקום בזיכרון ואם אחד ישתנה אז גם השני ישתנה.

מה שצריך לעשות זה לעשות העתקה מורחבת:

for (int i=0; i<arr.length;i++)

b[i]=arr[i];

ניתן למצוא סכום של מערך ע"י שימוש בלולאת for בפורמט אחר ממה שאנו רגילים:

**int** [] arr= {2,4,6,8,10,12,14,16,18,20};

**int** sum=0;

**for**(**int** x:arr)

sum+=x;

תוכנית הבודקת אם מספר הוא פולינדרום:

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

**int** n=MyConsole.readInt("n:");

**int** [] arr=**new** **int**[n];

**for**(**int** i=0;i<n;i++){

arr[i]=MyConsole.readInt("arr["+i+"]: ");

}

**boolean** ans=**true**;

**for**(**int** i=0;i<=n/2;i++){

**if**(arr[i]!=arr[n-1-i]){

ans=**false**;

**break**;

}

}

System.out.println("The array is "+((ans?"palindrome.":"not palindrome.")));

**מערך דו ממדי**

מערך של מערכים – כל איבר בפני עצמו הוא מערך המכיל איברים.

הגדרה: ניתן להגדיר באופן ידני:

**int** arr [][]= {

{0,1,2},

{4,5,6},

{7,8,9}};

אבל ההגדרה המקובלת היא באופן הבא:

**int** [] [] school = **new** **int** [n][m];

תרגיל לדוגמא: תוכנית המקבלת למערך דו ממדי את הציון של תלמיד בכל כיתה ומחשבת את הממוצע של כל כיתה ואת הכיתה המצטיינת

**int** n=MyConsole.*readInt*("n:"); //classes index

**int** m=MyConsole.*readInt*("m:"); //students index

**int** [] [] school = **new** **int** [n][m];

**double** [] avgs=**new** **double**[n];

**for**(**int** i = 0;i<n;i++) {

**for**(**int** j = 0; j<m; j++) {

school[n][m]=MyConsole.*readInt*("enter student number "+(m+1)+"in class "+(n+1));

avgs[i]+=school[i][m];

}

}

**int** MaxIndex=0;

**for** (**int** k=0; k<n; k++) {

avgs[k]/=(m\*1.0);

**if**(avgs[k]>avgs[MaxIndex]) {

MaxIndex=k;

}

}

System.***out***.println("best class is:"+school[MaxIndex+1]+" it has an avearge of: "+avgs[MaxIndex]);

**תירגול מערך דו ממדי 08/11/2021**

תוכנית המדפיסה את הסכום של השורה וה

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

**int** arr[][]=**new** **int** [5][5];

**for**(**int** i=0;i<arr.length;i++) {

**for**(**int** j=0; j<arr.length; j++) {

arr[i][j]= i+j;

}//System.out.println(arr); //[[I@515f550a אם נרשום רק את המערך בסוגריים נקבל: }

**for**(**int** i=0;i< arr.length;i++) {

System.***out***.println(Arrays.*toString*(arr[i])); }

כדי להציג את המערך השתמשנו בפונקציה הממירה את המספר למחרוזת:

(Arrays.*toString*(arr[i]))

שיעור 6 – 14/11/2021

**גישה לאיבר ספציפי בתוך מערך**:

**int** arr2d[][]=**new** **int** [4][2];

arr2[1][0]=8

השמה בתוך מערך בגודל 4 על 2, באיבר בשורה 2 באיבר הראשון את המספר 8.

**הדפסת האיברים במערך** תתבצע ע"י 2 לולאות for:

**for** (**int** row=0;row<arr2d.length;row++) {

**for** (**int** col=0;col<arr2d[row].length ;col++) {

System.***out***.println(arr2d[row][col]);

**מערך מרופט** – מערך דו ממדי שלא בהכרח כל המערכים שבתוכו הם אותו גודל. למשל :

**int** [][] arr = {{2,3,4},{1,0},{7,5,6,8}}

|  |  |  |
| --- | --- | --- |
| 2 | 3 | 4 |
| 1 | 0 |
| 7 | 5 | 6 | 8 |

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

**int** [][] arr =**new** **int** [5][]; // number of columns is indefinite

**for** (**int** cls=0; cls<5; cls++) {

**int** current\_class=MyConsole.*readInt*("enter number of students in class"+(cls+1));

arr[cls] = **new** **int** [current\_class];

}**for** (**int** [] a :arr) {

System.***out***.println(Arrays.*toString*(a));

**מערך רב ממדי**

int [][][]arr3d = new int [numbers of 2d arrays][rows][columns]

האיבר הראשון הוא כמות המערכים הדו ממדיים – כלומר אם נדמיין קובייה – זה בעצם העומק של הקובייה. ואז 2 הסוגריים המרובעים הבאים זה כמו מערך דו ממדי של שורות ועמודות.

יצירת מערך תלת מימדי והדפסתו:

**int** [][][]arr3d = **new** **int** [3][4][5];

arr3d[1][3][4]=65;

**for** (**int** [][]mat:arr3d) {

**for** (**int** []a: mat) {

System.***out***.println(Arrays.*toString*(a));

}

System.***out***.println("====");

**תו - char**

כאשר אני רוצה להתייחס לתו מסוים ולדעת מאיזה סוג הוא – מספר/אות/ סימן יחס וכו' ישנה טבלה שנקראת טבלת הסקי. בטבלה זו מסודרים כל 128 התווים שיש לנו על המקלדת למשל כל המספרים 0 עד 9 מסודרים בטבלה מהערך ה48 ועד 57. כך שלמשל אם אני רוצה לבדוק אם הערך הוא אחד מהמספרים אפשר לעשות את המספר של התו בטבלת הסקי פחות 48 שזה הערך המינימלי של סדרת סוג התווים שרציתי - אם קיבלתי מספר בין 0 ל9 זה אומר שקיבלנו מספר.

כמו כן ניתן לעשות char+1 וזה ידפיס לנו את התו אחד יותר לפי הטבלה! כך שלמשל המספר 9 הוא באינדקס 57 בטבלה, אם נוסיף לו 1 נקבל את הערך נקודתיים : שהוא הערך ה58.

הערה – אותיות גדולות ואותיות קטנות יש להם ערכים שונים בטבלה.

**מחרוזת - string**

המחרוזת היא למעשה מערך של תווים שזה למעשה טקסט

char[]st= {'h','e','l','l','o',' ','w','o','r','l','d'};

כך שכאשר אנו שומרים מחרוזת אנו למעשה שומרים מערך של תווים בזיכרון.

String st= "hello world";

(חשוב לשים לב לשים אות גדולה)

ואם אני רוצה לגשת לתו מסוים בתוך המחרוזת:

System.out.println(st.charAt(6));

כל מחרוזת שניתן לחשב את הערך שלה בזמן אמת

Heap –

יש 2 דרכים להגדיר מחרוזת – כך שאם נשווה ביניהם נקבל false! כי המשתנים מצביעים על מקום אחר בזיכרון.

String st0= "hello world";

String st1= **new** String ("hello world");

String st2= "hello world";

System.***out***.println(st0==st1);

**פעולות על מחרוזות:**

1. בדיקה האם הערך של המחרוזת בתוך המשתנים שווה, גם אם סוג הגדרת המחרוזת היא שונה נכתוב:

System.out.println(st0.equals(st1));

1. הדפסת חלק מהמחרוזת – למשל מהאיבר ה6 כולל עד ה8 לא כולל

System.out.println(st0.substring(6,8));

1. הדפסת חלק מהמחרוזת – למשל מהאיבר ה8 כולל עד הסוף

System.out.println(st0.substring(8));

1. האם המחרוזת מכילה את התווים בסוגריים – יחזיר ערך בוליאני

System.out.println(st0.contains("wor"));

1. האם המחרוזת מתחילה במילה בסוגריים – יחזיר ערך בוליאני

System.out.println(st0.startsWith("world"));

1. מה האינדקס של הפעם הראשונה שמופיע מה שבסוגריים

System.out.println(st0.indexOf("o"));

1. האינדקס של הפעם הראשונה שמופיע מה שבסוגריים החל ממקום מסוים

System.out.println(st0.indexOf("o",7));

1. שירשור מחרוזות

st0.concat(st2);

st0.concat("\n end of string");));

1. הפרש ערכי טבלת הסקי

System.out.println("abcd".compareTo("efg"));

1. יצירת מערך של מחרוזות

String [] st = {"hello", "my name", "is", "ohad"};

1. חיתוך מחרוזת לפי תו מסוים והשמתו בתוך מערך – למשל בכל מקום שיש רווח

String [] st = st0.split(" ");

System.out.println(Arrays.toString(st));

1. החלפת מילה/ תו מסוים באחר – הראשון יהיה המוחלף והשני במה להחליף

st0.replace('p','b')

קוד הבודק כמה פעמים מופיעה במילה האות c והאות d:

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

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("please enter a string");

String st1= sc.next();

**int** d=0,c=0;

**for** (**int** i=0;i<st1.length();i++) {

**if** (st1.charAt(i)=='d')

d++;

**else** **if** (st1.charAt(i)=='c')

c++;

}

System.***out***.println("there is "+c+" times c, and "+d+" times d");

במידה והיינו רוצים שהפונ' תחזיר כמה פעמים מופיעה כל אות:

System.***out***.println("please enter a string");

String st1= sc.next();

**int** arr[]=**new** **int** ['z'-'a'+1];

**for** (**int** i=0;i<st1.length();i++) {

arr[st1.charAt(i)-'a']++;

}

**for** (**int** j=0; j<arr.length ;j++) {

**if** (arr[j]>0)

System.***out***.println("there is "+arr[j]+" times "+(**char**)((**int**)'a'+j));

}

החלפת האות הראשונה באות גדולה:

String first = str.substring(0,1);

First = first.toUpperCase();

String Word = first + str.substring(1);

System.out.println(Word);

**פונקציות**

המרכיבים של פונקציה:

1. הגדרת הפרטיות שלה – public ניתן לגשת אליו מחוץ לאותה פונקציה, private – לא ניתן לגשת אליו מחוץ לפונקציה אלא רק מתוך הפונקציה אל עצמה.
2. Static נלמד בהמשך – בגדול לא נרשום אותם בקודים של מחלקה – כי אז לא יהיה אפשר להגיע לשמות עצם.
3. מה הפונקציה מחזירה – int / int [ ] / double / boolean וכו'
4. השם של הפונקציה
5. מה המשתנים שהפונקציה מקבלת, למשל 2 משתנים מסוג int נרשום: (int n, int m)
6. לאחר מכן נרשום את הפונקציה עצמה והחישובים שלה
7. בסוף נרשום return מה הפונקציה מחזירה.

**כמה דגשים חשובים –**

1. אפשר להגדיר 2 פונקציות **עם אותו שם** (!) אבל אחד מקבל int ואחד מקבל double המערכת תדע לבד לבחור לאיזו מהן לגשת אם הכנסת מספר שלם או עשרוני למשל.
2. פונקציה המחזירה void לא צריך לעשות return.
3. ברגע שפונקציה הגיעה כבר לפקודת return היא תצא מהפונקציה גם אם על פניו היא הייתה יכולה לקיים גם תנאי אחר ולהחזיר return אחר.

דוגמא לפונקציה:

פונקציה המקבלת מספר ובודקת אם הוא ראשוני ומחזירה true/false:

**public** **static** **boolean** IsPrime (**int** num) {

**if** (num<2)

**return** **false**;

**else** **if**(num==2)

**return** **true**;

**else** {

**for** (**int** i=2;i<Math.*sqrt*(num);i++) {

**if** (num %i==0)

**return** **false**;

}

}**return** **true**;

}

לאחר מכן בmain נקרא לפונקציה עם הערך שאכניס לו:

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

System.***out***.println(*IsPrime*(15));

System.***out***.println(*IsPrime*(23));

דוגמא נוספת:

**public** **static** **double** CircleArea(**int** radius) {

**double** answer = (Math.***PI***\*radius\*radius);

**return** answer;

}

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

System.***out***.println(*CircleArea*(3));

}

מתי נגדיר בפונקציה void? כאשר היא עושה פעולה מסוימת, למשל הדפסה, ולא מחזירה משהו.

פונקציה המקבלת מחרוזת טקסט ומזירה את ההיפוך שלו – בהתחלה הופכת את הסדר של המילים ואז את הסדר של האותיות בתוך כל מילה

public static String sentence (String s) {

String [] sentence= s.split(" ");

String [] new\_sentence= new String[sentence.length];

int counter=0;

for (int i=sentence.length-1; i>=0; i--) {

new\_sentence[counter]=sentence[i];

counter ++;

sentence[i]="";

}

String temp="";

counter=0;

for (int i=0; i<new\_sentence.length; i++) {

temp=new\_sentence[i];

for(int j=temp.length()-1; j>=0; j--) {

sentence[i]+=(temp.charAt(j));

}

}

String answer ="";

for (int k=0; k<sentence.length; k++) {

answer= answer+ sentence[k]+ " ";

}

answer = answer.substring(0, answer.length()-1);

return answer;

}

public static void main(String[] args) {

String m= "tnaw i revetahw etirw nac i";

System.***out***.println(*sentence*(m));

}

**פונקציה רקורסיבית**

פונקציה שלמעשה קוראת לעצמה שוב עד תנאי מסוים– **חייב** להגדיר תנאי עצירה אחרת הפונ' לא תעצור!

דוגמא: פונ' שמקבלת מספר ומדפיסה את כל המספרים מהמספר עד 0.

**public** **static** **void** rec\_for(**int** n) {

**if** (n<0)

**return**;

System.***out***.println(n);

*rec\_for*(n-1);}

דוגמא נוספת: פונקציה המחשבת מספר פיבונאצ'י

**public** **static** **int** fib\_rec(**int** n) {

**if** (n==0 || n==1)

**return** 1;

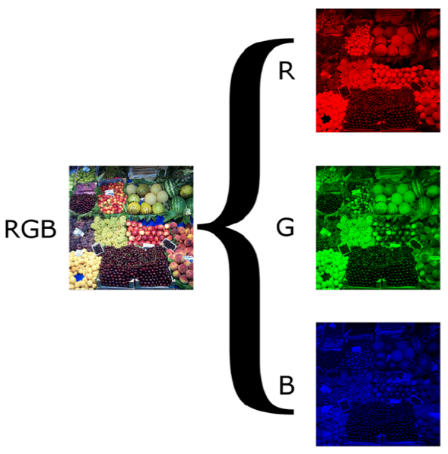
**return** *fib\_rec*(n-1)+*fib\_rec*(n-2);

}

מתי נשתמש ברקורסיה? כאשר לא בהכרח אני יודע כמה פעמים הפונקציה צריכה לרוץ

**RGB – פיקסלי צבע**

כל תמונה צבעונית היא אוסף של פיקסלים, כך שלכל תא פיקסל אחד יש צבע המיוצג במחשב ע"י שילוב של 3 עוצמות של שילוב אדום ירוק וכחול. העוצמות נעות בין 0 – ללא עוצמה, ו255 עוצמה מלאה.

את הצבע הלבן מייצג ע"י (255,255,255) ואת הצבע השחור מייצג ע"י (0,0,0).  
לכל תא ותא יהיה את הערך (R,G,B) שלה. ניתן לייצג תמונה באורך H וברוחב W ע"י 3 מטריצות של ערכי RGB כאשר כל אחת הינה בגודל H\*W.

יצירת תמונה תהיה במערך תלת מימדי: img [3] [h] [w] – תמיד נרשום בראשון 3 – עבור ערכי R,G,B. ואז נתייחס לזה כאל מערך דו ממדי של גובה התמונה כפול הרוחב. נכניס לכל מימד את הערך של הצבע שלו כך שכאשר נשים את המטריצות אחת על השנייה יתקבל הצבע של התמונה באיחוד המימדים.

למשל כדי לייבא תמונה מסויימת נייבא אותה ע"י רישום הנתיב להגיע אליה ולבסוף השם שלה, למשל עבור תמונה בשם cat.jpg:

**int** [][][] image= MyImageIO.*readImageFromFile("C://Documents and Settings/user1/Desktop/cat.jpg");*

הוא למעשה לקח תמונה והפך אותה למטריצה החדשה שהגדרנו.

ניתן לעשות גם את הפעולה ההפוכה: לקחת מטריצה ולהפוך אותה לתמונה ע"י השורה:

*MyImageIO.writeImageToFile("catCopy",image);*

במצב זה שלא הוספנו נתיב הוא ישמור את התמונה בתוך הפרוייקט, ניתן להוסיף נתיב להיכן מעוניינים לשמור.

כדי לרוץ על המערך נדרש לעשות 3 לולאות פור אחד בתוך השנייה כך שהראשון למעשה רץ 3 פעמים – על כל אחת מהמטריצות של כל צבע בנפרד.

כאשר נרצה את הגודל של המערך הראשון נרשום arr [0] [ ] [ ].length – יביא לי את מספר השורות שיש במטריצה הראשונה.  
ואם נרשום arr [0] [0] [ ].length – יביא לי את מספר האיברים במטריצה הראשונה בשורה הראשונה.

מעבר על כל איברי המערך התלת מימדי:

**for** (**int** i=0; i<img.length;i++) {

**for**(**int** j=0; j<img[i].length ;j++) {//img[i].length הראשונה במטריצה השורות מספר אל החל

**for**(**int** k=0; k<img[i][j].length;k++) { //ניגש לאיברים החל מהשורה הראשונה במטריצה הראשונה

ניתן למשל לעבור על כל איברי המערך ולהפוך את התמונה לגוון אדום למשל ע"י שינוי כל התאים במטריצה הראשונה בלבד ל255

**public** **class** shiur3 {

**public** **static** **int** [][][] test (**int** img[][][]){

**for** (**int** i=0; i<img[0].length;i++) {

**for** (**int** j=0; j<img[0][i].length ;j++) {

img[0][i][j]=255;

}

}

**return** img;

}

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

**int** [][][] image= MyImageIO.*readImageFromFile*("cat.jpg");

image = *test*(image);

MyImageIO.*writeImageToFile*("CatCopy", image);.

מה שיציג יהיה:



**שיעור 8 – 28/11/2021**

כאשר אני ניגש לכתוב פונ' רקורסיבית יש כמה קווי מחשבה שצריך לחשוב לפני:

1. מה הקלט שעבורו אני לא צריך לחשוב בכלל?
2. האם הוספתי תנאי לעצירה? (לרוב יהיה נוח פשוט ללכת אחורה מהקלט הנתון עד ל0)
3. אופן החשיבה היא "אני לא יודע לפתור לפי הקלט הזה אבל אני כן יודע לאחד לפניו" ואז נרשום פונקציה ל'שלב' אחד קודם לכן ונוסיף מהו את השלב הבא. למשל – mn - אנסה לייצר פונקציה שמחשבת mn-1\*m

דוגמא לאופן החשיבה הנ"ל:

**public** **static** **int** pow(**int** a,**int** b) {

**if** (b==0)

**return** 1;

**return** *pow*(a,b-1)\*a;

דוגמא נוספת – מציאת שארית של מספר (ללא שימוש ב%):

**public** **static** **int** reminder(**int** a,**int** b) {

**if** (a<b)

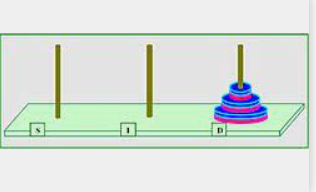
**return** a;

**return** *reminder*(a-b,b);

אם יש שארית 0 – האיטרציה האחת לפני אחרונה תשאיר לי את a כ0 ואז באיטרציה הבאה יוריד עוד b והוא יותר קטן מ0 לכן ידפיס את הa שהוא 0.

**דגש** – לא תמיד עדיף להשתמש בפונקציה רקורסיבית כי היא פותחת המון תאים בזיכרון וממשיכה להחזיק אותם עד סוף הריצה של הפונקציה כדי להשתמש בהן אחרי.

מגדלי האנוי:



**public** **static** **void** hanoi(**int** n, **char** src, **char** help, **char** dest) {

**if** (n==0)

**return**;

*hanoi*(n-1,src,dest,help);

System.***out***.println("move disc number "+n+" from "+src+" to "+dest);

*hanoi*(n-1,help,src,dest);

}

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

*hanoi*(3,'a','b','c');

פונקציה המחשבת סכום מספרים בתוך מערך לפי מספר איברים רצוי:

**public** **static** **int** sumArr(**int** []arr, **int** n) {

**if** (n==1)

**return** arr[0];

**return** *sumArr*(arr,n-1)+arr[n-1];

}

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

**int** [] arr = {2,3,4,5,6};

System.***out***.println(*sumArr*(arr,2));

בפונקציה זו המשתמש נדרש להכניס גם את המערך וגם את כמות האיברים שהוא רוצה לסכום, אבל יכול להיות שאני לא רוצה שהמשתמש יידרש להכניס לי את כמות האיברים שרוצה. במצב זה אפשר לבנות פונקציית **'עזר למעטפת'** שהיא קוראת לפונקציית מעטפת שבנינו למעלה ע"י קריאה אפילו באותו שם אלא שהקלט שונה:

**public** **static** **int** sumArr(**int** []arr) {

**return** *sumArr*(arr,arr.length);

בדיקת כמות פעמים שאות מסויימת נמצאת במחרוזת:

**public** **static** **int** occ(String st, **char** ch) {

**if**(st.length()==0)

**return** 0;

**int** num = 0;

**if** (st.charAt(0)==ch)

num=1;

**return** *occ*(st.substring(1), ch)+num;

זה אופן חשיבה שצריך להתרגל אליו אבל הוא חוסך לפעמים המון שורות קוד.

**תת מערך**

כתוב תוכנית המקבלת מערך ומספר, ומוצאת האם ניתן להגיע למספר הזה ע"י סכום של חלק מהאיברים במערך.

**public** **static** **boolean** subsetSum(**double** arr[], **int** size, **double** sum){

**if** (sum == 0)

**return** **true**;

**if** (sum < 0 || size == 0)

**return** **false**;

**return** *subsetSum*(arr, size - 1, sum) || *subsetSum*(arr, size - 1, sum - arr[size - 1]); }

לפונקציה זו ניתן להוסיף פונקציית עזר למעטפת:

**public** **static** **boolean** subsetSum(**double** arr[], **double** sum) {

**return** *subsetSum*(arr, arr.length, sum);

**שיעור 9 – 05/12/2021**

**מיונים**

כאשר יש לנו רשימה לא ממויינת קשה יותר לעשות פעולות שהיו פשוטות יותר אילו הרשימה הייתה ממויינת כמו מקס', מינ', חציון וכו'.

1. מיון בועות –

מסתכל על המערך כזוגות, עובר על האיבר הראשון והשני, ואז שני ושלישי וכן הלאה, אם יש זוג שלא מסודר כמו שצריך נבצע החלפה ביניהם. אם נחשוב על זה לעומק נבין כי בכל איטרציה האלגוריתם 'מביא' לסוף המערך את המקסימום הנוכחי. אם גודל המערך הוא n אז למעשה לקחת את n-1 המספרים הגדולים ולהעביר אותם לn-1 המקומות האחרונים:

**public** **static** **void** bubblesort(**int** []arr) {

**for** (**int** j=1; j<arr.length-1; j++) {

**for** (**int** i=0; i<arr.length-1; i++) {

**if** (arr[i]>arr[i+1])

*swap*(arr,i,i+1);

}

}

}

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

**int** arr[]= {1,3,5,7,1,2,6,5,2,4};

*bubblesort*(arr);

System.***out***.println(Arrays.*toString*(arr));

}

**יש לשים לב שבהדפסת המערך חייב להשתמש ב Arrays.*toString*!**

נשים לב שניתן לשפר את הקוד הזה, כי אם המערך ממויין כבר או כמעט ממויין אין מה לעבור על כל פקודות הלולאות. לכן ניתן להוסיף לקוד בדיקה האם התבצע שינוי מסוים ואם לא אז לצאת מהלולאה.

**public** **static** **void** bubblesort(**int** []arr) {

**boolean** haschanged=**true**; //whether we replaced two elements

**for** (**int** j=1; j<(arr.length-1) && (haschanged); j++) {

haschanged=**false**; //before enter the iteration i guess that sorted

**for** (**int** i=0; i<arr.length-1; i++) {

**if** (arr[i]>arr[i+1])

*swap*(arr,i,i+1);

haschanged=**true**;

}

}

1. מיון בחירה –

דומה למיון בועות אלא ששונה מעט –שמחפש את המינימום: עובר על כל אינדקס אחד אחרי השני ושואל על האינדקס – מי המינימום מאינדקס זה ועד הסוף ומחליף ביניהם. למשל עבור המערך 6,1,2,8,4,3,5

יתחיל עם אינדקס 6 – יחליף בינו לבין 1. לאחר מכן יעבור לאינדקס 2 – הוא שוב 6, ויחליף בינו לבין המינימום ממנו ועד הסוף – יחליף בינו לבין 2. לאחר מכן יבדוק מי המינימום בינו – 6 – ועד הסוף ויחליף אותו עם 3.

**public** **static** **void** selectionsort(**int** []arr) {

**for** (**int** i=0; i<arr.length-1; i++) {

**int** minIndex=*getminIndex*(arr,i);

*swap*(arr,i,minIndex);

}

}

**public** **static** **int** getminIndex(**int** arr[], **int** i) {

**int** answer= i;

**for** (**int** j=i; j<arr.length; j++) {

**if**(arr[j]<arr[answer])

answer = j;

}

**return** answer;

}

**public** **static** **void** swap (**int** []arr, **int** i, **int** j) {

**int** temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

1. מיון הכנסה –

האלגוריתם שם מעין קיר הפרדה בין האיבר הראשון לבין יתר האיברים, ובעצם יוצא מנקודת הנחה שמי שבצד אחר של הקיר ממויין ומי שמהצד השני לא ממויין.

למשל עבור המערך 6,1,2,8,4,3,5 הוא מתייחס ל6 כממויין ואז לוקח את 1 ומכניס משמאל ל6, ואז את 2 ומכניס בין 1 ל6, ואז 8 מכניס מימין ל6 וכן הלאה.

**public** **static** **void** insertionsort(**int** []arr) {

**for** (**int** i=1; i<arr.length-1; i++) { //elements before i is sorted

**for**(**int** j=i; (j>0) && (arr[j-1]>arr[j]); j--)

*swap*(arr,j,j-1);

}

}

**public** **static** **void** swap (**int** []arr, **int** i, **int** j) {

**int** temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

**מבוא לתכנות מונחה עצמים**

עצמים – אובייקטים מסוימים במערכת כך שלכל אובייקט יש תכונות ומאפיינים הרלוונטיים אליו ספציפית.

למשל – יש לנו עצם – רכב, מהעצם הזה יכולים להיות מספר עצמים – מספר רכבים שונים כך שלכל רכב יש מאפיינים שונים כמו יצרן, דגם, שנה, צבע וכו' – לכל אלו נקרא תכונות.

מה אבל כן משותף ביניהם? יכולת תנועה, הדלקת אורות, הפעלת ווישרים – לכל אלו נקרא פעולות/ מתודות.

לצורך זה נכיר את מושג האב-טיפוס/מחלקה – לכל העצמים האלו יש קטגוריה אחת – רכב.

המחלקה היא למעשה אב-טיפוס המגדירה איך העצם אמור להיראות , (למשל קובעת שיהיו לו את התכונות: יצרן, דגם, שנת ייצור, צבע) לכן עצם הינו ישות בפועל שנוצרה ממחלקה. לדוגמא הרכב פיאט 500 בצבע אדום הינו ישות של הטיפוס רכב שיש לו את התכונות שלו.

בתוך מחלקה כזו לא נכניס את הpublic static void main.

מקובל שאת שם המחלקה נתחיל באות גדולה, ולרשום את המאפיינים של המחלקה.

למשל-

**public** **class** Car {

//attributes

String company;

**int** year;

String model;

String color;

String fuel;

}

לאחר שהגדרנו את המאפיינים נבנה פעולת **בנאי** שלמעשה יהיה ניתן לקרוא למחלקה ולהגדיר לה אובייקטים חדשים עם אותם מאפיינים - למשל ייצור של כמה מכוניות והכנסת המאפיינים שלהם.

**public** car (String comp, String mod, **int** year, String color) { //בנאי - שיוך למשתנה

**this**.company=comp;

**this**.model=mod;

**this**.year=year;

**this**.color=color;

לאחר מכן צריך להגדיר את המשתנה החדש דרך הmain ולתת מאפיינים לאובייקט הספציפי מסוג המחלקה – לפי הסדר שהגדרתי בבנאי!

car c1;

c1 = **new** car ("fiat", "500", 2010, "blue");

אם אני אנסה בשלב זה להדפיס אקבל את המקום בזיכרון של האובייקט: car@626b2d4a

אני כן יכול לבדוק מה המאפיין המסוים של אובייקט System.***out***.println(c1.company);

וכך נעשה לכל אחד מהמאפיינים.

עם זאת ניתן לעשות פעולה אחרת להדפסה ע"י פעולת toString, אך צריך לעשות לזה שכתוב:

**public** String toString() {

**return** "company: "+ **this**.company+", year: "+**this**.year+", model: " +**this**.model+", color: "+**this**.color;

פעולת בנאי ברירת מחדל – לא נכתוב כלום בפרמטרים אלא נשאיר סוגריים ריקים

**public** car() {

company="b.m.w";

model="x6";

year=1990;

color="orange";}

במצב זה אם נקרא ישירות לאובייקט עצמו ללא הגדרת המשתנים – ידפיס לנו את הברירת מחדל.

car c2=**new** car();

System.***out***.println(c2);

ניתן אף לשלב בין ברירת מחדל לבין מה שאני רוצה שהמשתמש יכניס, למשל:

**public** car (**int** year, String color) {

**this**.company="volvo";

**this**.model="i8";

**this**.year=year;

**this**.color=color;

**בנאי מעתיק**

במידה ואני רוצה לייצר אובייקט חדש עם אותם משתנים כמו של משתנה אחר שכבר יצרתי אני לא יכול פשוט ליצור אובייקט חדש ולהגדיר ששווה לאחר – כי אז שניהם יצביעו לאותו מקום ובשינוי אחד גם השני ישתנה:

car c3 = **new** car();

c3=c2;

לכן כדי להעתיק נתונים אבל לא להצביע על אותו מקום בזיכרון נרשום:

**public** car(car other) {

**this**.company=other.company;

**this**.model=other.model;

**this**.year=other.year;

**this**.color=other.color;

לאחר מכן כאשר נרצה ליצור אובייקט חדש עם אותם נתונים נוכל לרשום:

car c3 = **new** car(c2);

אנו יכולים ליצור פונקציות על האובייקט, למשל אם נוסיף מאפיין fuel, נוכל לעשות למשל פונקציה שבודקת כמה דלק יש ומתריעה כאשר הוא נגמר:

**public** **boolean** move(**int** km) {

**if**(fuel<=km\*0.1) {

System.***out***.println("car: "+ **this**.company+", model: "+**this**.model+"has been move: "+km+" km");

fuel=fuel-km\*0.1;

**return** **true**;

}

System.***out***.println("there is no more fuel");

**return** **false**;

}

נבנה מחלקה חדשה בשם person כדי לתרגל דבר נוסף:

ניתן ליצור בנאי מעתיק במהירות דרך source – generate constructor using fields

יופיע לי לפי כל הפרמטרים, אלא שכרגע יש למחוק את הsuper ( )

**public** Person(String name, **double** age, String id, **double** height) {

**super**();

**this**.name = name;

**this**.age = age;

**this**.id = id;

this.height = height;

הערה נוספת – ניתן להגדיר במחלקה את המאפיינים עם קו תחתון לפניהם ואז להגדיר אותם בלי this.

מאפיין סטטי – משתנה עצם שקבוע כמאפיין לכל העצמים תחתיו.

**שילוב מערכים ואובייקטים**

נבנה תוכנית שמקבלת אנשי קשר ושומרת אותם במערך.

המאפיינים שיש לכל איש קשר - שם, מספר, והמספר סידורי שלו

הפעולות שאני רוצה שיעשה - הוספה, מחיקה, וחיפוש

מאפיינים נוספים שאני צריך - כמה אנשים יש לי כרגע וכמה אנשים אני יכול להוסיף (מקום בזיכרון).

**תרגול אלדד**

**מערך דינאמי (מערך רשימה) – Array List**

זהו מעין שילוב של מערך ורשימה המקצר את הכתיבה ובעיקר – עוזר בכך שהוא לא סגור למספר איברים קבוע מראש. כאשר אנחנו יוצרים מערך אנו צריכים להגדיר מראש כמה תאים לשמור, במערך דינאמי הוא מתחיל ממקום אחד במערך וכל פעם שמנסים להוסיף איבר כאשר המערך כבר מלא, הוא יוצר מערך חדש באורך של פי 2 מאורך המערך שעליו דיברנו, מעתיק את כל האיברים מהמערך הקודם למערך החדש, ומוסיף את האיבר בחדש שהכנסנו. למשל אם יש לנו מערך שהכנסנו אליו 4 איברים כבר, כאשר נכניס את האיבר החמישי הוא ייצור מערך חדש של 8 תאים ויכניס את כל ה4 עם החמישי החדש ומעכשיו יתייחס לחדש כמערך ואת הקודם יזרוק. (הערה - ככה זה גם עם מחיקה, אם היה לי 8 במערך ומחקתי עד שנשאר לי רק 2 – אז הוא ימחק את המערך של ה8 ויקטין את המערך למערך של 4 – (לא של 2).)

למערך דינאמי יש פונקציות מובנות, למשל:

ArrayList arr = new ArrayList<>();

arr.add(5);

arr.add(2);

arr.add(10);

ברירת המחדל זה טיפוס מסוג int אבל ניתן להוסיף בין ה<> את סוג הטיפוס וכך יוגדר המערך

משהו מיוחד שיש ברשימה/מערך הזה – שכל פעם שאני מכניס איבר לתוך המערך כאשר כל המקומות כבר תפוסים – "מאחורי הקלעים" הוא הוא יוצר מערך חדש בגודל כפול מגודל המערך שהתמלא ומכניס למערך החדש את כל האיברים מהקודם לפי הסדר ואז מוסיף את האיבר החדש.

פעולות נוספות שיש במערך רשימה:

arr.remove(2); - מסיר את הפעם הראשונה שמופיע איבר עם הערך 2

arr.get(2); - מחזיר למשתנה/להדפסה את האיבר באינדקס 2

arr.size( ); - מחזיר את גודל המערך

אם כן למדנו 4 פונקציות שחייבות להיות כאשר אנו יוצרים מערך כזה (או כל מבנה נתונים), שנקרא לו קונטיינר container, שיכיל את הפונקציות – add, remove, size, get.

**שיעור 11 – 26/12/21**

יש 2 סוגים של מתודות/פקודות שניתן להוסיף כדי לקבע נתון מסוים, כל אחד עם המאפיינים שלו.

Final – ערך סופי, למשל : final int a = 7; לא יהיה ניתן לשנות אותו.

Static – משתנה קבוע ש"חשוף" לכל מי שיקרא לו. למשל אם אני רוצה לעשות ספירה של דבר מסוים אני יכול להגדיר בהתחלה שהוא שווה ל0 ובכל פעם שיקרה דבר שארצה אעשה ++.

הערה - Final זה ערך מספרי, וstatic זה מקום בזיכרון ש"קוראים" לו. נראה זאת בהקשר של הנושא:

נראה דרך נוספת להשתמש באובייקטים, למשל – רשימת אנשי קשר.

נתחיל בהגדרת המאפיינים ולאחר מכן פעולת בנאי – חייב לעשות זאת בכל יצירת אובייקט:

public class Contact {

String \_name;

String \_PhoneNumber;

int \_id; //סידורי מספר

static int *counter* = 0;

public Contact(String name, String PhoneNumber) {

\_name = name;

\_PhoneNumber = PhoneNumber;

*counter*++;

\_id = *counter*;

}

ניתן עכשיו להוסיף אובייקטים חדשים עם המאפיינים שהגדרנו:

Contact c1 = new Contact ("Dani","9761248");

אבל אז הוא יהיה עצמאי, אם אני רוצה לאחד אותם לרשימה אני אבנה מערך עם כמות אנשי הקשר שיהיה ניתן לשמור ואבנה פונקציה להוספת אנשי קשר:

list = new Contact[500];

public void addContact(String name, String PhoneNumber) {

if (size == list.length) {

resize(); //פונקציה אחרת שבניתי כדי להגדיל את המערך

}

Contact c = new Contact(name, PhoneNumber);

list[size] = c;

size++;

}

כעת ניתן לשמור איש קשר חדש:

lst.addContact("moshe", "04-99999");

דגש חשוב – אם אני אנסה להדפיס איש קשר מסוים זה ידפיס לי את המקום בזיכרון של אותו עצם, ולא את הערכים עצמם, לכן בתוך גוף המחלקה שהגדרתי את המאפיינים אוסיף פונקציה שהופכת למחרוזת ואז בכל פעם שהאובייקט יקבל קלט שאמור להיות מחרוזת – ידפיס מחרוזת:

public String toString() {

return "Contact [\_name= " + name + ", PhoneNumber= " + PhoneNumber +", id= "+ id + "]";

ניתן לעשות פונקציית חיפוש:

public Contact[] search(String name) {

Contact[] res;

int c = 0;

for (int i = 0; i < size; i++) {

if (list[i].\_name.equals(name)) {

c++;

}

}

res = new Contact[c];

c = 0;

for (int i = 0; i < size; i++) {

if (list[i].\_name.equals(name)) {

res[c] = list[i];

c++;

}

}

return res;

}

או פונקציית הסרה של איש קשר (איתור המקום במערך של זה שחיפשנו, ולאחר מכן החל מאותו אחד שחיפשנו נבצע השמה בכל איבר את הערך שמימינו - שכל i יקבל להצביע על i+1.):

public Contact remove(int id) {

int j = -1;

for (int i = 0; i < size; i++) {

if (list[i].\_id == id) {

j = i;

break;

}

if (j == -1) {

System.***out***.println(id + " is not in the list");

return null;

}

}

Contact c = list[j];

for (int i = j; i < size - 1; i++) {

list[i] = list[i + 1];

}

size--;

list[size] = null;

return c;

}

**Private**

למדנו על public שזו קידומת למחלקה/ פונקציה ששניתן לגשת ולקרוא לה. אני יכול להגדיר משתנה פרטי למשל עניין מסוים שאני לא בהכרח רוצה לאפשר לגורם חיצוני לשנות. למשל שם של מקצוע, או ציון סופי וכו. עם זאת אני יכול להגדיר לאיזה מהמשתנים אני מעוניין שתהיה יכולת לשנות (set) ומה מהמשתנים תהיה יכולת לראות (get).

**רשימה מקושרת / שרשרת חוליות**

הייתי רוצה מבנה שלכל משתנה לא רק שהוא שומר את הערך במשתנה אלא שומר מקום פנוי בזיכרון בשביל המשתנה הבא. למה? כדי שתהיה אפשרות לסווג משתנים 'דומים', למשל אנשי הקשר שלי שמתחילים בא' ובב' וכו'. מה חשוב לזכור לפני שניכנס לעניין? שיצירת משתנה שנרשום לו = לעצם שכבר קיים – הוא לא מעתיק אותו אלא שווה לו ממש, כלומר מצביע על אותו מקום בזיכרון כך ששינוי של העצם ישנה גם אותו.

כל חוליה כזו נכנה בשם node, והיא מכילה ערך (data), ומצביע למקום הבא (next)

public class Node {

int data;

Node next;

public Node (int data, Node next) {

this.data=data;

this.next=next;

}

public String toString() {

return ""+data;

}

לתא הראשון נקרא head ולתא האחרון tail.

הרבה פעמים בבנייה של השרשרת אנחנו נבנה מהסוף להתחלה, נגדיר למשל 4, שלא מצביע על כלום. 3 עם מצביע על 4, 2 עם מצביע על 3, ו1 עם מצביע על 2.

public static void main(String[] args) {

Node p3= new Node(-2, null);

Node p2= new Node (5,p3);

Node p1= new Node (100,p2);

System.***out***.println(p3);

System.***out***.println(p1.next); ידפיס את הערך של אחד אחריו – 5.

דרך לעבור על כל השרשרת ולהדפיס אותה:

Node pointer =p1;

while (pointer!=null) {

System.***out***.println(pointer+", ");

pointer=pointer.next;

היתרון של מערך זה שכדי לגשת לתא מסוים אני פשוט מכניס ישירות את המספר שלו ועובר אליו, אבל ברשימה מקושרת אם אני רוצה לגשת לאיבר ספציפי (למשל במיקום ה6) אני חייב לעבור על כל האיברים עד אותו איבר ולעשות לכל אחד מהם next.

היתרון של רשימה מקושרת זה שלא חייב לדוגמא להגדיר מראש כמה איברים יש לי, בשונה ממערך שחייב להגדיר, וניתן ברשימה מקושרת להוסיף איברים באמצע השרשרת בצורה פשוטה מאוד. ועוד כל מיני פעולות שנראה בהמשך.

איך נוסיף איבר באמצע? ניצור node חדש, ונגדיר לו להצביע על האיבר שנרצה שיהיה אחריו, ואז נגדיר לאיבר שלפניו להצביע עליו.

למשל נבנה פונקציה שמוסיפה חוליה כאיבר ראשון ופונקציה שמוסיפה חוליה כאיבר אחרון:

public void addFirst(int data) {

Node newNode= new Node(data,head);

head=newNode;

size++;

}

public void addLast(int data) {

Node pointer= head;

Node newNode= new Node(data);

if(size>0) {

while (pointer.next!=null) {

pointer=pointer.next;

}

pointer.next(newNode);

}else {

head = newNode;

}size++;

}

פונקציה שמדפיסה את השרשרת כמחרוזת:

public String toString() {

Node p = head;

String st= "";

if(p!=null) {

while(p.next!=null) {

st=st+p+", ";

p=p.next;

}

st=st+p;

}

else System.***out***.println("list is empty");

return st;

}

**שיעור 12 – 02/01/2022**

מחיקה של איבר ראשון בחוליה (אבל אותו רציונל בכל מחיקה)

ניצור משתנה עזר temp, נגדיר לו להצביע על הhead, נעביר את הhead להצביע על אחד אחריו, ואז נגדיר את temp.next=null כלומר לא להצביע על כלום

בג'אווה אותו איבר יישאר בזיכרון פשוט הוא כבר לא יהיה מקושר לשום דבר ולא תהיה אליו גישה, בשפת C למשל נצטרך גם למחוק אותו מהזיכרון.

public Node remove (int data) {

if (size>0) {

if(head.getData()==data) {

Node temp = head;

head=head.getNext();

temp.setNext(null);

size--;

return temp;

}

Node prev = head;

while ((prev!=null) && (prev.getNext().getData()!=data)) {

prev=prev.getNext();

}

if(prev.getNext()==null)

return null;

if (prev.getNext()==last)

last=prev;

Node temp=prev.getNext();

prev.setNext(temp.getNext());

temp.setNext(null);

size--;

return temp;

}

return null;

}

ניתן גם לעשות פונקציה המשתמשת בפונקציה זו ומחוקת את כל המופעים של אותו ערך בשרשרת:

public void removeAllOccurences(int data) {

int i=0;

while (remove(data)!=null) {

System.***out***.println("remove the #"+i+" place of "+data);

i++;

}

פונקציה אם אנחנו רוצים לקבל את המיקום של ערך מסוים ברשימה מקושרת:

פונקציה קצרה מאוד להוספה של מספר איברים לתוך רשימה מקושרת:

public void addFirst (int ...arr) { //list of parameters

for (int x:arr) {

addFirst(x);

}

נשים לב שהכנסנו בתוך הסוגריים משתנה חדש שלא ראינו לפני כן : (int ...arr)

במצב זה הפונקציה מקבלת מספר ערכי מספרים שלמים למשל (4,8,6,8,2,4,12) הפונקציה כביכול תשמור אותם במערך ותכניס אותם אחד אחד כאינטים לפונקציה אליה הגדרנו.

**Interface- ממשקים**

מעין 'חוזה' שאם אני מגדיר אותו בין 2 מחלקות לדוגמא כל הפונקציות והמשתנים צריכים להתאים לחוזה הזה – מעין להכריח מחלקה לממש פונקציה שהגדרתי לה.  
למשל אם אני אגדיר חוזה למחלקות של צורות גאומטריות. כולן צריכות להכיל אפשרות של:

1. שטח
2. היקף
3. מקבלת נקודות של המצולע
4. מרכז המסה – מרכז הצורה.

הגדרת ממשק:

יצירת החוזה ע"י הוספה אחרי הגדרת הממשק בשורה הראשונה implements GeoShape – ממי יורש.

ניקח למשל את הממשק comparable שכבר מוגדר בתוך ג'אווה, אז כל מחלקה המיישמת את הממשק צריכה למשל ליישם את הפונקציה : public int compareTo (object obj)

לדוגמא אם נגדיר שהאובייקט שלנו זה Point המיישמת את comparable נכתוב:

Public class Point implements comparable {

…

Public int compareTo (object obj) {…}

}

מעכשיו נוכל להגדיר מערך שמכיר כל דבר שמממש את comparabale כולל את הPoint, (כרגע אני לא בהכרח יודע מה יש בממשק comparabale אבל אני יודע לפחות שיש בו את compareTo)

[טיפ – פונקציית compareTo בודקת השוואה בין 2 ערכים, אבל יכול להיות שאני רוצה לבדוק מול ערכים של נקודה, ומכניסים לי בקלט משתנה מסוג מחרוזת, או אובייקט של נקודה האם שווה לאובייקט של מכונית – אין אפשר להשוות. לכן אפשר לרשום עם סוגריים משולשים ולהגדיר את סוג הטיפוס.

Public class Point implements comparable <Point> {

…

Public int compareTo (Point obj) }

למשל אנו יכולים להגדיר לArrayList שתקבל רק מחרוזות:

ArrayList<String> arr = new ArrayList<>() ;

arr.add("hii");

נחזור לעניין שלנו – אני יכול להגדיר ממשק – חיה שיש לה 2 פונקציות - קול, והגיל שלה:

Public interface Animal {

Public String talk( );

Public int getAge( );

תחת המחלקה הזו אני יכול להגדיר מחלקות שמממשות פונקציות ממנה ולכן חייבות להשתמש בתנאים של המחלקת אב – זה למעשה מימוש החוזה. למשל - כלב, מממש מחיה, יש לו מאפיינים משל עצמו אבל הוא בוודאות משתמש בפונקציות מהמחלקת אב שלו – הקול והגיל.

Public class Dog implements Animal {

Private final kDog\_year = 7;

Private int age;

Public Dog (int a) {age=a;}

Public String talk( ) {return "bark"};

Public int getAge( ) {return kDog\_year \* age};

צריך לשים לב שזה לא ירושה! ירושה זה לקחת נתון של מחלקת אב, מימוש זה "להתייחס" או להגדיר למחלקת בת מאפיינים לפי אותן פונקציות שקבענו בזו שיש לה ממשק איתה.

באופן הזה אני יכול ליצור מחלקות גם של עוד חיות ואז ליצור חיה ולקבל נתונים וכו':

Animal [ ] a = new animal [10];

a[0] = new Dog (3);

a[1] = new Cat (2); …

for (int i=0; i<a.length ; i++){

system.out.println(a[i].talk( ) );

במידה ויצרנו חיה חדשה - כלב או חתול למשל:

Animal a = new cat( );

המערכת לא יודעת את התכונות של החתול כי עדיין לא הגדרנו אותם, אלא מקבלת תכונות מחיה,  
אבל אם יצרנו כבר תכונות ספציפיות לכלב או לחתול (שהם סוג של חיה הרי😊) אני יכול לשאול האם אובייקט a - האם הוא כלב? ונקבל ערך בוליאני. זה מתבצע ע"י הפקודה **instanceof** :

Boolean b = a instanceof Dog;

ברגע שאנחנו יודעים איזה אובייקט יש לנו אנו יכולים להשתמש בתכונות שלו (זה נקרא casting)

If (a instanceof cat){

((cat)a).meow( );

כלומר אני מגדיר לחיה a שיצרתי שהיא חתול כדי לעשות עליה פעולה מסויימת.

שילוב רשימה דינאמית - ArrayList וinterdace

אני רוצה שיהיה לי את היכולת להוסיף אובייקטים כרצוני לכן ניתן להשתמש ברשימה דינאמית:

public static void main(String[] args) {

ArrayList <Animal> arr = new ArrayList<>();

arr.add(new Cat(2)); //2 is the age

arr.add(new Dog(5));

arr.add(new Cat(1));

for (Animal s : arr) {

System.***out***.println(s.talk);

נשים לב שכאשר הגדרנו כלב אמרנו שהקול שלו יהיה bark ולחתול זה יהיה meow, לכן בהדפסה הוא ידפיס את הקול הרלוונטי לכל אובייקט.

הגדרת דרישה נוספת בinterdace

אני יכול לדרוש ("ליצור חוזה" - interface - ממשק) שכל מי שיקיים את animal אני רוצה שיממש גם את פונקציה/ תכונות ממשק אחר, למשל - comparable. נעשה זאת ע"י **extends** ושם הממשק.

public interface Animal extends comparable<Animal>{

public String talk( );

public int getAge( );

public default int compareTo (Animal o) {

if(this.getAge() < o.getAge())

return -1;

if(this.getAge() > o.getAge())

return 1;

}

return 0;

דוגמא נוספת: ניצור ממשק חדש של סטודנט, וניצור מחלקות שמממשות את התנאים שנדרשים – כלומר בסטודנט נגדיר מה יש לכל סטודנט - שם, גיל, תחום התואר, ממוצע ציונים והאם יש לו חיים. ולכל תואר אחר כך נפרט את הדברים הספציפיים אליו:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

או ליצור אובייקט חדש – סטודנט למדעי המחשב, כמובן לא לשכוח את הגטים והסטים שלו:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

## טיפים ודקויות שטוב לדעת:

1. המרה של מחרוזת למספר שלם תתבצע ע"י: int num=Integer.*valueOf* (str);
2. כשעושים s.length( ) למחרוזת – כן מוסיפים סוגריים כי זו פונקציה של מחרוזות,  
   בעוד שכשעושים arr.length למערך – לא מוסיפים סוגריים כי זו פעולה שמוגדרת בתוך ג'אווה.
3. טריק שימושי לספירה כמה פעמים מופיע תו מסוים במחרוזת :

int count = str.length() - str.replace("s","").length();

1. ג