מבוא לתכנות (Java)

סיכום זה נכתב במהלך הרצאות של הקורס מבוא לתכנות, בסמסטר סתיו תשע"ז (2016-2017) של הנדסת מערכות מידע. ניסיתי לכלול בסיכום את כל החומר שעבר בהרצאות, בתוספת הערות שלי. למרות זאת, אני לא יכול להבטיח שהוא כולל מאה אחוז מהחומר, או שאין בו שום טעויות.

בהצלחה,
יונתן מורג

תוכן עניינים:

מבוא, המושג אלגוריתם

09:22 31 October 2016

בשיעור הזה:

- הגדרת המושג אלגוריתם ודוגמאות, כולל דוגמאת אלגוריתם אוקלידוס למציאת מכנה משותף גדול ביותר.

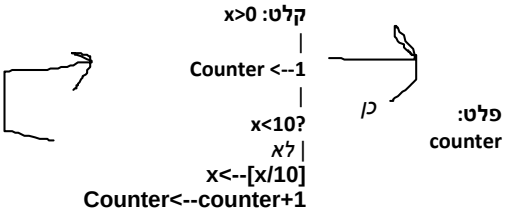
<u>אלגוריתם</u>: סדרה סופית של הוראות חד משמעיות לביצוע משימה מסויימת.
לדוגמא: מתכון לעוגה, מיון דואר.
תכונות האלגוריתם:

- סופיות
- מוגדרות (כל הוראות מוגדרות בצורה חד משמעית)
- נכונות - על כל קלט חוקי להחזיר תשובה נכונה
- קלט
- פלט

תיאור אלגוריתם:

- תיאור מילולי. לדוגמא: אלגוריתם לחישוב ממוצע של K מספרים:
  1. קלט K מספרים ממשיים.
  2. חבר את כל איברי הקלט ותשמור את התוצאה ב-S.
  3. החזר פלט (תשובה) S/K.
- תרשים זרימה. לדוגמא: אלגוריתם מקבל מספר שלם חיובי ומחזיר את כמות הספרות שבו.

3 <- 310
5 <- 11028
1 <- 8



דוגמא: מציאת המכנה המשותף הגדול ביותר.
קלט: n>0, m>0, שלמים
כל עוד (לא מתקיים: i מחלק גם את M וגם את N)
גוף הלולאה i-1 ---> i
עצור: פלט i

אלגוריתם אוקלידוס למציאת מכנה משותף מקסימלי

האלגוריתם מבוסס על הטענה הבאה:
gcd(m,n)=gcd(n,r) כאשר r=m%n (שארית מחלוקה של m ב-n)

הוכחת טענה: נראה של m ו- n ול- n ו- r יש אותה קבוצה של מחלקים ומכאן נובעת הטענה.
m=x*n+r
נניח d מחלק את m ו-n ונוכיח כי d מחלק את n ו-r.
x*n+r מתחלק בd ללא שארית.
כיוון ש- xn מתחלק בd, אז גם r חייב להתחלק בd.

כיוון שני: נניח d מחלק משותף של n ו-r, ונוכיח כי d גם מחלק את m.
m=x*n+r
r מתחלק בd.
x*n מתחלק בd
לכן גם m=x*n+r מתחלק בd ללא שארית.

תיאור האלגוריתם של אוקלידוס

1. קלט: שני מספרים שלמים. n>0, m>0
2. r <-- m%n (חישוב שארית מחלוקה)
3. כל עוד r=/=0
   a. m<--n
   b. n<--r
   c. r<--m%n
4. עצור, פלט n

היכרות עם שפת Java

09:00 01 November 2016

בשיעור הזה:

- היכרות עם שפת java
- מציאת מספר ספרות במספר טבעי
- מציאת המכנה המשותף הגדול ביותר
- מציאת המכנה המשותף הגדול ביותר ע"פ אלגוריתם אוקלידוס
- שלבי העבודה בתכנות

היכרות עם שפת Java

1. המחשב מבין תוכנה בשפה שנקראת שפת מכונה - 001010110100
2. אנחנו משתמשים בשפה עילית, שנוח לתכנת בה
3. מהדר (compiler) תוכנה שמתרגמת משפה עילית לשפת מכונה

בJava התרגום מתבצע לשפה שנקראת bytecode. לאחר מכן יש מתרגם (interpreter), שלוקח פקודה-פקודה, מתרגם לשפת מכונה ומריץ.

מציאת מספר ספרות במספר טבעי

תזכורת/
קלט: 325 => פלט: 3

1. קלט m שלם חיובי
2. Counter <---1
3. כל עוד m>=10
   a. m<--[m/10]
   b. Counter<--counter+1
4. פלט: counter

כעת נכתוב קוד ב-Java שיבצע את הפעולה:

```
import java.util.Scanner;

public class Digits{
	public static void main(String[] args){
		Scanner sc=new Scanner(System.in);
		int m = sc.nextInt();
		int counter=1;
		while(m>=10){
			m=m/10;
			counter=counter+1;
		}
		System.out.println(counter);
	}//main
}//class
```

הוראה לcompiler לייבא משהו שיעזור לנו לקלוט קלט מהמשתמש.

Digits הוא שם המחלקה ונהוג לכתוב אותם עם אות גדולה

int הוא הטיפוס ו-m הוא שם בטיפוס int מספרים שלמים. התוצאה של חילוק תחת int היא ללא שארית

סגירת main. מה שאחרי // הוא הערה ולא נקרא על ידי קומפיילר
סגירת class.

מציאת המחלק משותף הגדול ביותר

תזכורת/

1. קלט m,n
2. i<--m
3. כל עוד (i לא מחלק את m או לא מחלק את n)
   a. i<--i-1
4. פלט i

```
import java.util.Scanner;

public class GCD1{
	public static void main(String[] args){
		Scanner sc=new Scanner(System.in);
		int m = sc.nextInt();
		int n = sc.nextInt();
		int i=m;
		while(m%i != 0 | n%i != 0){
			i=i-1;
		}
		System.out.println(i);
	}
}
```

% מחזיר את שארית החלוקה. =! זה "שונה". | זה "או"

עכשיו ע"פ אלגוריתם אוקלידוס
gcd(m,n)=gcd(n,r)

```
import java.util.Scanner;

public class GCD2{
	public static void main(String[] args){
		Scanner sc=new Scanner(System.in);
		int m = sc.nextInt();
		int n = sc.nextInt();
		int r=m%n;
		while(r != 0){
			m=n;
			n=r;
			r=m%n;
		}
		System.out.println(n);
	}
}
```

שלבי העבודה

בעייה =>אלגוריתם=>תוכנית מחשב=>קומפילציה (+מתקנים קומפילציה) =>מריצים (+שגיאות זמן ריצה)=>
מריצים (+שגיאות לוגיות)

<u>דוגמא</u>:
בעייה: קבע האם מספר טבעי גדול מ1 הוא מספר ראשוני.

1. לקלוט m.
2. לבדוק האם m ראשוני (נרוץ על כל המספרים מ2 עד m-1 ונבדוק האם שארית שונה מ-0.)
3. להחזיר תשובה

```
import java.util.Scanner;

public class Prime{
	public static void main(String[] args){
		Scanner sc=new Scanner(System.in);
		int m = sc.nextInt();
		int i =2;
		boolean isPrime=true;
		while(i<m & isPrime=true){
			if(m%i==0){
				isPrime=false;
			}//if
			i=i+1
		}//while
		System.out.println(isPrime);
	}//main
}//class
```

טיפוס boolean יכול לקבל שני ערכים true/false

אלמנטים בסיסיים בשפת Java, המשך הדפסת ראשוניים

09:07 07 November 2016

בשיעור הזה:

- המשך: הדפסת מספרים ראשוניים
- אלמנטים בסיסיים בשפת java
- מערכים דו מימדיים

המשך: הדפסת מספרים ראשוניים

תזכורת/

```
import java.util.Scanner;

public class Prime{
	public static void main(String[] args){
		Scanner sc = new Scanner(System.in);
		int m = sc.nextInt();
		int i =2;
		boolean isPrime=true;
		while(i<m & isPrime=true){
			if(m%i==0){
				isPrime=false; //m is not prime
			}//if
			i=i+1;
		}//while
		System.out.println(isPrime);
	}//main
}//class
```

מספיק לבדוק האם m מתחלק במספרים $2,3,\ldots,\sqrt{m}$.
נניח p הוא מחלק של m, אז קיים q:
m=p*q
נניח כי p<=q ואם לא אז נחליף ביניהם.
נראה כי $p \leq \sqrt{m}$

```
import java.util.Scanner;

public class Prime{
	public static void main(String[] args){
		Scanner sc = new Scanner(System.in);
		int m = sc.nextInt();
		int i =2;
		boolean isPrime=true;
		while(i*i<=m & isPrime=true){
			if(m%i==0){
				isPrime=false; //m is not prime
			}//if
			i=i+1;
		}//while
		System.out.println(isPrime);
	}//main
}//class
```

כעת יש לכתוב קוד המקבל מהמשתמש מספר שלם n>=2 ומדפיס את כל המספרים הראשוניים עד ל n.
לדוגמא, אם n=32 , יש להדפיס 2,3,5,7,11,13,17,19,23,29,31

נתחיל לעבור על כל המספרים מ2 עד n (כולל), ועבור כל מספר נבדוק האם הוא ראשוני. אם המספר הוא ראשוני, נדפיס אותו.

```
import java.util.Scanner;

public class Prime{
	public static void main(String[] args){
		Scanner sc = new Scanner(System.in);
		int n = sc.nextInt();
		int m = 2;
		while(m<=n){
		//check if m is prime
			int i=2
			boolean isPrime=true;
			while(i*i<=m & isPrime=true){
				if(m%i==0){
					isPrime=false; //m is not prime
				}//if
				i=i+1;
			}//while
			if(isPrime){ // if isPrime=true
			System.put.println(m);
			}
			m=m+1;
		}//while
	}//main
}//class
```

אלמנטים בסיסיים בשפת Java

המידע ששמור על מחשב מיוצג ע" סיביות (bits)
bit = binary digit
byte - רצף של 8 ביטים
כשמגדירים משתנה נותנים לו שם (לדוגמא - m) וטיפוס (לדוגמא - int).
שם המשתנה יכול להכיל אותיות גדולות וקטנות ומספרים. הוא חייב להתחיל באות. האות הראשנוה יכולה להיות גדולה, אבל נהוג להתחיל מאות קטנה. המילים הבאות יתחילו באות גדולה. לא יהיה רווח, אבל לפעמים משתמשים בקו_תחתון במקום רווח.

טיפוסים פרימיטיביים

טיפוס	גודל זיכרון	ערכים
byte	8 bits	-128,...,127
short	16 bits	-32768,..,32767
int	32 bits	-2^31,...,2^31 -1
long	64 bits	-2^63,...,2^63 -1

טיפוס	גודל זיכרון	ערכים
float	32 bits	מספר ממשי (7 ספרות אחרי הנק')
double	64 bits	מספר ממשי (15 ספרות אחרי הנק')
boolean		ערכים בוליאניים (true/false)
char	16 bits	תווים: 'a','E','1'

פעולות שניתן לבצע על מספרים:

+ חיבור
- חיסור
* כפל
/ חילוק
% שארית מחלוקה

המרה (casting):
כשעושים פעולות על ערכים מטיפוסים שונים, המחשב ימיר את שניהם לגבוה יותר לזמן החישוב וגם התוצאה תהיה תהיה בטיפוס הגבוה יותר.

```
Int x=5;
int y=2;
	(double)x/y  //casting has precedence over math operators
the result will be 2.5 (a double)

Int x=5;
int y=2;
	(double)(x/y)
 the result will be 2.0
```

פעולות השוואה:

<,>,<=,>=,==,=!
*(חשוב: שווה אחד (=) זה השמה, שני שווה (==) זה השוואה)

פעולות לוגיות:

&&,& - גם (הכפול אומר שהוא לא ימשיך ויבדוק גם את הערך השני במקרה שהראשון כבר false)
||,| - או (הכפול אומר שהוא לא ימשיך ויבדוק גם את הערך השני במקרה שהראשון כבר true)
! - not

if(*condition*){
things to do
}
...Rest of code

אם יש רק פקודה אחת בif אז לא חייבים לשים סוגריים מסולסלות.
if(){
}else{
}
אם, ורק אם if לא מתקיים, בצע את מה שבelse.

```
if(num1<num2)
	if(num1<num3)
		min=num1;
	else //this else belongs to the second if
	//curly brackets can distinguish which "else" belongs to which "if".
		min=num3;
```

לולאות ומערכים

09:09 08 November 2016

בשיעור הזה:

- לולאות: while, for
- מבוא למערכים

לולאות

While:

תנאי->פקודות-> חזרה לתנאי
ממשיך עד שמפסיק להתקיים התנאי

For:

0 1 3
for(איתחול;תנאי;קידום){
פקודות 2

איתחול: פעולה או נתון לתחילת הלולאה (למשל: int i=2)
תנאי
קידום: פעולה שתתבצע לקראת האיתרציה הבאה.

יש לשים לב במשתנה שהכרזנו עליו בתוך הלולאה יהיה נגיש רק בתוך הלולאה.
יתכן שחלק מאלמנטים יהיו ריקים.

דוגמא:

```
for(i=2;i<m;i=i+2){
        If(m%i==0)
                isPrime-false;
        i=i+1
}
```

דוגמא2:

התוכנית הבאה מגרילה מספר שלם בין 1 ל 100 ומבקשת מהמשתמש לנחש אותו.

```
import java.util.Scanner;

public class Guess{
        public static void main(String[] args){
                Scanner sc = new Scanner(System.in);
                int solution, guess=0;
                solution = 1 + (int)(Math.random()*100;
                while (guess!=solution)
                        guess=sc.nextInt();
                        if(guess<solution)
                                System.out.println("too small");
                        else if (guess>solution)
                                System.out.println("too big");
                }
                System.out.println("you guessed!");
        }//main
}//class
```

Math.random()
נותן מספר ממשי בין 0 ל 1
0<=X<1
ממשי
אם נכפיל ב100
0<=100X<100
אם נגדיר מראש כint, נזרוק את הספרות אחרי האפס וכך נקבל מספר שלם.
יש להוסיף +1 כי אנחנו נקבל 0<=X<100 ואנחנו רוצים 1<=X<=100

else if בגלל שelse יהיה נכון לכל מה שאינו guess<solution, משמע guess>=solution

תרגיל: במסיבה נפגשו 10 אנשים. כל אדם לחץ את ידו של כל אדם אחר. תדפיסו את כל הזוגות של לחיצות ידיים.

```
import java.util.Scanner;

public class HandShakes{
        public static void main(String[] args){
                Scanner sc = new Scanner(System.in);
                int i,j;
                int counter=0;
                for(i=1;i<=10;i+1){
                        for(j=i+1;j<=10;j=j+1){
                                System.out.println(i + "Handshaked" + j);
                                counter=counter+1;
                        }
                }
                System.out.println(counter);
```

```
    }//main
}//class
```

וריאציה: במסיבה נפגשו 10 אנשים. כל אחד ירק לפרצופו של כל אחד אחר. כמה יריקות היו?

```
import java.util.Scanner;

public class Wet{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int i,j;
        int counter=0;
        for(i=1;i<=10;i+1){
            for(j=1;j<=10;j=j+1){
                if(i!=j){
                    System.out.println(i + "Spat on" + j);
                    counter=couner+1;
                }
            }
        }
        System.out.println(counter);
    }//main
}//class
```

מערכים

רוצים להגדיר מספר גדול, או לא ידוע (עד ריצת התכנית), של משתנים מאותו טיפוס.

מערך- (array) - אוסף סדור של איברים מאותו טיפוס.

```
int [] arr;
arr = new int[10];

int [] arr = new int [10];
```

new - תקצה מקום בזיכון למשהו

מערך יוצר כתובת בזיכרון שבו יכולים להיות ערכים. המערך סדור ומתחיל מ-0.

מערך עם חמישה ערכים:

n0	n1	n2	n3	n4
0	8	0	16	0

```
arr[1]=8;
arr[3]=arr[1]*2
```

מילוי מערך במספרים רנדומליים:

```
for(int i=0; i<arr.length; i=i+1){
    arr[i]=Math.random();
}
```

משתמש קובע את גודל המערך:

```
int n=sc.nextInt());
int arr1 = new int[n]
```

מערכים

09:12 14 November 2016

בשיעור הזה:

- מערכים
- "הנפה של ארטוסתנס" למציאת מספרים ראשוניים
- מערכים דו מימדיים

שם [] טיפוס
int [] arr

int[] arr = new int [10]

הקצאת מקום בזיכרון למערך. חייב להיות מאותו טיפוס. לא ניתן לשנות גודלו לאחר שנקבע.

int [] arr1 = new int[0]

החלפת מערך במערך אחר.

int [] arr2 = null

ריק. אין לו כתובות.

לכל מערך יש מאפיין בשם length - גודל המערך (כמות האיברים)

```
for (int i=0;i<10;i=i+1)
        arr[i] = sc.nextInt();
```

כל פעם קולט משתנה מהמשתמש לתוך איבר אחר מתוך המערך על פי הערך של i באיטרציה הנוכחית.

```
double [] a = new double[10];
double b=a
for(inti=0;i<a.length;i=i+1)
        a[i]=i;

System.out.println(b[8]); //8
b[4]=40
System.out.println(a[4]); //40
```

יוצר מערך חדש a עם 10 איברים.
b מצביע על אותו זיכרון של a.
כל שינוי באחד מהם יהיה בשניהם.

טיפוס	שם	ערך
[]double	a	כתובת1
[]double	b	כתובת1
[]int	x	כתובת2
[]int	y	כתובת3

```
int [] x = {1,2,3};
int [] y = {1,2,3};
System.out.println(a==b); //true
System.out.println(x==y); //false
```

x מערך עם שלושה איברים ששווים 1,2,3.

המחשב משווה את הערך של המערך, שהוא כתובת. לא את הערכים שבתוך המערך.

נכתוב תכנית שקולטת מספר שלם חיובי n, יוצרת מערך בגודל n, קולטת n ערכים לתוך המערך, ומוצאת ומדפיסה את האיבר המינימלי במערך ואת האינדקס שלו.

```
import java.util.Scanner;

public class Practice{
        public static void main(String[] args){
                Scanner sc = new Scanner(System.in);
                int [] arr;
                int size = sc.nextInt(); //input array size
                arr = new int [size];
                for (int i=0;i<arr.length;i=i+1){
                        arr[i] = sc.nextInt();
                int minIndex=0;
                int minValue=arr[0]
                for(int j=1;j<arr.length;j=j+1);
                        if(arr[j]<minValue){
                                minValue=arr[j]
                                minIndex=j;
                        }
                System.out.println(minValue + minIndex);
        }
}
```

הנפה של ארטוסתנס למציאת מספרים ראשוניים

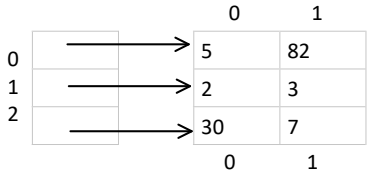
n=30
מתחילים במספר 2. לוקחים את כל הכפולות שלו עד 30 ומוחקים אותן.
עושים את אותו הדבר עם 3, וכן הלאה.
נבנה את זה עם מערך בוליאנים.

```
import java.util.Scanner;

public class PrimeSieve{
```

```
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        boolean [] isPrime = new boolean [n+1];
        for(int i=2;i<=n;i=i+1){
            isPrime[i]=true; //initially asuume all are prime.
        }
        //mark num-primes
        for(int m=2;m<=n;m=m+1){
            if(isPrime[m]) // if this number is prime
                for(int j=2;m*j<=n;j=j+1){
                    isPrime[m*j]=false; //mark as false
            }
        }
        //print all primes
        for( int i=2; i<=n; i=i+1)
            if(isPrime[i])
                System.out.println(i);
    }
}
```

מערכים דו-מימדיים

מערך יכול להכיל בתאים שלו כתובת למערכים אחרים.




```
int [][] arr = new int [3][2];

int [][] arr1={{1,2,3},{4,6,8},{1,2}};
```

ניתן בצורה דומה גם ליצור מערכים תלת-מימדיים, ארבע מימדיים, וכן הלאה.

פונקציות סטטיות

09:11 15 November 2016

בשיעור הזה:

- מבוא לפונקציות

פונקציות

```
public static <return type> <name> (<parameters>){
                            signature

}
```

return type -טיפוס התוצאה שהפונקציה אמורה להחזיר.
Prameters - טיפוסי הקלטים שהפונקציה מצפה להם והפרמטרים (בפונקציה) לתוכם יוכנסו

דוגמא:

```
import java.util.Scanner;

public class GCD {

	public static int gcd (int m, int n){
		int r=m%n;
		while(r!=0){
			m=n;
			n=r;
			r=m%n;
		}
		return n;
	}//gcd


	public static void main(String[]args){
		Scanner sc = new Scanner (Sytem.in);
		int x=48, y=30, z=6;
		int res = gcd(x,y);
		System.out.println(res);
		System.out.println(gcd(y,z));
	}//main
}//classGCD
```

אם הפונקציה באותו class אפשר לקרוא לה. אם היא בclass אחר צריך לקרוא לה דרך classהאחר.
יש לציין טיפוס לצד כל פרמטר

void - הפונקציה לא מחזירה ערך. בjava, פונקציות יכולות לבוא אחרי או לפני main.

קריאה לפונקציה שכתבנו למעלה ומקבלים את ה return של הפונקציה

נכתוב תכנית הקולטת מהמשתמש שני מספרים שלמים ומחשבת ומדפיסה את המינימלי מבין שני המספרים וגם את סכומם.
נחלק לפונקציות:
פונקציה לחישוב המינימלי.
פונקציה לחישוב הסכום.

```
import java.util.Scanner;

public class FunctionExample{
	public static void main(String[]args){
		Scanner sc = new Scanner(System.in);
		int n = sc.nextInt();
		int m = sc.nextInt();
		int res1 = min(n,m);
		int res2 = sum(n,m);
		System.out.println("The minimum is " + res1);
		System.out.println("The sum is " +res2);
	}//main

	public static int min(int n, int m){ //return the minimal of two numbers.
		int minValue;
		if(m<n)
			minValue=n;
		else
			minValue=m;
			return minValue;
	}//function min

	public static int sum(int m, intn){ //returns sum
		int res=n+m;
		return res;
	} //sum
} //class
```

התוכנית הבאה קולטת מהמשתמש:

- גודל המערך
- ערכים לתוך המערך
- ערך נוסף

על התוכנית להחזיר את אינדקס הערך הנוסף בתוך המערך, או 1- אם הוא לא נמצא במערך.

```
import java.util.Scanner;

public class LinearSearch{
	public static void main (String[]args){
		Scanner sc = new Scanner(System.in);
		int  n = sc.nextInt(); //array size
		int[]arr = new int[n];
		//input elements
		for(int i=0;i<arr.length;i=i+1)
			arr[i]=sc.nextInt();
		int key = sc.nextInt(); //input additional value
		int index = linearSearch(arr,key);
		System.out.println("The index is " + index);
	} //main
	public static int linearSearch(int[]arr, int key){ //traverse on array
		int ans=-1;
		for(int i=0; i<arr.length; i=i+1)
			if(arr[i]==key)
				ans=i;
		return ans;
	}
}//class
```

אם נעשה פה שינויים בarr, הם יהיו קבועים, כי השינויים יהיו באותו מקום בזיכרון של arr המקורי. זאת לעומת המשתנים, כמו key, שהשינויים בהם יזרקו כשהפונקציה תסגר.

פונקציות סטטיות (המשך)

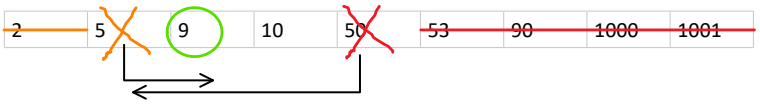
07:16 21 November 2016

בשיעור הזה:

- חיפוש בינארי
- מיונים (Selection Sort, Insertion Sort)
- זריקת חריגה

חיפוש בינארי (Binary Search)

נתון מערך ממויין (סדר עולה או יורד) וערך key. יש למצוא אינדקס של key במערך, או להחזיר 1- אם הוא לא נמצא.



high ו low הם אינדקסים המציינים את ההתחלה והסוף של הקטע שעובדים איתו כרגע.
middle - אינדקס אמצעי בין high ו-low.

```
public static int binarySearch(int[]arr, int key){
	int ans=-1;
	boolean found=false;
	int low=0, high=arr.length-1, middle;
	while (!found && low<=high){
		middle = (low+high)/2;
		if(key == arr[middle]){
			found = true;
			ans=middle;
		} else if(key<arr[middle])
			high=middle-1;
		else
			low=middle+1;
	}
	return ans;
} //function
```

השוואה בין חיפוש בינארי וחיפוש לינארי

המקרה הגרוע ביותר:
Key לא נמצא במערך.
בחיפוש לינארי נבדוק n איברים (n=גודל המערך)
בחיפוש בינארי נבדוק $\log_2(n)$

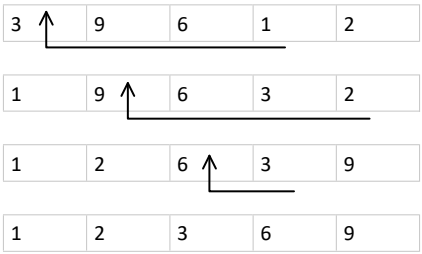
n	חיפוש לינארי	חיפוש בינארי
1000	1000	10~
1000000	1000000	20~
10^9	10^9	30~

מיון

מיון - סידור איברים לפי ערך המפתח.

מיון בחירה

קלט: מערך arr.
פלט: מערך הקלט arr ממויין.



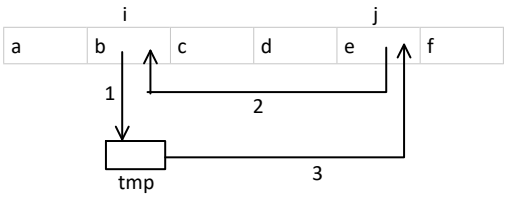
האלגוריתם:

1. מצא את האיבר בעל הערך הנמוך ביותר במערך.
2. החלף בינו לבין האיבר הראשון.
3. המשך באותה השיטה על שאר המערך (מלבד האיבר הראשון).

void כי הפונקציה לא תחזיר ערך. רק תמיין

```
public static void selectionSort(int[]arr){
	int minIndex;
```

```
        //search for minimum from array[i] to arr[arr.length-1]
        for(int i=0; i<arr.length-1; i=i+1){
                minIndex=i;
                for(int j=i+1; j<arr.length; j=j+1){
                        if(arr[j]<arr[minIndex])
                                minIndex=j;
                }
                //swap the values
                swap(arr, i, minIndex);
        }//for i
}//selection sort
public static void swap(int[] arr, int i, int j){
        int tmp=arr[i];
        arr[i]=arr[j];
        arr[j]=tmp;
}
```

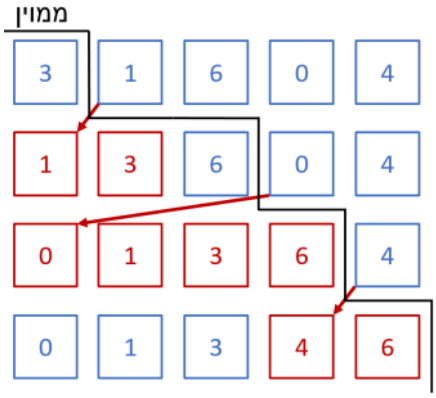


מיון הכנסה (Insertion Sort)

בכל שלב לוקחים את האיבר הבא ומכניסים אותו למקומו הנכון בחלק שכבר מויין.

```
public static void insertionSort(int[]arr){
        for(int i=1; i<arr.length; i=i+1){
                //insert arr[i] into a sorted sequence arr[0]…arr[i-1]
                int key=arr[i];
                int j=i;
                while(j>0 && arr[j-1]>key)
                        arr[j]=arr[j-1];
                        j=j-1;
                }
                //insert key to its right position
                arr[j]=key;
        }//for
}//insertionSort
```

לא מתחילים מ0 כי אין עם מי לבדוק אותו



- את יעילות המיון נהוג למדוד במספר ההשוואות שעושים על איברי המערך. גם הכנסה וגם בחירה עושים בערך n^2 השוואות.

זריקת חריגה (דוגמא עבור קוד המיון שכתבנו):

```
if(arr=null) throw new NullPointerExeption();
```

העמסה (overloading) ומחרוזות

09:15 22 November 2016

בשיעור הזה:

- העמסה
- מחרוזות - פעולות על מחרוזות, שרשור מחרוזות, דוגמאות

העמסה (overloading)

<return type> <name>(<parameters>)
חתימה

ג'אבה מאפשר לכתוב פונקציות שיש להן אותו שם, אבל חתימה שונה. משמע, פרמטרים שונים, או מספר שונה של פרמטרים (לא שמות שונים של פרמטרים).

```
public static double func(int x, double y){
        return x*y
}
public static double func(double x, int y){
        return x+y
}

…… main(….)
        int a=2;
        double b=3.0;
        System.out.println(func(a,b)); //6.0
        System.out.println(func(b,a)); //5.0
        System.out.println(func(a,a)); // comp. error (ambiguous)
        System.out.println(func(b,b)); // comp. error (no such function)
        System.out.println(func(b,(double)b)); // 6.0
```

מחרוזות (string)

מחרוזת היא רצף תווים בין " ". בג'אווה יש מחלקה String שמאפשרת ליצור משתנים מטיפוס String.

יש להקצות לה מקום בזיכון

```
String str;
        str = new String("abc")
        //and together:
String str2 = new String("abc")
```

מחרוזות הן **משתנה לא פרימיטיבי** (מקבלות כתובת בה שמור הערך, לא ערך)
אפשר גם מקוצר:

String str = "abc";

strings are **immutable** (cannot be changed after creation)

*הכרזת final אומר שלא ניתן יהיה לשנות את המשתנה. כך, גם אם הוא משתנה שמקבל כתובת שממילא לא ניתן לשנות את תוכנה, לא ניתן אף לשנות את הכתובת שלו.

ההבדל בין char ל String:

String str="a"; char c='a';

char פרימיטיבי, String לא פרימיטיבי.

```
char ch1='a'; ch2='b';
…print(ch1); //a
…print((int)ch1); //97
…print((char)97); //a
…print('a' +2); //99
```

במחלקה String נמצאות **<u>פעולות שניתן לבצע על המחרוזות</u>**.
API - application programming interface -אוסף של כל המחלקות שבאות ביחד עם סביבת Java.

במחלקה String יש שיטה בשם **charAt(int index)** - מחזירה תו הנמצא במיקום index במחרוזת.

```
str="Hello"
str.cahrAt(2); // l
str.charAt(10); // runtime error
str.charAt(0); // H
```

length() - מחזירה את אורך המחרוזת.

```
int len = str.length(); // len = 5
```

indexOf(char c) - מחזירה את האינדקס של תו c במחרוזת, או 1- אם התו לא נמצא.

```
str.indexOf('e'); //1
str.indexOf('u'); //-1

String str1 = new String ("abc");
        str2 = new String("abc");
        str 3 = 3 str1:
str1==str2 //false
str1==str3 //true
str1.equals(str2) //true
```

<u>שרשור מחרוזות</u>

```
String str4=str1+str2; // str4 ="abcabc"

System.out.println("The answer is" + 3 + 6); // The answer is36
System.out.println(3 + 6 + "The answer"): // 9The answer
```

<u>דוגמאות</u>

נכתוב פונקציה שמקבלת תו c, מקבלת מחרוזת str, ומחזירה מספר המציין כמה פעמים תו c מופיע בתוך str.

```
public static int countChar(String str, char c){
        int counter=0;
        for(int i=0; i<str.length(); i=i+1){
                if(str.charAt(i)==c)
                        counter=counter+1;
        }
        return counter;
}
```

נכתוב פונקציה המקבלת מחרוזת המציינת מספר שלם חיובי בבסיס 10, 8, או 16, ומחזירה מספר שלם מתאים בבסיס 10.

```
int num=10;
int num1=0xA; // base 16
int num2=012; //base 8
```

```
public static int intValue(String str){
        int base; //8, 10 ,16
        int first; //where does the number start
        if(str.length()==1 || str.charAt(0)!='0'){
                base=10;
                first=0;
        } else if(str.cahrAt(0)=='0' && str.charAt(1)=='x'){
                base=16;
                first=2;
        } else {
                base=8;
                first=1;
```

רקורסיה

09:11 28 November 2016

בשיעור הזה:

- רקורסיה - הכרות
- רקורסיה - דוגמאות

רקורסיה הוא תהליך שבו מגדירים משהו במונחים של עצמו.

דוגמא: רוצים לתת הגדרה לרשימה של מספרים (1, 27, 36, 48, 5).
רשימה של מספרים:

- מספר בודד

או

- מספר בודד, פסיק, רשימה של מספרים

פונקציה לחישוב עצרת בעזרת **רקורסיה**

```
public static int fact(int n){
     int ans;
     if(n==0 || n==1)
          ans = 1;
     else
          ans = fact(n-1)*n;
     retrun ans;
}
```

חישוב **איטרטיבי** של !n

```
public static int fact(int n){
     int ans=1;
     for(int i=1; i<=n; i=i+1)
          ans=ans*i;
     return ans;
}
```

סדרת פיבונצ'י: 0 , 1 , 1, 2, 3 ,5 ,8 ,13 ,21...

$F_0=0$
$F_1=1$
$F_n = F_{n-1} + F_{n-2}$, n>1

נכתוב פונקציה המקבלת n ומחזירה את המספר **פיבונצ'י** ה- F_n עם **רקורסיה**

```
public static int fib(int n){
     int ans;
     if(n==0)
          ans=0;
     else if (n==1)
          ans = 1;
     else
          ans = fib(n-2) + fib(n-1);
     return ans
}//fib
```

נכתוב פונקציה דומה עם **איטרציה**

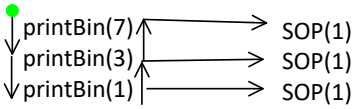
```
public static int fib(int n){
     int ans;
     if(n==0)
          ans=0;
     else if (n==1)
          ans = 1;
     else {
          int fib_n_1 = 1 , fib_n_2 = 0;
          for(int i=2; i<=n; i=i+1){
```

```
            ans = fib_n_2 + fib_n_1;
            fib_n_2 = fib_n_1;
            fib_n_1 = ans;
        }
    }
    return ans;
}//fib
```

פונקציה **רקורסיבית** המקבלת מספר שלם בבסיס 10 ומדפיסה על המסך את ה**ייצוג הבינארי** שלו.

לדוגמא: 8 -> 1000

10 -> 1010

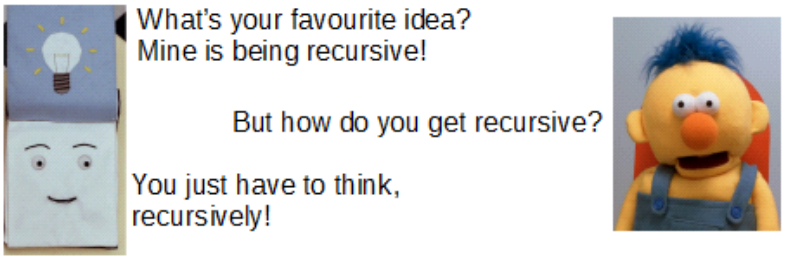



```
public static void printBin(int n){
    if(n>=2)
        printBin(n/2); //print first digits
    System.out.println(n%2); //print last digits
}
```

פונקציה **רקורסיבית** המחשבת את **gcd(m,n)**

תזכורת : gcd(m,n) = gcd(n,r) , r=m%n;

```
public static int gcd(int m, int n){
    int ans;
    if(m%n==0)
        ans=n;
    else
        ans=gcd(n, m%n);
    return ans;
}
```



רקורסיה (עוד דוגמאות)

09:16 29 November 2016

בשיעור הזה:

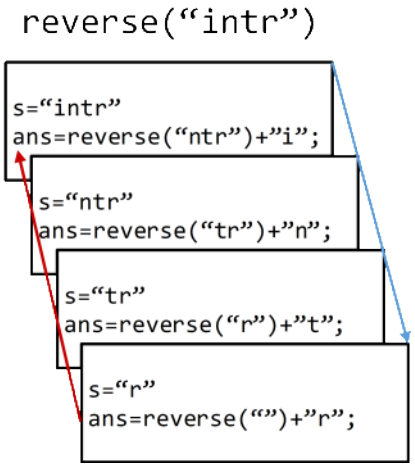
- מחרוזת הפוכה ומגדלי הנואי
- חיפוש בינארי
- מיון מיזוג

חזרה/ פונקציה רקורסיבית חייבת להכיל שני חלקים:

1. תנאי עצירה.
2. חלק רקורסיבי.

בדיקה לתנאי עצירה חייב להיות לפני קריאה רקורסיבית

פונקציה שנמקבלת מחרוזת ומחזירה **מחרוזת הפוכה**:



לדוגמא: "ugb" <= "bgu"

```
public static String reverse(String s){
    String ans;
    if(s.length()==0)
        ans=s;
    else
        ans=reverse(s.substring(1))+s.charAt(0);
    return ans;
}
```

מגדלי הנואי (Hanoi)

נתונים 3 עמודים, נסמנם A, B, ו- C.
על עמוד A נמצאים דיסקים בגדלים שונים. הדיסק הגדול ביותר נמצא בתחתית, הדיסק הקטן ביותר למעלה.
המטרה היא להעביר את כל הדיסקים מעמוד A לעמוד C תוך עמידה בתנאים הבאים:

1. בכל שלב מותר להעביר דיסק אחד בלבד.
2. אסור להניח דיסק יותר גדול על דיסק יותר קטן.
3. אסור להניח דיסקים בצד.

נכתוב **פונקציה** המקבלת מס' n (מס' דיסקים) ושמות של שלושה עמודים: source, destination, extra ומדפיסה על המסך את סדרת ההעברות. hanoi (n, A, B C) (n, source, destination, extra)

```
public static void main hanoi(int n, char source, char destination, char
extra){
    if(n>0){
        hanoi(n-1, source, extra, destination);
        System.out.println("move disk from "+source+" to "+destination);
        hanoi(n-1, extra, destination, source);
    }
}
```

חיפוש בינארי

תזכורת/ נתון מערך ממויין arr וערך key. יש להחזיר אינדקס של key במערך או 1- אם key לא נמצא במערך.

```
public static int binarySearch(int[]arr, int key, int low, int high){
    if(high<low)
        return -1;
    int mid=(low+high)/2;
    if(arr[mid]==key)
        return mid;
    else if(arr[mid]<key)
        return binarySearch(arr, key, mid+1, high);
    else //arr[mid]>key
        return binarySearch(arr, key, low, mid-1);
}
```

מיון מיזוג MergeSort

השוואות $\approx n \log n$

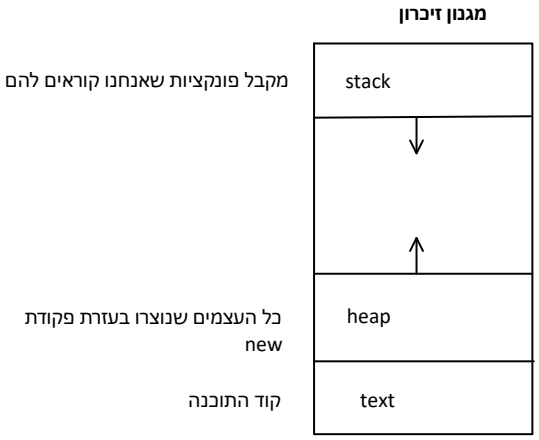
```
public static void mergeSort(int[]a, int low, int high)
     if(low<high) //has more than one element
          int mid=(low+high)/2;
          //recursively sort
          mergeSort(a, low, mid);
          mergeSort(a, mid+1, high);
          merge(a, low, mid, high); //auxilary functinon that merges
     }//if
}
```

רקורסיה, memoization

09:12 05 December 2016

בשיעור הזה:

- רקורסיית זנב
- memoization



רקורסיית זנב

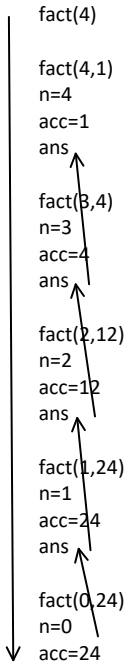
רקורסיית זנב היא סוג של רקורסייה בה הדבר האחרון שנעשה הוא הקריאה הרקורסיבית.
בקוד זה לא נשמרות הסביבות עם המשתנים שלהן אלא קיימת כל פעם סביבה אחת עם ערכים מתאימים.
היתרון הוא שנחסך מקום בזיכרון שמוקצה לתוכנית. אחרת, ברקורסיה ארוכה, הוא עלול להיגמר (ראה תרשים). בג'אווה הסביבות לא באמת נסגרות, אבל בהרבה שפות אחרות כן.

לדוגמא: חישוב gcd.
דוגמא לפונקצייה רקורסיבית שאינה זנב, ולכן לא יכולה למחזר את אותו המקום במחסן: factorial. מאחר ויש פעולת כפל ב(n) שמחכה לתשובה מהפונקציה.

נכתוב את פונקציית fact בצורת רקורסיית זנב:

```
public static int fact(int n, int acc){
    int ans;
    if(n==0)
        ans=acc;
    else
        ans = fact(n-1, n*acc);
    return ans;
}
public static int fact(int n){
    return(n,1);
}
```

מעטפת (wrapper) (הפונקצייה שבסוף משתמשים בה בקוד)



היפוך מחרוזת

```
public static String reverse(String s){
    return reverse(s,  )
}
public static String reverse (String s, String ans)
    String ans;
    if (s.length()==0)
        ans=acc;
    else
        ans=reverse(s.substring(1), s.charAt(0)+acc);
    return ans;
}
```

מעטפת

reverse("intro")

reverse("intro","")

reverse("ntro","i")

reverse("tro","ni")

reverse("ro","tni")

reverse("o","rtni")

reverse("","ortni")

printBin

הפעם נשמור תוצאת ביניים במחרוזת וההדפסה תבוצע בסוף, כשתהיה לנו התוצאה כולה.

```
public static void printBin(int n, String acc){
    if(n=0) System.out.println(acc);
    else if(n==1)
        System.out.println(1+acc);
    else
        printBin(n/2, n%2+acc)
}
public static void printBin(int n){
    printBin(n,"");
```

מעטפת

printBin(7)

printBin(7,"")

printBin(3,"1")

printBin(1,"11") ⟶ "111"

```
}
```

memoization

שמירת תוצאות שחושבו לשימוש עתידי. לפני קריאה רקורסיבית נבדוק האם התוצאה חושבה בעבר, אם כן, נשלוף אותה, אם לא, נחשב את התוצאה ונשמור אותה.
פונקציה לסדרת פיבונצ'י עם memoization (לא זנב):

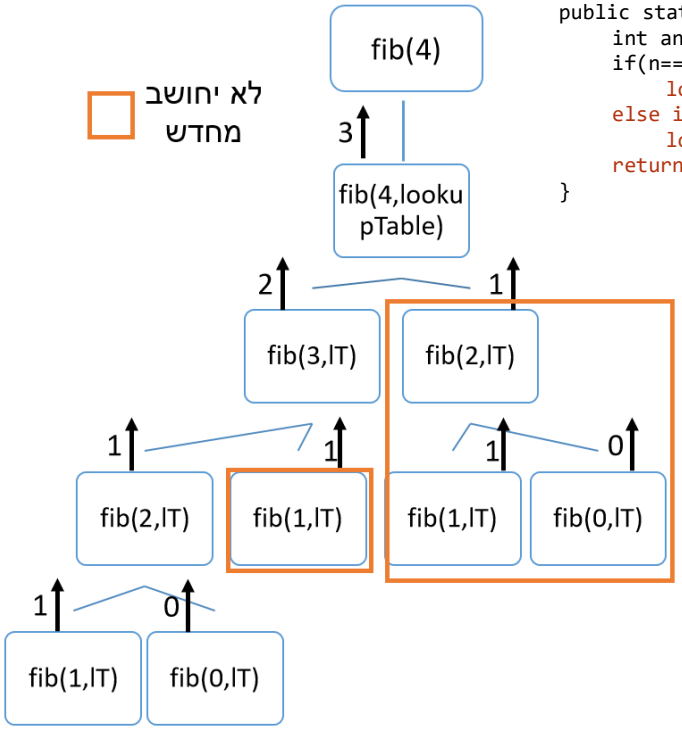
```
public static fib(int n){
    int[] lookupTable = new int[n+1];
    //lookupTable[i] will keep the i-th Fibonacci number
    for(int i=0; i<lookupTable.length; i=i+1)
        lookupTable[i]=-1; //not calculated
    return fib(n, lookupTable);
}
public static int fib(int n, int[] lookupTable){
    int ans;
    if(n==0 || n==1)
        lookupTable[n]=n;
    else if (lookupTable[n]==-1){ //not calculated
        lookupTable[n]=fib(n-1, lookupTable)+fib(n-2, lookupTable);
    return lookupTable[n];
}
```

lookup table

0	1	2	3
-1	-1	-1	-1

lookup table

0	1	2	3
0	1	1	2



רקורסיה, memoization (המשך)

09:13 06 December 2016

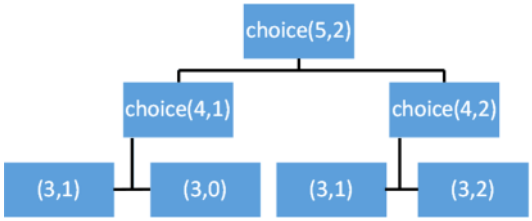
סיכום זה נכתב במהלך הרצאה של הקורס מבוא לתכנות,
בסמסטר סתיו תשע"ז (2016-2017) של הנדסת מערכות מידע.
הוא משקף את החומר שהועבר ע"פ הבנתי בלבד, ולא מהווה
תחליף לנוכחות בהרצאה.
בהצלחה,
יונתן מורג

בשיעור הזה:

- n choose k
- maxUse (אולם הרצאות)

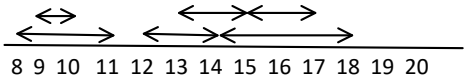
דוגמא: צירופים

בהינתן n איברים שונים, רוצים לבחור k מתוכם. סדר הבחירה אינו חשוב. בחירה מתבצעת ללא חזרות




```
public static int choice(int n, int k, int[][]table){
    if(table[n][k]==0){ //didn't calculate yet
        if(n==k || k==0)
            table[n][k]=1;
        else
            table[n][k]=choice(n-1,k-1,table) + choice(n-1,k,table);
    }//if
    return return[n][k];
}
public static int choice(int n, int k){
    int[][] table=new int[n+1][k+1];
    //default values are zero
    return choice(n,k,table);
}
```

דוגמא: נתון אולם הרצאות



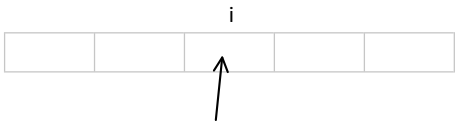
יש בקשות בשעות שלמות (8-11, 9-10, 12-14, 13-15, 15-17, 14-18...). מטרתינו למקסם את הניצול של האולם.
פלט: משך זמן מקסימלי שהאולם נמצא בשימוש

1. נמיין את הבקשות לפי זמני סיום
2. -אם יש 0 בקשות => ניצול מקסימלי הוא 0.
   -אחרת: א. נשבץ את ההרצאה האחרונה, ונוסיף זמן ניצול מקסימלי של כל ההרצאות האחרות שלא מתנגשות איתה. (with)
   ב. לא נשבץ אותה. נחשב זמן ניצול מקסימלי של כל ההרצאות האחרות. (without)
3. נחזיר max(with, without).

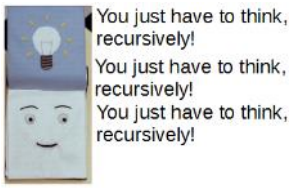
נקבל את הקלט במערך דו מימדי, seg.
זמן התחלה של ההרצאה ה-i -> seg[0][i]
זמן סיום -> seg[1][i]
(memo)

```
public static int maxUse(int[][] seg){
    sortByEndPoint(seg);
    int[]memo = new int [seg[0].length];
    return maxUse(seg, seg[0].length-1, memo)
}

public static int maxUse(int[][] seg, int i, int[]memo){
    int res=0; //removed
    if(i<0)
        return 0;
    if(i>=0){
        if(memo[i]==0){
            int prev = prevSeg(seg, i);
            int with = maxUse(seg, prev,memo)+(seg[1][i]-seg[0][i]);
            int without = maxUse(seg, i-1,memo);
            memo res = Math.max(with, without); //memo instead res
        }
    }
    return res; //memo[i] instead of res
}
public static int prevSeg(int[][]seg, int i){
    int maxAllowedPoint=seg[0][i];
    i=i-1;
    while (i>=0 && seg[1][i]>maxAllowedPoint)
        i=i-1;
    return i;
}
```

המערך ואינדקס של ההרצאה שכרגע מתבוננים בה



ניצול מקסימלי של הרצאות 0,1,...,i



רקורסיה (סיום)

09:09 12 December 2016

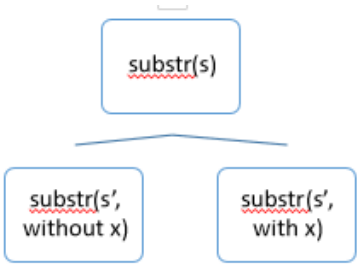
בשיעור הזה:

- הדפסת תת מחרוזות
- מספר מסלולים מונוטוניים באריג

סיכום זה נכתב במהלך הרצאה של הקורס מבוא לתכנות,
בסמסטר סתיו תשע"ז (2016-2017) של הנדסת מערכות מידע.
הוא משקף את החומר שהועבר ע"פ הבנתי בלבד, ולא מהווה
תחליף לנוכחות בהרצאה.
בהצלחה,
יונתן מורג

הדפסת כל תת מחרוזת של מחרוזת נתונה

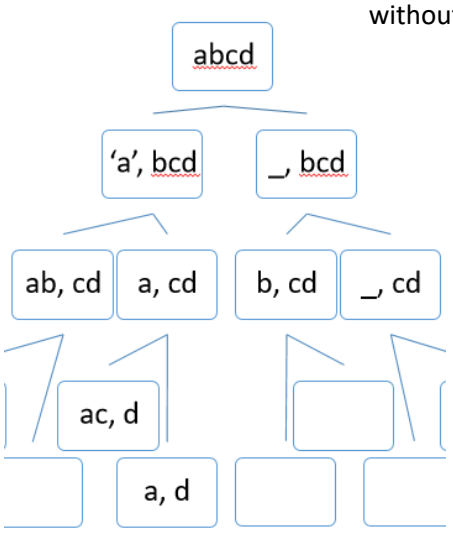
נתונה מחרוזת של תווים. יש להדפיס את כל התת מחרוזות שלה.
תת מחרוזת - רצף של תווים של מחרוזת נתונה השומרים על הסדר שלהם.



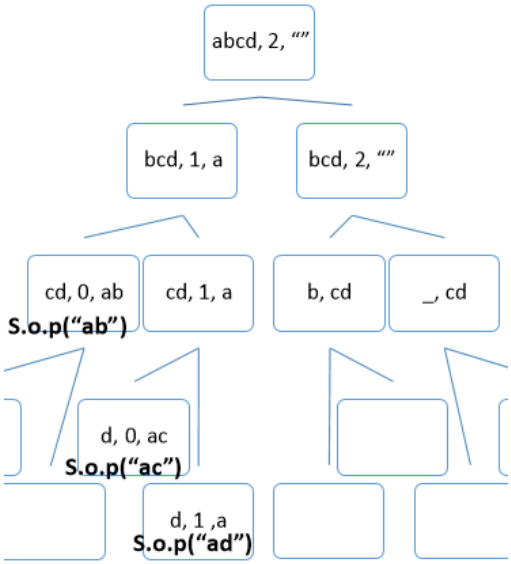

```
public static void substrs(String s){
    substrs(s, "");
}
public static void substrs(String s, String acc)
    if(s.length()==0)
        System.out.println(acc);
    else{
        substrs(s.substring(1), acc + s.charAt(0));
        substrs(s.substring(1), acc);
    }
}
```

בגלל שacc מחרוזת, ומחרוזת אינה ניתנת לשינוי,
כש"משנים" אותה, נוצרת מחרוזת חדשה בזיכרון

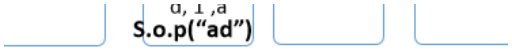
with
without



הדפסת **כל תת מחרוזות בגודל K**
לדוגמא:
k=2: "abcd" --> ab, ac, ad, bc, cd, bd

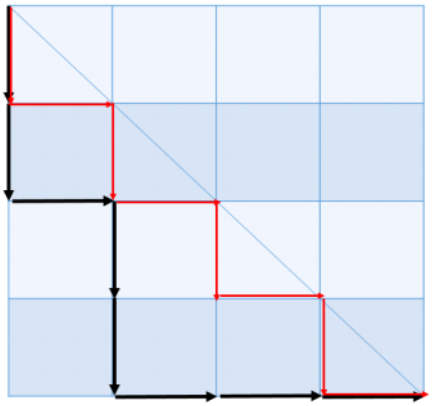



```
public static void substrs( String s, int k){
    substrs(s, k, "");
}
public static void substrs(String s, int k, String acc){
    if (k==0)
        System.out.println(acc);
    else if (k==s.length())
        System.out.println(acc + s);
    else
        substrs(s.substring(1), k-1, acc + s.cahrAt(0));
        substrs(s.substring(1), k, acc);
    }
}
```




```
}
```

מציאת מספר מסלולים מונוטוניים באריג



נתון אריג בגודל n,n. יש להגיע מנקודה (0,0) לנקודה (n,n). תנאים:

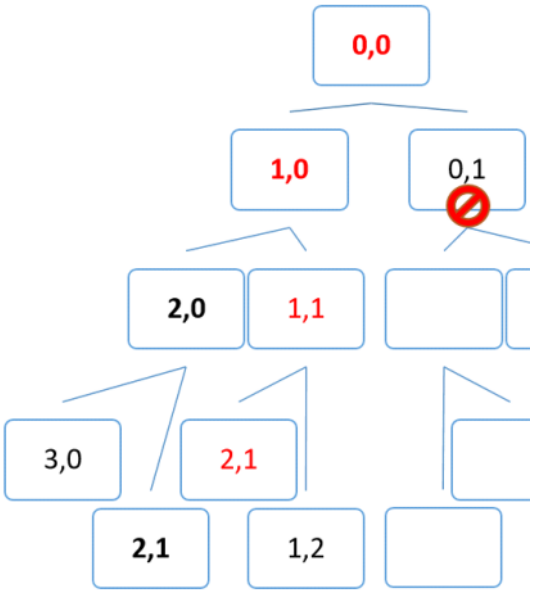
- המסלול בנוי מצעדים למטה או ימינה
- המסלול אינו יעבור בנקודות שמעל האלכסון

יש לכתוב פונקציה (paths) שמקבלת n המציין את גודל האריג, ומחזירה את מספר המסלולים המונוטוניים מ(0,0) ל(n,n).

בכל צעד:

- בוחרים ללכת ימינה
- בוחרים ללכת למטה

צריכים שני אינדקסים (i,j) - קואורדינטות המיקום הנוכחי.
i=n וגם j=n - הגענו.
אם j>i - אנחנו מעל האלכסון ולא ניתן להמשיך.




```
public static int path(int n){
    return paths(n, 0, 0);
}
public static int paths(int n, int i, int j){
    int res=0;
    if(i==n && j==n)
        res = 1;
    else if(i>n || j>n || j>i)
        res = 0;
    else
        res = paths(n, i+1, j) + paths(n, i, j+1);
    return res;
}
```

תכנות מונחה עצמים (Object Oriented Programming - OOP)

09:15 13 December 2016

סיכום זה נכתב במהלך הרצאה של הקורס מבוא לתכנות,
בסמסטר סתיו תשע"ז (2016-2017) של הנדסת מערכות מידע.
הוא משקף את החומר שהועבר ע"פ הבנתי בלבד, ולא מהווה
תחליף לנוכחות בהרצאה.
בהצלחה,
יונתן מורג

בשיעור הזה:

- תכנות מונחה עצמים (OOP) - הכרות
- OOP - דוגמאות

עצם (Object) - ישות בעלת **מצב (state) והתנהגות (behaviour)**.
לדוגמא: בן אדם. מצב - תכונות (צבע עיניים, משקל גובה). התנהגות - איזה פעולות הוא יכול לבצע (ללכת, לדבר).
המצב לרוב יכול להשתנות.
לדוגמא: Scanner.

```
Scanner sc = new Scanner(System.in);
int n = sc.nextInt();
```

Scanner יכול לבצע פעולה של לקלוט קלט

רוצים לכתוב תוכנית שעובדת עם נקודות במישור. נקודה במישור מאופיינת ע"י שתי קואורדינטות (x,y).

```
int x;
int y;
```

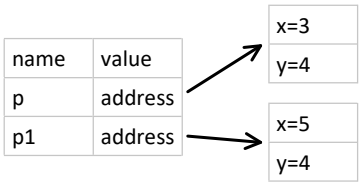
נכתוב טיפוס חדש - Point

```
public class Point{
    int x;
    int y;
}
```

בלי static. אם המain והקלאס החדש (Point) הם תחת אותו קלאס, צריך להוסיף.
משתני המחלקה/שדות (fields)

כשנרצה להשתמש בטיפוס החדש צריך לפתוח זיכרון חדש עם new.

```
… main…
Point p;
p = new Point();
p.x=3;
p.y=4;

Point p1 = new Point();
p1.x=5;
p1.y=4;

System.out.println(p.x); //3
System.out.println(p1.x); //5
p.x = p1.x; // p.x = 5
p1=p // p1 now also points to p's address
```

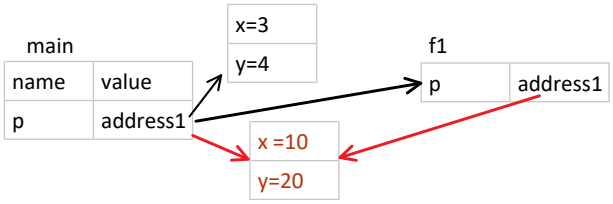


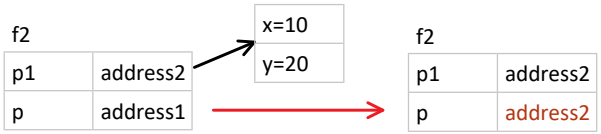
נתנו ערכים למשתנה בְמָקום x באובייקט

name	value
p	address1
p1	address1

נכתוב **פונקציה** שמקבלת פרמטר מטיפוס Point ו**מדפיסה את הקואורדינטות** על המסך.

```
public static void printPoint(Point p){
    System.out.println("("+p.x+","+p.y+")");
}

public static void f1(Point p){
    p.x=10;
    p.y=20;
}

public static void f2(Point p){
    Point p1 = new Point();
    p1.x = 10;
    p1.y = 20;
    p=p1;
}
…main…
printPoint(p); // (3,4)
f2 (p);
printPoint(p); // (3,4)
```





p שבטבלת הערכים של main לא שונה על ידי f2! (לא השתמשנו ב f1)

בעת נוסיף לאובייקט **התנהגות**

```
public class Point{
    int x;
    int y;
    public void printPoint(){
        System.out.println("("+p.x+","+p.y+")");
    }
}
```

הפונקציה יכולה לגשת לשדות של המחלקה, לכן לא צריך להזין לה ערכים.
הורדנו את המילה static כי לא ניתן לקרוא לה מכל מקום אלא רק באובייקט.
לכן, נהוג לקרוא לה שיטה (method) ולא פונקציה.

```
    public void printPoint(){
        System.out.println("("+p.x+","+p.y+")");
    }
}
…main…
Point p = new point();
p.x=3; p.y=4;
p.printPoint(); //(3,4)

Point p1 =  new Point();
p1.x=10; p1.y=20;
p1.printPoint(); //(10,20)
```

הפונקציה יכולה לגשת לשדות של המחלקה, לכן לא צריך להזין לה ערכים.
הורדנו את המילה static כי לא ניתן לקרוא לה מכל מקום אלא רק באובייקט.
לכן, נהוג לקרוא לה שיטה (method) ולא פונקציה.

עוד שיטה

```
public class Point{
    int x;
    int y;
    public void printPoint(){
        System.out.println("("+p.x+","+p.y+")");
    }
    public void translate(int deltaX, int deltaY){
        x=x+deltaX;
        y=y+deltaY;
    }
}
…main…
Point p = new point();
p.x=3; p.y=4;
p.translate(4,1);
p.printPoint(); // (7,5)
```

translate = הזזה של סט נקודות במישור. פה נזיז רק אחת

ניגש למקום של p בזיכרון ועשה את השינוי

נכתוב שיטה ל**חישוב מרחק בין שתי נקודות** ($d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$)

```
public class Point{
    int x;
    int y;
    public void printPoint(){
        System.out.println("("+p.x+","+p.y+")");
    }
    public void translate(int deltaX, int deltaY){
        x=x+deltaX;
        y=y+deltaY;
    }
    public void distance(Point other){
        int dx = x - other.x;
        int dy = y - other.y;
        return Math.sqrt(dx*dx+dy*dy);
    }
}
…main…
Point p = new point();
p.x=3; p.y=4;
Point p1 = new Point();
p1.x=8; p1.y=0;
double dist = p.distance(p1);
```

יש לשיטה גישה לקואורדינטות של p כי p זו שהפעילה אותה. את הקואורדינטות של p1 ניתן לה (בסוגריים), והיא תהיה ה"other" בחתימה של distance.

נכתוב שיטה ל**השוואת נקודות**

השוואה פשוטה תשווה בין הכתובות:

```
Point p1 = new Point();
Point p2 = new Point();
p1.x=5; p1.y=8;
p2.x=5; p2.y=8;
System.out.println(p1==p2); // false
```

אנחנו רוצים להשוות בין הקואורדינטות:

```
public class Point{
    int x;
    int y;
    public boolean equals(Point other){
        return (x==other.x)&&(y==other.y);
    }
}
Point p1 = new Point();
Point p2 = new Point();
p1.x=5; p1.y=8;
p2.x=5; p2.y=8;
```

```
System.out.println(p1==p2); // false
System.out.println(p1.equals(p2)); // true
```

מה יקרה אם other==null? שגיאת זמן ריצה

תכנות מונחה עצמים (המשך)

09:16 19 December 2016

סיכום זה נכתב במהלך הרצאה של הקורס מבוא לתכנות, בסמסטר סתיו תשע"ז (2016-2017) של הנדסת מערכות מידע. הוא משקף את החומר שהועבר ע"פ הבנתי בלבד, ולא מהווה תחליף לנוכחות בהרצאה.
בהצלחה,
יונתן מורג

בשיעור הזה:

- המשך היכרות עם תכנות מונחה עצמים
- בנאים (constructors)

המשך הכרות עם תכנות מונחה עצמים

שיטה שתדפיס את העצם כמחרוזת

```
public class Point{
    public double x;
    public double y;
}
public string toString(){
    return "<"+x+","+y+">";
}
```

ג'אווה מוסיף אוטומטית שיטה toString לכל מחלקה והיא תחזיר את שם העצם והכתובת בזיכרון.
אם נדפיס למשל את העצם P מטיפוס Point שהגדרנו, בלי להוסיף לו שיטה toString משלנו, ג'אווה ישתמש אוטומטית בtoString שהוא יצר.

נוסיף למחלקה Point, שיטה update, המעדכנת את הקואורדינטות של הנקודה

```
public void update(double x , double y){
    x=x; //???
    y=y;
}
```

בעיה - איזה x הוא איזה?

ברגע שהקומפיילר נתקל בשיטה, יש שני מקומות בהם הוא יכול לחפש את המשנתים עבורה. ראשית הקומפיילר יבדוק מבין המשתנים הלוקאליים של השיטה. אם הוא לא מצא הוא יחפש גם במשתנים של המחלקה.
אפשר לשנות את השם של המשתנה הלוקאלי כדי להבדיל אותו, אך קיימת דרך אחרת להגדיר לקומפיילר שישתמש בכתובת של המשתנה בשדות של העצם שהפעיל את השיטה

```
public void update(double x , double y){
    this.x=x;
    this.y=y;
}
```

בנאים

- בנאי היא שיטה מיוחדת שמאתחלת את תוכן השדות.
- היא לא מחזירה שום ערך, אפילו לא רושמים לה void.
- השם של הבנאי הוא כמו שם המחלקה.
- לא ניתן להפעיל את הבנאי במפורש, הוא מופעל אוטומטית בזמן יצירת האובייקט.

```
public class Point{
    public double x;
    public double y;

    //constructor
    public Point(double x, double y){
        this.x=x;
        this.y=y;
    }
}
…main…
Point p = new Point(8,3.2);
}
```

בלי void

הקומפיילר יחפש בנאי עם חתימה מתאימה וישתמש בו

ניתן לעשות **העמסה לבנאים**.
נהוג להגדיר בכל מחלקה: בנאי ריק (ברירת המחדל), בנאי העתקה, ובנאי עם פרמטרים.

```
    publilc Point(){
        this.x=0;
        this.y=0;
    }
```

```
    public Point(Point p){
        this.x=p.x;
        this.y=p.y;
    }
...main...{
    Point p1 = new Point(8,3.2);
    Point p2 = new Point();
    Point p3 = new Point(p1);
```

בנאי ראשון - פרמטרים
בנאי שני - ברירת מחדל
בנאי שלישי - העתקה

אם לא מגדירים אף בנאי במחלקה, ג'אווה מספק בנאי ברירת מחדל, שנותן ערכים 0, null, false. ברגע שהגדרנו איזשהו בנאי, בנאי ברירת המחדל של ג'אווה לא ייווצר, ויש להוסיף אותו.
p1, p2, p3 נקראים מופעים (instances) של מחלקה Point

נגדיר מחלקה **circle** המייצגת מעגל במישור

```
public class Circle{
    public Point center;
    public double r;

    //constructors
    public Circle(Point center, double radius){
    this.center = new Point(center);
    this.r = radius;
    }
    public Circle(){
    this.center = new Point();
    this.r=1;
    }
    public Circle(Circle other){
    this.center = new Point(other.center);
    this.r=other.r;
    }
```

הטיפוס circle מכיל טיפוס ממחלקה אחרת. זה נקרא הכלה (

נהוג לרשום את הבנאים מיד אחרי השדות

אם לא היינו משתמשים ב new Point. אז שני הcenter באובייקטים השונים היו מצביעים על אותו הזיכרון.

נוסיף מספר **שיטות**

```
    // behaviours
    public double area(){
        return Math.pi*r*r;
    }
    public double perimeter(){
        return Math.pi*r*2;
    }
    public boolean isInside(Point p){
        return center.distance(p)<=r;
    }
```

נהוג להוסיף שיטת השוואה שתבצע **השוואה לוגית**

ההשוואה הרגילה תשווה בין הכתובות

```
    public boolean equals(circle other){
        if(other!=null)
            return center.equals(other.center) &&
            r==other.r;
        return false;
    }
}
```

public/private

```
public static void main(String[] args){
    Point p1=new point(1,2);
    circle c1=new Circle(p1,8);
    System.out.println(c1.area);
    c1.center=null; //invalid!!
    c1.radius=-8; // invalid!!
```

אם נהפוך אותה לפרטי אז רק שיטות מתוך המחלקה יוכלו לעדכן ערכים, או שנכתוב שיטות שיקבלו קלט, יבדקו אותו, ואם הוא תקין, יעשו את השינוי.

עקרון ההכמסה (encapsulation)

09:11 20 December 2016

סיכום זה נכתב במהלך הרצאה של הקורס מבוא לתכנות,
בסמסטר סתיו תשע"ז (2016-2017) של הנדסת מערכות מידע.
הוא משקף את החומר שהועבר ע"פ הבנתי בלבד, ולא מהווה
תחליף לנוכחות בהרצאה.
בהצלחה,
יונתן מורג

בשיעור הזה:

- עקרון ההכמסה
- דוגמא - קבוצה של נקודות

עקרון ההכמסה

נרצה להגן על עצמים מהשמות לא חוקיות. נהוג להגדיר שדות של המחלקה להיות פרטיים, בעזרת מילה שמורה private (במקום public).
אם ננסה לגשת לשדה ממחוץ למחלקה נקבל שגיאת קומפילציה. כדי לגשת לשדות פרטיים ולעדכן את הערך שלהם, נגדיר שיטות ציבוריות שמבצעות את העבודה.
השיטות שמעדכנות את מצב העצם נקראות mutators או setters.

```
public class Circle{
    private Point center;
    private double r;

    publilc void setRadius(double r){
        if(r>=0)
            this.r=r;
    }
    public void setCenter(Point p){
        if(p!=null)
            this.center = new Point(p);
    }

…main…
    Circle c1 = new Circle();
    c1.r=-8; //compilation error
    c1.setRadius(10);
    c1.setRadius(-8); //no change
    c1.setCenter(newPoint(3,4));
```

לא יכול לגשת לזיכרון כי הוא Private

נגדיר גם שיטות ציבוריות שמחזירות את הערכים של r ו-center בתוך מחלקה circle. שיטות אלה נקראות getters.

```
public double getRadius(){
    return r;}
public Point getCenter(){
    return new Point(center);}
```

נעתיק את center לנקודה חדשה כדי לא להעביר את הכתובת המקורית של השדה, מה שיאפשר לערוך אותו.

הגנה כזו של מידע מפני שימוש לא חוקי נקרא עקרון ההכמסה (encapsulation), או עקרון ההסגרה / עקרון הקופסא השחורה).

יש שירצו לתת אינדיקציה אם ההשמה התבצעה או לא:

```
publilc boolean setRadius(double r){
    boolean ans = false;
    if(r>=0){
        this.r=r;
        ans=true;
    }
    return ans;
}
```

ניתן להגדיר גם את השיטות להיות פרטיות. בד"כ שיטות עזר נהוג להגדיר כ private.

קבוצה של נקודות

נניח שאנחנו רוצים להחזיק אוסף של נקודות, כלומר אוסף של עצמים מטיפוס Point. נכתוב מחלקה בשם pointSet לנהול קבוצת הנקודות. למען הפשטות, נניח שנתונה קיבולת מקסימלית של הקבוצה (מס' מקסימלי של איברים בכל רגע נתון).

מערך של Pointים
אומר כמה איברים יש בקבוצה, וגם מה המקום הפנוי הבא

elements

p1	p2	p3	...	pn

```
public class PointSet{
    private Point[] elements;
    private int size;

    public PointSet(int capacity){
        elements = new Point[capacity];
        size=0;
    }
    public void add (Point p){
        if(!contains(p)){
            elements[size]=p;
            size=size+1;
        }
    }
    // is this element already a member of this set?
    private boolean contains(Point p){
        boolean found=false;
        for(int i=0; i<size && !found; i=i+1){
            if(elements[i].equals(p))
                found=true;
        }
        return found;
    }
    public void remove(Point p){
        boolean found=false;
        for(int i=0; i<size && (!found); i=i+1){
            if(elements[i].equals(p)){
                elements[i]=elements[size-1];
                elements[size-1]=null;
                size=size-1;
                found=true;
            }
        }
    }
```

נייעל את השיטות בעזרת שיטה חדשה

```
    private int indexOf(Point p){
        int ans=-1;
        for(int i=0; i<size && ans==-1; i=i+1){
            if(elements[i].equals(p))
                ans=i;
        }
        return ans;
    }

    private boolean contains(Point p){
        return (indexOf(p)!=-1);
    }
    public void remove(Point p){
        int i = indexOf(p);
        if(i!=-1){
            elements[i]=elements[size-1];
            elements[size-1]=null;
            size=size-1;
        }
    }
```

דוגמא לשימוש במחלקה PointSet

```
public static void main(string[] args){
    PointSet pSet = new PointSet(100);
    Point p1 = new Point(3,4);
    Point p2 = new Point(1,2);
    Point p3 = new Point(-1,-1);
    pSet.add(p1);
    pSet.add(p2);
    System.out.println(pSet.contains(p1)); //true
```

```
System.out.println(pSet.contains(p3)); //false
pSet.remove(p1);
System.out.println(pSet.contains(p1)); //false
```

נניח שרוצים לכתוב מחלקה CircleSet שמנהלת קבוצה של עיגולים. ההבדל היחיד הוא טיפוס האיברים. כך יהיה גם עם קבוצות אחרות רבות. לכן, נרצה לכתוב קוד פעם אחת עם טיפוס כללי, ויהיה ניתן להשתמש בו עם כל טיפוס אחר. בג'אווה יש **טיפוס כללי** שנקרא **Object**. כל זאת ועוד.. בפרק הבא!

הורשה, פולימורפיזם

09:13 26 December 2016

סיכום זה נכתב במהלך הרצאה של הקורס מבוא לתכנות,
בסמסטר סתיו תשע"ז (2016-2017) של הנדסת מערכות מידע.
הוא משקף את החומר שהועבר ע"פ הבנתי בלבד, ולא מהווה
תחליף לנוכחות בהרצאה.
בהצלחה,
יונתן מורג

בשיעור הזה:

- הורשה
- פולימורפיזם

הורשה

נתחיל מדוגמא:
ברצונינו לכתוב מחלקה המייצגת סטודנט. סטודנט מיוצג ע"י שם, ת"ז, ומספר הקורסים שהוא לומד.

```
public class Student{
    private String name;
    private int id;
    private numOfCourses;
    private static final int COURSE_PRICE=1000;

    public Student(int id, String name){
        this.id=id;
        this.name=name;
        this.numOfCourses=0;
    }
    public int computeTuitionFee(){
        return numOfCourses*COURSE_PRICE;
    }
    public String getName() {
        return name;
    }
    public int getID() {
        return id;
    }
    public int getNumOfCourses() {
        return numOfCourses;
    }
    public void setNumOfCourses(int numOfCourses) {
        this.numOfCourses = numOfCourses;
    }

    public String toString() {
        return "Student name: " + getName()
        + ", ID:" + getID()
        + ", No. of courses: " + getNumOfCourses();
    }
}
```

מה ש static, ניתן לגשת אליו גם בלי שיצרנו אובייקט מטיפוס student,
אותיות גדולות כי זה final
לזכור לא לרשום void בבנאים

לא חייבים this ב numOfCourses

עבשיו נניח שאנחנו רוצים להציג סטודנט מלגאי. סטודנט מלגאי הוא סטודנט, ובנוסף יש לו מלגה וחישוב שכר לימוד שונה. (מ' הקורסים * עלות הקורס - מלגה).

אפשרות א : לשנות מחלקה Student. להוסיף שדה milga, שיהיה 0 עבור סטודנט רגיל.
חסרונות:

- רוב הסטודנטים אינם מלגאים וסתם מוסיפים להם שדה.
- יש צורך לעדכן את הקוד במחלקה Student.
- אם יש שינוי בבנאי, אז כל מי שמשתמש במחלקה Student יצטרך לשנות את הקוד שלו.

אפשרות ב : לכתוב מחלקה חדשה שמייצגת מלגאי. נעתיק את רוב הקוד מStudent, ונוסיף פרמטר מלגה.
חסרונות:

- אם קוד כבר קיים, נרצה להשתמש בו.

אפשרות ג - הורשה (Inheritance) : המחלקה Milgay תרחיב את המחלקה Student.
המחלקה שאנחנו מרחיבים נקראת super class / parent class / base class. מחלקה יכולה להוריש לרבים אבל לרשת רק מאחד. משתמשים בהורשה כשיש יחס "is a". כל מה שקיים במחלקה Student עובר בהורשה למחלקה Milgay, למעט בנאים.

```
public class Milgay extends Student {
    private int milga;
```

```
    public Milgay(int id, String name, int  milga){
        super(id, name);
        this.milga= milga;
    }

    // getters and setters
    public int getMilga() {
        return milga;
    }

    public void setMilga(int milga) {
        this.milga = milga;
    }

    public int computeTuitionFee(){
        return Math.max(0, super.computeTuitionFee()-
        milga);
    }
}
```

פנייה לבנאי של מחלקת האב. חייב להיות שורה ראשונה אם אין קריאה מפורשת לבנאי של האב, הקומפיילר מוסיף super() מעצמו. אם במחלקת האב אין בנאי ללא פרמטרים, נקבל שגיאת קומפילציה.

דריסה. דורסים את מה שקיבלנו בהורשה, עם שיטה עם חתימה זהה. משתמשים בשיטה מהאב כי אין גישה ישירה למשתנים.

מחלקה יורשת יכולה לגשת ישירות למשתנים ושיטות שאינם פרטיים, ואין גישה ישירה למשתנים ושיטות פרטיים.

protected modifier - כל המשתנים והשיטות שהוגדרו ב protected, ניתנים לגישה ישירה ממחלקת הבן, אבל לא ניתנים לגישה מבחוץ.

קצת שיקרו לנו. זה נגיש גם מה package ומה subclass

```
…
public class Student{
    private String name;
    private int id;
    protected numOfCourses;
    protected static final int COURSE_Price=1000;
…
public class Milgay extends Student {
    …
    public int computeTuitionFee(){
        return Math.max(0, numOfCourses*COURSE_PRICE-
        milga);
    }
}
```

ניתן לכתוב מחדש שיטות שקיבלנו בהורשה. זה נקרא **דריסה (overriding)**. לא ניתן לדרוס שיטות פרטיות. לא ניתן לדרוס שיטות סטטיות.
נרשום main בו נראה דוגמאות לשימוש במחלקות Student ו- Milgay ועבור כל שורת קוד ננסה להבין אם היא גורמת לשגיאת קומפילציה, או שגיאת זמן ריצה, או שהיא תקינה.

```
public class TestInheritance{


    public static void main(String[] args) {
        Milgay m = new Milgay(123456789, "Moshe", 1000);
        //ID, name, milga
        m.setNumOfCourses(2);
        System.out.println("The milga of " + m.getName() +
        " is " + m.getMilga());
        // executes the computeTuitionFee of Milgay
        System.out.println("The TF of " + m.getName() + "
        is " +  m.computeTuitionFee());

        Student s1 = new Milgay(11,"Dani", 500);
        s1.setNumOfCourses(2);
        // executes the computeTuitionFee of Milgay
        System.out.println("The TF of " + s1.getName()+ "
        is " + s1.computeTuitionFee());


        System.out.println("The milga of s1 is "+
        s1.getMilga());

        //casting
        System.out.println("The milga of "+ s1.getName()+
        " is "+((Milgay) s1).getMilga());

        Student s2 = new Student(11,"Dani");
```

תקין

עובר קומפילציה. אפשר להגדיר מצביע יותר כללי ולאתחל עם משהו יותר ספציפי. תקין כי הכרזנו על זה כStudent

השיטה שתתבצע בפועל נבחרת ע"פ הטיפוס שאיתו אתחלנו (Milgay). זהו עקרון **הפולימורפיזם** - שיטה שמבצעת לפי טיפוס האובייקט בפועל

שגיאת קומפילציה. הקומפיילר בודק לפי הטיפוס שהכרזנו, אז הוא לא מכיר שיש שיטה כזאת. בדוגמא הקודמת היתה שיטה בעלת אותה חתימה ב Student (דרסנו אותה ב Milgay), אז זה עבר.

מבטיחים לקומפיילר שהוא באמת ימצא שיטה כזאת

מבטיחים לקומפיילר שהוא באמת ימצא שיטה בזאת

```
System.out.println("The milga of "+ s1.getName()+
" is "+((Milgay) s1).getMilga());

Student s2 = new Student(11,"Dani");
s2.setNumOfCourses(2);
// run time error
System.out.println("The milga of (the non-milgay)
s2 is " + ((Milgay) s2).getMilga());

    }//main
}//class TestInheritance
```

לא קיימת שיטה כזאת ב Student

המחלקה Object, מחלקות אבסטרקטיות

09:15 27 December 2016

סיכום זה נכתב במהלך הרצאה של הקורס מבוא לתכנות,
בסמסטר סתיו תשע"ז (2016-2017) של הנדסת מערכות מידע.
הוא משקף את החומר שהועבר ע"פ הבנתי בלבד, ולא מהווה
תחליף לנוכחות בהרצאה.
בהצלחה,
יונתן מורג

בשיעור הזה:

- מחלקה Object
- מחלקות אבסטרקטיות

בג'אווה כל מחלקה יורשת ממחלקה Object (כאילו שהיינו כותבים public class MyClass extends Object).
במחלקה Object נמצאות השיטות הבאות:

```
boolean equals(Object obj)
//return true if this object is an alias of the obj
String toString()
//returns a string representation of this object
object clone()
//creates and returns a copy of this object
```

בודק שוויון מבחינת כתובות
זהות בדויה

יש נוספות אך אלו הנפוצות

נכתוב קטעי קוד ב main ולגבי כל שורה ננתח האם היא עוברת קומפילציה, האם יש שגיאת זמן ריצה ומה היא מבצעת:

```
Point p = new Point(2,3);
Object o1=p;
System.out.println(p.getX()); //2
System.out.println(o1.getX()); //compilation error
System.out.println((Point)o1.getX()); //2
Object o2 = new circle(p,5);
System.out.println((Point)o2.getX()); //runtime error

//var instanceof Type //returns true if var is of type Type

System.out.println(p instanceof Point); //true
System.out.println(o1 instanceof Point); //true
System.out.println(o2 instanceof Point); //false
System.out.println(p instanceof Circle); // false
p=null;
System.out.println(p instanceof Point); //false
```

תקין
תקין - Point הוא סוג של Object. לא צריך new בי p קיים.
תקין
שגיאת קומפילציה - קומפיילר לא מביר שיש ל Object שיטה getX
casting
תקין
שגיאת זמן ריצה - לא ניתן לעשות casting מ Circle ל Point

אם היינו שואלים Object, גם true
אם היינו שואלים Object, היה true

null לא שייך לשום טיפוס ולכן תמיד יחזיר false

בשבוע שעבר הגדרנו מחלקות PointSet ו- CircleSet, כאשר ההבדל היחיד היה בטיפוס.
המטרה שלנו כעת, היא ליצור מחלקה לעבודה עם קבוצת איברים, כאשר האיברים יכולים להיות מכל טיפוס שהוא.

```
public class ObjectSet {
    private Object[] elements;
    private int size;

    // construct a set of given capacity
    public ObjectSet(int capacity){
        elements = new Object[capacity];
        size = 0;
    }

    public void add(Object o){
        if (!contains(o) && size<elements.length){
            elements[size] = o;
            size = size + 1;
        }
    }//add

    public boolean contains(Object o){
        return indexOf(o) != -1;
    }//contains

    private int indexOf(Object o){
        int ans = -1;
```

```
        for (int i=0; (i<size) && (ans == -1); i=i+1){
            if (elements[i].equals(o)){
                ans = i;
            }
        }
        return ans;
    } //indexOf

    public void remove(Object o){
        int i = indexOf(o);
        if (i != -1){
            elements[i] = elements[size-1];
            elements[size-1] = null;
            size = size - 1;
        }
    }//remove
```

בכל מחלקה יש לדרוס את השיטה equals שקיבלנו בהורשה מ Object.
בתוך מחלקה Point :

```
public boolean equals(Point other){
    if(other!=null)
        return(this.x==other.x && this.y==other.y);
    else
        return false;
}
```

בעיה - לא דרסנו את השיטה הקיימת (החתימה שונה)

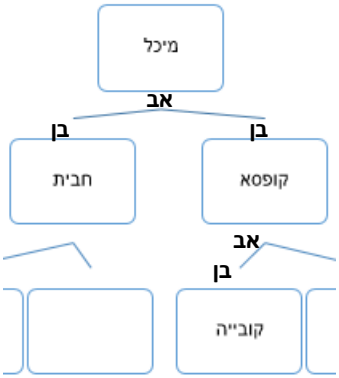
```
public boolean equals(Object other){
    if(other instanceof Point)
        return this.x==((point)other).x &&
        this.y==((Point)other).y;
    else
        return false;
}
```

Object ולא Point, כדי לדרוס במקום להעמיס.
עכשיו אנחנו לא בטוחים שבאמת קיבלנו Point

מחלקות אבסטרקטיות (abstract classes)

נרצה לכתוב קוד המנהל מיכלי דלק מצורות שונות: חבית, קופסא וקובייה. עבור כל מיכל אפשר:

- לשאול מה התכולה הנוכחית שלו
- לשאול מה הקיבולת המקסימלית שלו
- להוסיף דלק




```
public abstract class Tank{
```

ברגע שמחלקה הוגדרה כאבסטרקטית, לא ניתן להגדיר פרמטר מטיפוס כזה -
~~Tank t = new Tank();~~

```
    private double contents;
    public Tank(){
            contents = 0;
    }

    public abstract double capacity();

    public void fill(double amount){
        contents = Math.min(contents + amount,
        capacity());
    }

    public boolean isFull(){
        return contents == capacity();
    }

    public double getContents(){
            return contents;
    }
 } // class Tank
```

abstract -אנחנו לא כותבים פה את המימוש. הבנים יממשו.
אם לא היינו מכריזים על המחלקה כאבסטרקטית היה שגיאה

```
public class Cylinder extends Tank{
    private double r;
    private double height;

    public Cylinder(double radius, double height){
        super();
        this.r = r;
        this.height = height;
    }

    public double capacity(){
        return height * Math.PI * r * r;
    }
}
```

זה סגנון טוב לקרוא לבנאי האב גם אם צריכים בנאי() ממילא

אם לא היינו מספקים מימוש לcapacity הייתה שגיאת קומפילציה

```
public class Box extends Tank{
    private double length, width, height;

    public Box(double length, double width, double height)
    {
        super();
        this.length = length;
        this.width = width;
        this.height = height;
    }

    public double capacity(){
        return length * width * height;
    }
}
```

בתוך main

~~Tank t = new Tank;~~

Tank t = new Box(1,2,3);

Tank t1 = new Cylinder(3,8);

ממשקים (Interface)

09:14 02 January 2017

סיכום זה נכתב במהלך הרצאה של הקורס מבוא לתכנות,
בסמסטר סתיו תשע"ז (2016-2017) של הנדסת מערכות מידע.
הוא משקף את החומר שהועבר ע"פ הבנתי בלבד, ולא מהווה
תחליף לנוכחות בהרצאה.
בהצלחה,
יונתן מורג

בשיעור הזה:

- דוגמא מסובכת שאמורה להכיר לנו את נושא ממשקים

ניזכר במחלקה Student משבוע שעבר

```
public class Student{
     private String name;
     private int id;
     private numOfCourses;
     private static final int COURSE_PRICE=1000;
     …
```

מחלקה שתייצג מכונית

```
public class Car{
    private String company;
    private int model;

    public Car(int model, String company) {
         this.company = company;
         this.model = model;
    }

     public int getModel() {
         return model;
     }
     …
```

בתוך main...

```
Student[] arr = new Student[10];
```

נרצה למיין את המערך:

1. יש לבחור קריטריון מיון. לדוגמא, נמיין לפי ת"ז.
2. לבחור מיון - נקח מיון הכנסה (insertion sort)

```
public class InsertionSort{
     public static void insertionSort(Student[] arr) {

     for (int i = 1; i < arr.length; i = i + 1) {
          Student value = arr[i];
          int j = i;

          while (j > 0 && arr[j - 1].getID()>value.getID(){
               arr[j] = arr[j - 1];
               j = j - 1;
          }
          arr[j] = value;
          }
     }
}//insertion sort
```

בתוך main...

```
insertionSort(arr);
car[] carArr = new car[8];
insertionSort(carArr);
```

לא יעבוד! אין שיטה שיודעת למיין Car!
נכתוב קלאס של Insertion Sort שיוכל לעבוד עם כל טיפוס:

```
public class InsertionSort{
    public static void insertionSort(Object[] arr) {

    for (int i = 1; i < arr.length; i = i + 1) {
        Object value = arr[i];
        int j = i;

        while (j > 0 && arr[j - 1].???()>value.???(){
            arr[j] = arr[j - 1];
            j = j - 1;
        }
        arr[j] = value;
        }
    }
}//insertion sort
```

ל car ול Student יש שיטות שונות. מה נבחר?

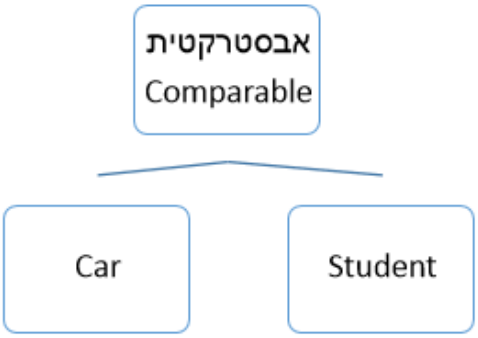
1. נגדיר מחלקה אבסטרקטית **Comparable** ובתוכה שיטה
Public int compareTo(Object other)
שמשווה אובייקט this ל other.
נחזיר מספר שלילי אם this<other.
נחזיר 0 אם הם שווים.
נחזיר מספר חיובי אם this>other.
2. המחלקות Student ו- Car ירחיבו את Comparable (extends).
3. נשנה את הקוד של המיון כך שנקבל מערך מטיפוס Comparable, וניתן לעקרון הפולימורפיזם לפתור את הבעיה.

```
public class InsertionSort{
    public static void insertionSort(Comparable[] arr) {

    for (int i = 1; i < arr.length; i = i + 1) {
        Comparable value = arr[i];
        int j = i;

        while (j > 0 && arr[j - 1].compareTo(value)>0{
            arr[j] = arr[j - 1];
            j = j - 1;
        }
        arr[j] = value;
        }
    }
}//insertion sort
```

יחפש ב Student או ב Car את השיטה (פולימורפיזם)

המחלקה האבסטרקטית:

```
public class Comparable {
    public  abstract  int compareTo(Object other);
}
```



ובחזרה ל Student:

```
public class Student extends Comparable{
    …
    public int compareTo(Object other){
        return id-((Student)other).id;
     }
    …
}

public class Car extends Comparable{
    …
    public int compareTo(Object other){
        return model-((Car)other).model;
     }
    …
```

ניתן לגשת ל other.id למרות שהיא private
כי השיטה נמצאת באותה מחלקה (Student)

```
}
```

בתוך main...

```
Comparable[] arr = new Student[10];
insertionSort(arr);
```

ממשק (Interface)

לא ניתן לבצע הורשה מרובה. לעומת זאת ניתן לממש ממשקים מרובים. נכתוב את comparable כממשק:

```
public interface Comparable {
    int compareTo(Object other);
}
```

הקומפיילר מוסיף אוטומטית public abstract כי לא ניתן להגדיר משהו אחר

כל מחלקה שמממשת את הממשק חייבת לממש את כל השיטות שלו. הדרך היחידה שהיא לא תממש את כולן היא אם היא אבסטרקטית.

```
public class Car implements Comparable{
    …
    public int compareTo(Object other){
        return model-((Car).other).model;
     }
    …
}
```

דוגמא לשימוש ביותר מממשק אחד

```
public class Student implements Comparable, Human{
    …
    public int compareTo(Object other){
        return id-((Student).other).id;
     }
    …
}
```

קודם עושים extends ורק אחרי זה implements.
הממשק Comparable קיים בג'אווה מראש.
לדוגמא, מחלקה String מממשת ממשק Comparable ולכן יכולנו להשוות בין המחרוזות בעזרת compareTo

```
str1.compareTo(str2)
```

ההשוואה היא לקסיקוגרפית - כמו במילון (ע"פ א"ב, ומילים קצרות יותר קודם).

נשנה שוב את Insertion Sort, הפעם כך שנוכל לשלוח קריטריון מיון ולמיין לפיו. לטובת זאת נכתוב ממשק נוסף, **Comparator** (גם הוא קיים כבר בג'אווה).
נניח ונרצה למיין פעם לפי id, פעם בסדר לקסיקוגרפי ופעם נוספת לפי מספר הקורסים.

```
public interface Comparator{
    int compare(Object o1, Object o2)
}
```

השיטה Compare מקבלת שני אובייקטים o1, o2 ומחזירה מספר שלילי אם o1<o2, מחזירה אפס אם o1=o2 ומספר חיובי אם o1>o2.

```
public class StudentComparatorByID implements Comparator{
    public int compare(Obj o1, Obj o2){
        int id1 = ((Student)o1).getID();
        int id2 = ((Student)o2).getID();
        return id1-id2;
    }
}

public class StudentComparatorByName implements Comparator{
```

```
    public int compare(Obj o1, Obj o2){
        String name1 = ((Student)o1).getName();
        String name2 = ((Student)o2).getName();
        return name1.compareTo(name2);
    }
}
```

בעזרת compareTo של מחלקה String

נכתוב את InsertionSort כך שהוא יקבל מערך וממיין וימיין את המערך על פיו:

```
public class InsertionSort{
    public static void insertionSort(Object[] arr,
    Comparator comp) {

    for (int i = 1; i < arr.length; i = i + 1) {
        Object value = arr[i];
        int j = i;

        while (j > 0 && comp.compare(arr[j-1], value)>0{
            arr[j] = arr[j - 1];
            j = j - 1;
        }
        arr[j] = value;
        }
    }
}//insertion sort
```

בתוך main...

```
Student[] arr = new Student[10];
Comparator comp1 = new StudentComparatorByID();
Comparator comp2 = new StudentComparatorByName();
insertionSort(arr, comp1);
insertionSort(arr, comp2);
```

ימיין לפי ת"ז

ימיין לפי שם

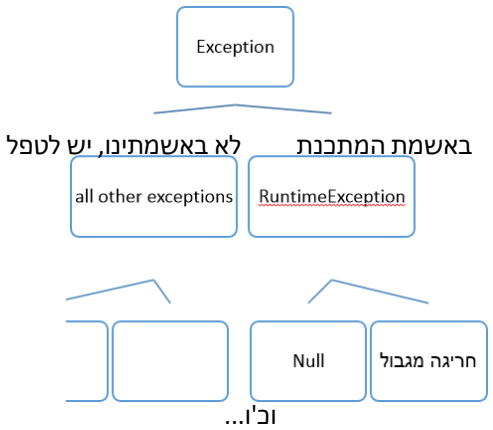
חריגות (Exceptions)

09:18 03 January 2017

סיכום זה נכתב במהלך הרצאה של הקורס מבוא לתכנות,
בסמסטר סתיו תשע"ז (2016-2017) של הנדסת מערכות מידע.
הוא משקף את החומר שהועבר ע"פ הבנתי בלבד, ולא מהווה
תחליף לנוכחות בהרצאה.
בהצלחה,
יונתן מורג

בשיעור הזה:

- מה הן חריגות (Exceptions)
- טיפול בחריגות



חריגה

אירוע המתרחש במהלך התוכנית שמשבש ריצה תקינה של התוכנית. לדוגמא:
IndexOutOfBoundException
NullPointerException

ב- Java יש מחלקות מוכנות של חריגות. המתכנת יכול להרחיב את המחלקות הקיימות
וליצור מחלקות שמתארות חריגות נוספות

```
public static void main(String[] args){
    int numerator = 10;
    int denominator = 0;
```

יהיה חריגה והתוכנית תעצור

```
    System.out.println(numerator/denominator);
    System.out.println("This line will not be printed");
```

נקבל חריגה מסוג:

```
Exception in thread "main" java.lang.ArithmaticException
/zero
at zero.main(Zero.java: 17)
```

שורה שבה קרתה החריגה

אחרי זה יבוא כל המסלול של איך הגענו לשורה שבה החריגה (מפונקציה א לפונקציה ב' וכ'ו)

טיפול בחריגות

ישנן שלוש דרכים להתייחס לחריגות:

1. לא לטפל - התוכנית תעצור (זה מה שעשינו עד עכשיו).
2. לטפל בחריגה באותו מקום בו היא התרחשה.
3. לטפל בחריגה במקום אחר בתוכנית.

<u>טיפול בחריגה במקום בו היא התרחשה</u>

נעטוף את קטע הקוד בו עשויה להתרחש החריגה ב**בלוק try-catch**.

```
try{
    …
} catch(exception e){
    …
}
```

ניישם:

```
public static void main(String[] args){
    int numerator = 10;
    int denominator = 0;

    try{
        System.out.println(numerator/denominator);
        System.out.println("This line will not be
```

```
        printed");
    }catch(ArithmaticException e){
        System.out.println("Division by zero");
    }
```

e הוא כמו אובייקט, שמייצג חריגה

ניתן לרשום e.getMessage() והוא יחזיר מחרוזת שמתארת את החריגה. אפשר להדפיס אותה, בדומה למה שיקרה אם לא נטפל בחריגה, רק שהפעם התוכנית תמשיך. השורה של "This line will not be printed" עדיין לא תתבצע, כי מ try קופצים ישירות ל catch.

אפשר להוסיף catch שונים כדי לטפל בחריגות שונות:

```
public static void main(String[] args){
    int numerator = 10;
    int denominator = 0;

    try{
        System.out.println(numerator/denominator);
        System.out.println("This line will not be
        printed");
    }catch(ArithmaticException e){
        System.out.println("Division by zero");
    }catch(IndexOutOfBoundsException e1){
        …
    }
```

האם זה צריך להיות e אחר?

finally clause

אם התרחשה חריגה ולא תפסנו אותה, התכנית תיזרק. נוסיף finally אחרי הבלוק של הcatchים. אם יש חריגה היא תטופל ע"י ה catch המתאים, או ע"י אף אחד מהם אם לא קיים מתאים. כך או כך, אם התרחשה שגיאה, ה finally יתבצע.

```
public static void main(String[] args){
    int numerator = 10;
    int denominator = 0;

    try{
        System.out.println(numerator/denominator);
        System.out.println("This line will not be
        printed");
    }catch(Exception e1){
        …
    }catch(Exception e2){
        …
    }finally{
        …
    }
```

אף פעם לא ייכנס לשני כי לא משנה מה החריגה, היא תתפס כבר בראשון

"להעביר את החריגה הלאה" (exception propogation)

טיפול בחריגה במקום אחר בתוכנית.

```
public static divide(int numerator, int denominator) throws
ArithmaticException{
    return numerator/denominator;
}

public static void main(String[] args){
    int numerator = 10;
    int denominator = 0;

    try{
        divide(numenator, denominator);
```

הפונקציה מזהירה שהיא עלולה לזרוק חריגה מסוג זה, ומי שמשתמש בה יצטרך לטפל בה בהתאם

```
    }catch(ArithmaticException e){
        System.out.println(e.getMessage());
    }
}

public static void anotherExceptionExample() throws
RuntimeException{
    throw new RuntimeException("Throw exception example");
}

public static void main(String[] args){
    try{
        anotherExceptionExample();
        System.out.println("This line will not be
        printed");
    }catch(RuntimeException e){
        System.out.println(e.getMessage());
    }
}
```

אם זה Runtime ובניו אז לא חובה להשתמש בו בתוך בלוק try-catch, כי לא מחוייבים ע"י הקומפיילר לטפל בהם. אם זה חריגה שחייבים לטפל בה אז חייבים בלוק try-catch

יחזיר את ה String שכתבנו בפונקציה:
"Throw exception example"

Exception in thread "notes" java.lang.IndexOutOfBoundsException: endOfClass

משימת תכנון, מבני נתונים (התחלה)

09:15 09 January 2017

בשיעור הזה:

- משימת תכנון
- מבני נתונים (התחלה)

היום נדבר על איך לתכנן עבודת תכנות

נתון ממשק המגדיר פונקציה חד מקומית y=f(x)

```
public interface FunctionInterface(
    /** returns function's value at x */
    double valueAt(double x);
}
```

הלקוח מבקש ארבע מחלקות שמיישמות את הממשק

1. Linear Function - מייצגת פונקציה לניארית y=ax+b. בנאי מקבל a ו b בתור פרמטרים.
2. ConstantFunction - פונקציה y=c (בנאי מקבל c)
3. SinFunction - פונקציה מהצורה y=a*sin(f*x+p). a= amplitude, f = frequency, p= phase. (בנאי מקבל a,f,p)
4. CosinusFunction - y=a*cos(f*x+p)

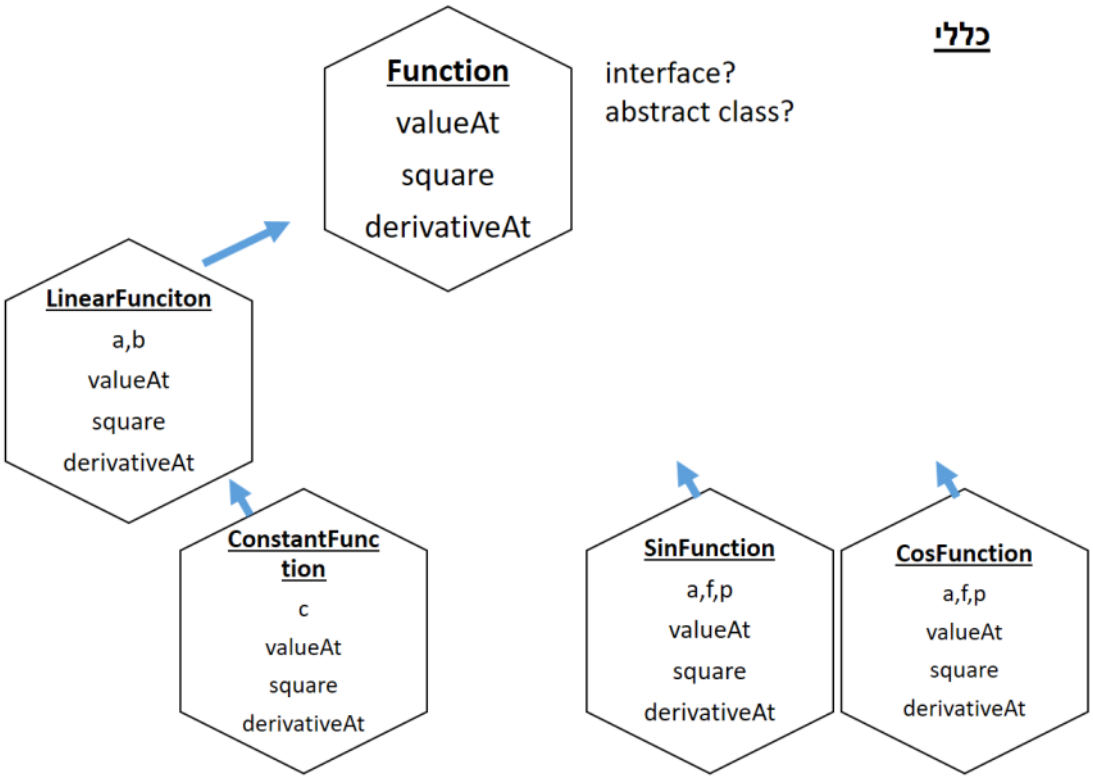
דרישות נוספות למימוש:

- square - מחזירה את הערך של הפונקציה בריבוע לדוגמא: y=3x+5, x=1 אז הפונקצי תחזיר 64.
- derivativeAt(x) - לחשב ערך של נגזרת הפונקציה בנק' x.
- toString

לפני שנתחיל נחשוב על התכנות:

1. האם מישהו מקרה פרטי של מישהו אחר?
2. האם כולם מקרה פרטי של מישהו?

אפשר לכתוב מחלקה אבסטרקטית שתכיל פעולות שיחזרו על עצמן באותה צורה באובייקטים שלנו, כמו square.



```
public abstract class Function implements
FunctionInterface{
     public abstract double valueAt(double x);
     public double square(double x){
          double ans = valueAt(x)
          return ans*ans;
     }

     protected abstract Function derivative();

     public double derivativeAt(double x){
          function der = derivative();
          return der.valueAt(x);
     }
}//function

public class LinearFunction extends Function{
     private double a;
     private double b;
     public Linearfunction(double a, double b){
          this.a=a;
          this.b=b;
     }
     public double valueAt(double c){
          return a*x+b;
     }
     protected function derivative(){
          return new ConstantFunction(a);
     }
}
```

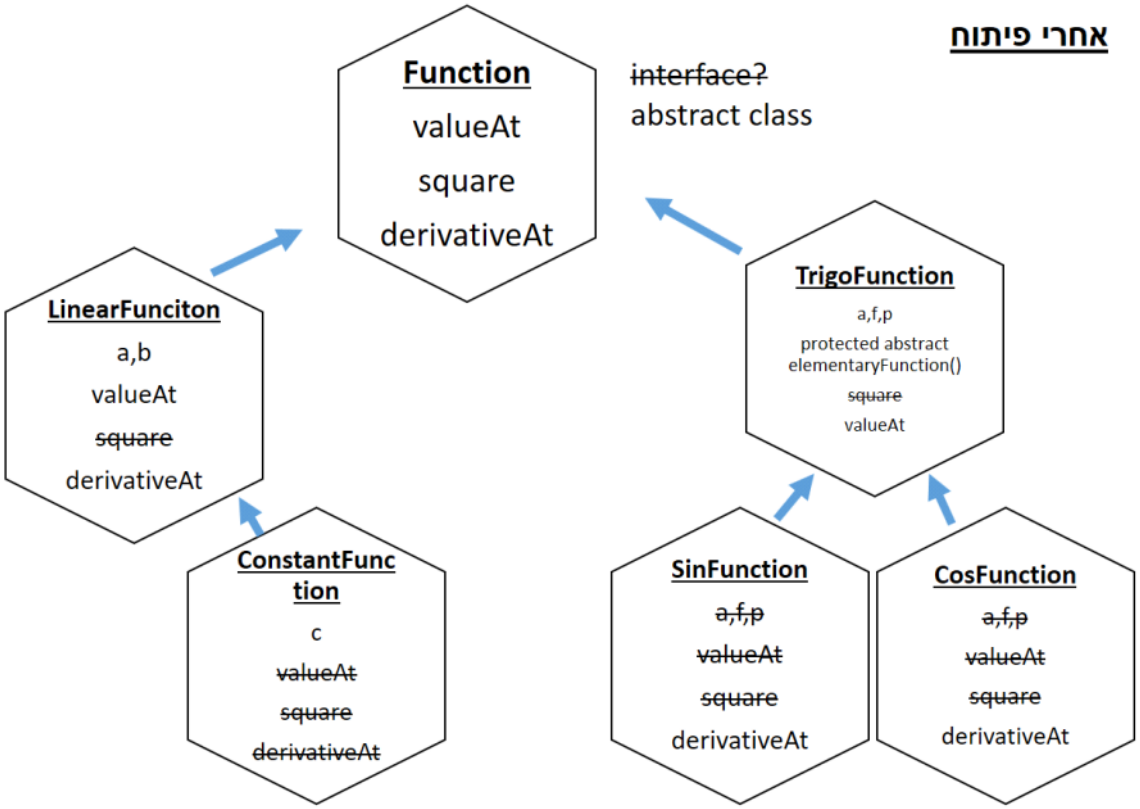
קומפיילר מוסיף קריאה לבנאי של super ושם יש בנאי שהקומפיילר הוסיף (כי לא כתבנו) שלא עושה כלום

```
public class ConstantFunction extends LinearFunction{
     public ConstantFunction(double c){
          super(0,c);
     }
}
```

לא צריך לכתוב כלום מחדש. פשוט נממש אותה עם הורשה ועם בנאי שמקבל (0,c)

```
Public abstract TrigoFunction extends Function{
     protected double a;
     protected double f;
     protected double p;
     public TrigoFunction(double a, double f, double p){
          this.a=a;
          this.f=f;
          this.p=p;
     }
     public double valueAt(double x){
          return a*elementryFunction(f*x+p);
     }
     protected abstract double elementryFunction(double x);
}//TrigoFunction

public class SinFunction extends TrigoFunction{
     public SinFunction(double a, double f, double p){
          super(a,f,p);
     }
     protected double elementryFunction(double x){
          return Math.sin(x);
     }
     protected Function derivative(){
          return new CosFunction(a*f, f ,p);
     }
}//SinFunction
```

```
public class CosFunction extends TrigoFunction{
    public cosFunction(double a, double f, double p){
        super(a,f,p);
    }
    protected double elementryFunction(double x){
        return Math.cos(x);
    }
    protected Function derivative(){
        return new SinFunction(-a*f, f ,p);
    }
}//CosFunction
```



מבני נתונים

נתונים => שמרנו במשתנים / מערך -> ביצענו פעולות על הנתונים.
המקום שמשמש לאכסון הנתונים נקרא מבנה נתונים. בהתאם למה שנרצה לעשות עם הנתונים, מתאימים מבני נתונים שונים.
מבנה נתונים - דרך לארגן נתונים במחשב. נהוג להגדיר מבני נתונים בשתי רמות:

1. מופשטת - מגדירים פעולות שרוצים לבצע (ללא מימוש). מבנה נתונים מופשט (abstract data type - ADT).
2. קונקרטית - מחליטים איך נארגן את הנתונים ונממש את הפעולות.

<u>קבוצה - ADT</u> - תכונות לדוגמא:
contains
isEmpty
add
remove

<u>קונקרטי</u> - תכונות לדוגמא:
מערך
באורך של בדיוק כמות האיברים
ממויין

מבני נתונים

09:13 10 January 2017

בשיעור הזה:

- מבני נתונים
- מחסנית (Stack)
- דוגמא לשימוש במחסנית

מבני נתונים

תזכורת/

נתונים -> שמרנו במשתנים או במערך -> ביצענו פעולות על הנתונים.

המקום שמשמש לאכסון הנתונים נקרא מבנה נתונים. בהתאם למה שנרצה לעשות עם הנתונים, מתאימים מבני נתונים שונים.

מבנה נתונים - דרך לארגן נתונים במחשב. נהוג להגדיר מבני נתונים בשתי רמות:

1. מופשטת - מגדירים פעולות שרוצים לבצע (ללא מימוש). מבנה נתונים מופשט (abstract data type - ADT).
2. קונקרטית - מחליטים איך נארגן את הנתונים ונממש את הפעולות.

נממש מבנה נתונים של מערך. מערך - רצף של איברים, כאשר לכל איבר מוצמד אינדקס:

```
public interface Array{
     public Object get(int i);
     public void set(Object data, int i);
     public int size();
}
```

נכתוב מחלקה המממשקת את הממשק Array שמגדירה מערך דינאמי - מערך שגודלו אינסופי

```
public class DynamicArray implements Array{
     private Object[] arr;

     public DynamicArray(){
          arr = new Object[0];
     }
     public Object get(int i){
          if(i<arr.length)
               return arr[i];
          else
               return null;
     }
     public void set(Object data, int i){
          ensureCapacity(i+1);
          arr[i]=data;
     }
     private void ensureCapacity(int capacity){
          Object[] tmpArr;

          if(capacity>arr.length){
               tmpArr = new Object[capacity];
               for(int j=0; j<arr.length; j++)
                    tmpArr[j]=arr[j];
               }
          }
          arr = tmpArr;
     }
     public int size(){
```

```
        arr = tmpArr;
    }
    public int size(){
        return Integer.MAX_VALUE;
    }
}//class
```

נרצה לייצג אינסוף. אין ערך כזה בג'אווה, אז נחזיר את אינט הגדול ביותר שיכול להיות בג'אווה

דוגמא לשימוש:

```
main...
Array arr = new DynamicArray();
    for(int i=1; i<=1000; i++)
        arr.set(new Integer(i), i);
```

לכל משתנה פרימיטיבי יש מחלקה עוטפת שאינה פרימיטיבית. נחוץ כי אנחנו צריכים Object לקוד שלנו

נשים לב שבכל פעם, אלף פעמים, הקוד ייצור מערך חדש. נרצה לכתוב קוד יעיל יותר. בכל פעם שצריך להגדיל את המערך הוא יגדל בעוד 10 מעבר לגודל הנחוץ.

```
public class DynamicArray implements Array{
    private Object[] arr;
    private static final int INCREMENT=10;

    public DynamicArray(){
        arr = new Object[INCREMENT];
    }
    public Object get(int i){
        if(i<arr.length)
            return arr[i];
        else
            return null;
    }
    public void set(Object data, int i){
        ensureCapacity(i+1);
        arr[i]=data;
    }
    private void ensureCapacity(int capacity){
        Object[] tmpArr;

        if(capacity>arr.length){
            tmpArr = new Object[capacity+INCREMENT];
            for(int j=0; j<arr.length; j++)
                tmpArr[j]=arr[j];
            }
        }
        arr = tmpArr;
    }
    public int size(){
        return Integer.MAX_VALUE;
    }
}//class
```

static - כל האובייקטים יוכלו לראות את אותו הערך באותו מקום בזיכרון

מחסנית (Stack)

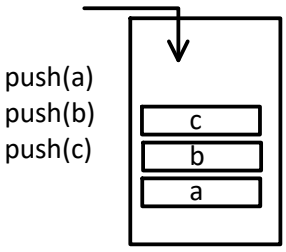
מחסנית - מבנה נתונים מופשט שתומך בפעולות הבאות:

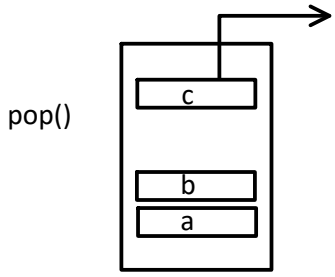
- isEmpty - בדיקה האם המחסנית ריקה
- Push - הכנסת איבר חדש למחסנית
- pop - מחיקת איבר אחרון שהוכנס למחסנית

```
public interface Stack{
    public boolean isEmpty();
```

מסירה מראש המחסנית ומחזירה ליוזר

```
    public void push(Object o);
    public Object pop();
}

public class StackAsArray implements Stack{
    private DynamicArray arr;
    private int index;

    public stackAsArray(){
        arr = new Dynamic Array();
        index = 0;
    }
    public boolean isEmpty(){
        return index==0
    }
    public void push(Object o){
        arr.set(o, index);
        index++;
    }
    public Object pop(){
        if(isEmpty())
            throw new RuntimeException("The stack is
            empty");
        index = index-1;
        Object res = arr.get(index);
        arr.set(null, index);
        return res;
    }
}
```





דוגמא לשימוש במחסנית

נתונה מחרוזת המייצגת ביטוי בו מופיעים סוגריים. יש לבדוק האם ביטוי חוקי מבחינת הסוגריים (כל סוגר פותח נסגר ע"י סוגר מאותו סוג). ישנם 3 סוגים של סוגריים [], (), {}.

```
public static boolean checkBalanced(String atr){
    Stack s = new StackAsArray();
    for(int i=0; i<str.length; i++){
        char c = str.charAt(i);
        if(c=='(' || c=='[' || c=='{')
            s.push(new Character( c));
        else if(c==')' || c==']' || c=='}'){
            if(s.isEmpty() || !matches((Character)
            s.pop()).charValue(), c) )
                return false;
    }
    return s.isEmpty();
}
```

תור (Queue) ורשימה מקושרת (Linked List)

09:13 16 January 2017

בשיעור הזה:

- תור (Queue)
- רשימה מקושרת (Linked List) - הכרות וקוד לחולייה

תור (Queue)

תור (Queue) הוא מבנה נתונים מופשט המנוהל לפי שיטה "ראשון נכנס, ראשון יוצא" (FIFO) - בניגוד למחסנית בה האחרון שנכנס הוא הראשון שיוצא. (LIFO - last in first out)

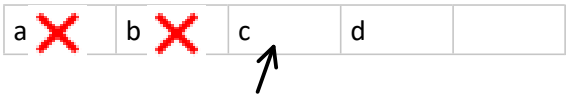
תור תומך בפעולות הבאות:

```
public interface Queue{
    public boolean isEmpty();
    public void enqueue(Object o);
    public Object dequeue();
}
```

מוסיף איבר לתור
מחזיר ומוחק את האיבר הכי ותיק.
נהוג להחזיר את האיבר שמוחקים
ולא רק למחוק אותו

מימוש 1:

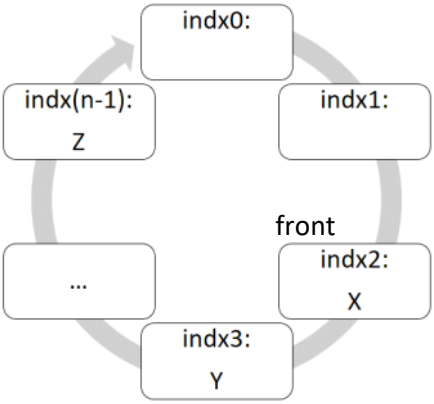
נשתמש במערך דינאמי: ההכנסה משמאל לימין.
משתני עזר: **front** - אינדקס של האיבר הכי ותיק, numOfElements - מספר האיברים בתור.
איבר חדש יכנס למקום front+numOfElements.
חסרון: לא ניתן לחזור לתאים שבתחילת המערך (front וגודל המערך כל הזמן יגדלו ויגדלו).



front - אינדקס של האיבר ה"ותיק ביותר"

מימוש 2:

נסה להתאים את גודל התור לכמות המקסימלית של איברים שיכולים להיות בו בו זמנית.
נשתמש במערך בגודל קבוע (n). נתייחס עליו כאל מבנה מעגלי בו חוזרים לאינדקס הראשון אחרי האחרון. בשיטה זו אנחנו מוגבלים בגודל התור.



נניח שכתבנו מחלקה של מערך בגודל קבוע

```
public class CircularQueue implements Queue{

    private Array arr;
    private int front, numOfElements, capacity;

    public CircularQueue(int capacity){
        this.capacity = capacity;
        arr = new FixedSizeArray(capacity);
        numOfElements = 0;
        front = 0;
    }

    public boolean isEmpty(){
        return numOfElements == 0;
    }

    public void enqueue(Object o){
        if (numOfElements == arr.size()){
```

```
                throw new RuntimeException(
                    "Queue is full!");
            }

            arr.set((front + numOfElements) %
                capacity, o);
            numOfElements = numOfElements+1;
        }

        public Object dequeue(){
            if (isEmpty()){
                throw new RuntimeException("Queue is
                empty!");
            }

            Object res = arr.get(front);
            arr.set(front, null);
            front = (front+1) % capacity;
            numOfElements = numOfElements-1;
            return res;
        }


    } //class CircularQueue
```

מימוש 3:

נממש תור בעזרת מחסנית. זה מימוש לא יעיל שלא באמת נשתמש בו בעתיד.

```
    public class QueueAsStack implements Queue{
        private Stack stack;

        public QueueAsStack () {
            stack = new StackAsArray();
        }
        public boolean isEmpty() {
            return stack.isEmpty();
        }
        public void enqueue(Object o) {
            stack.push(o);
        }

        public Object dequeue() {
            if (stack.isEmpty())
                throw new RuntimeException("Queue is
                empty!");

            Stack tmp = new StackAsArray();

            while(!stack.isEmpty())
                tmp.push(stack.pop());
            Object ret = auxStack.pop();
            while(!auxStack.isEmpty())
                stack.push(tmp.pop());

            return ret;
        }
    }//class QueueAsStack
```

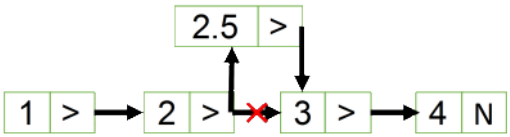
לא יעיל כי מעבירים איברים מפה לשם כל הזמן

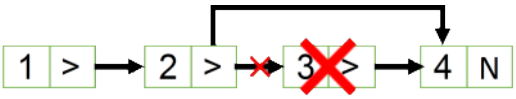
רשימה מקושרת

עד עכשיו איחסון הנתונים היה מבוסס על מערכים. מערך - רצף של תאים בזיכרון.
יתרון - גישה ישירה (לאיזה תא שנבחר):
נניח ובזיכרון של המחשב, כתובת ההתחלה עבור מערך Int שלנו היא 1000. כל int לוקח 4 בייטים. כדי לגשת לתא arr[5] המחשב עשה 1000+4*5 ומצא את המקום הרלוונטי בזיכרון.
חסרון - אי אפשר להכניס איברים נוספים באמצע של המערך.

מה נעשה? נשמור ערכים בתפזורת. כל איבר יהיה חלק מזוג - הערך של האיבר והכתובת של האיבר הבא. זה בא על חשבון הגישה הישירה, שכן כעת צריך לעבור דרך כל האיברים שנמצאים לפני האיבר שאנחנו מחפשים, כדי למצוא אותו.
כל איבר ברשימה נקרא חולייה או צומת (link/node) והמבנה כולו נקרא **רשימה מקושרת (Linked List)**. רשימה מקושרת נתונה ע"י כתובת של האיבר הראשון.

```
public class Link{
     private Object data;
     private Link next;

     //constructors
     public Link(Object data, Link next){
          this.data=data;
          this.next=next;
     }
     public Link(Object data){
          this.(data,null);
     }

     public Object getData(){…}
     public Object setData(){…}
     public void setNext(Link next){
          this.next=next;
     }

     public void addNext(Object o){
          Link toAddLink = new Link(o,next);
          next = toAddLink;
     }

     public Object removeNext(){
          if(next==null)
               throw new RuntimeException("no object to
               remove");
          Object ans = next.getData();
          next = next.getNext();
          return ans;
     }
```





מה קורה לחולייה שמחקנו? בג'אווה יש garbage collector שמסתכל מדי פעם אם יש אובייקטים שאף אחד לא מצביע לכתובת שלהם. כשהוא מוצא כזה הוא מוחק אותו מהזיכרון.

מחלקה שתנהל את הרשימה המקושרת:

```
public class LinkedList{
     private Link first;

     public LinkedList() {
          first = null;
     }
```

```
public void addFirst(Object o) {
     Link newLink = new Link(o, first);
     first = newLink;
}
```

נמשיך מחר...

רשימה מקושרת/משורשרת

09:11 17 January 2017

בשיעור הזה:

- רשימה מקושרת - קוד של מחלקה

רשימה מקושרת

Link - מייצגת חולייה
LinkedList - מייצגת רשימה

```
public class LinkedList{
    private Link first;

    public LinkedList() {
        first = null;
    }

    public void addFirst(Object o) {
        Link newLink = new Link(o, first);
        first = newLink;
    }

    public Object removeFirst() {
        if (first==null)
            throw new NullPointerException("No object to
            remove");

        Object res = first.getData();
        first = first.getNext();
        return res;
    }

    public boolean contains(Object o) {
        boolean ans = false;
        Link tmp = first;
        while(!ans && tmp != null){
            if (tmp.getData().equals(o)){
                ans = true;
            }
            tmp = tmp.getNext();
        } // while
        return ans;
    }

    public String toString() {
        String ans = "List: ";
        if (first==null) ans = ans + "Empty List";
        else{
            Link tmp = first;
            while(tmp != null){
                ans = ans + tmp.toString()+ ", ";
                tmp = tmp.getNext();
            }
        }
        return ans;
    }
```

מייצר חולייה חדשה עם המידע הנתון וכתובת החולייה הראשונה. שם את החלוייה החדשה בראשונה ברשימה

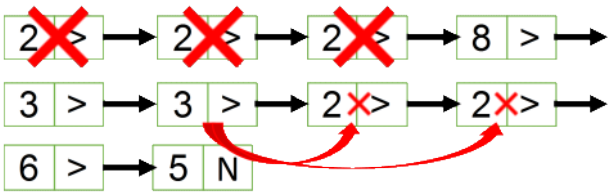
```
public class Link{
    …
    public String toString(){
        return ""+data;
```

בגלל שהתחלנו פה מחרוזת אז הוא ישתמש בtoString של data ויחזיר מחרוזת

נוסיף שיטה שבהינתן איבר מוחקת את כל המופעים שלו ברשימה

מוחק את כל המופעים של האיבר מתחילת הרשימה

מוחק את כל מהופעים של האיבר משאר הרשימה




```
public void removeAll(Object o) {
    while(first!=null && first.getData().equals(o))
        removeFirst();
    if (first!=null){
        Link tmp = first;
        while(tmp.getNext() != null){
            if(tmp.getNext().getData().equals(o)){
                tmp.removeNext();
            }
            else
                tmp = tmp.getNext();
        } // while
    } // if
}//removeAll
```

מייצר מחדש רשימה בסדר הפוך ואז הופך את first ברשימה המקורית לfirst בחדשה.
הרשימה המקורית ננטשת

```
public void reverse1() {
    LinkedList tmp = new LinkedList();
    for(Link i=first; i!=null; i=i.getNext()) {
        tmp.addFirst(i.getData());
    }
    first = tmp.first;
}
```

נחזיק שלושה דברים בזיכרון: קודם, נוכחי, והבא.
נרוץ על הרשימה מההתחלה שלה ובכל שלב נשנה את הכתובות כך שהשני יצביע לראשון, השלישי לשני, וכן הלאה.
לא מאותחל כי אם first==null יהיה שגיאת ריצה

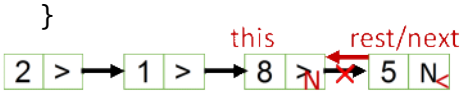

```
public void reverse2(){
    Link prev = null;
    Link curr = first;
    Link nextLink;
    while(curr != null){
        nextLink = curr.getNext();
        curr.setNext(prev);
        prev = curr;
        curr = nextLink;
    }
    first = prev;
}
```

נשתמש בטבע הרקורסיבי של הרשימה כדי לממש פעולות שונות.
בתוך מחלקת LinkedList נכתוב פונקציית מעטפת ובתוך מחלקה Link נכתוב פונקצייה רקורסיבית שמבצעת את העבודה.
נוסיף שיטה print להדפסת איברי הרשימה

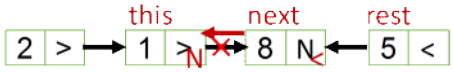
```
public class Link{
    …
    public void print(){
    System.out.println(data);
    if(next!=null)
        next.print();
    }

    public void printBackward(){
    if(next!=null)
        next.print();
    System.out.println(data);
    }

    public void reverse3(){
        if(next==null)
            return this;
        Link rest=next.reverse3();
        next.setNext(this);
        next = null;
```

```
public void print() {
    if(first != null)
        first.print();
}

public void printBackward(){
    if (first != null)
        first.printBackward();
}

public void reverse3() {
    if (first != null)
        first = first.reverse3();
}
```

```
        return this;                                    first = first.reverse3();
    Link rest=next.reverse3();   }
    next.setNext(this);
    next = null;
    return rest;
}
```



המחלקה Link נותנת שירות למחלקה LinkedList והמשתמש לא צריך לראות אותה. נגדיר את המחלקה Link בתוך המחלקה LinkedList ונגדיר אותה פרטית.




```
public class LinkedList{
    private class Link{ … } //Link
    …
} //LinkedList
```

איטרטורים, עצים בינאריים

23 January 2017 08:13

בשיעור הזה:

- איטרטורים (Iterator)
- עצים בינאריים (Binary Tree)

איטרטורים

יש צורך לעבור על איברי המבנה. נזכר בממשק שמייצג קבוצה:

```
public interface Set {
    public void add(Object o){
    public boolean contains(Object o){
    public void remove(Object o){
    public int size();
}
```

כותבים מחלקה שתיישם את הממשק:

בנאי מעתיק

```
public class mySet implements set{
    public MySet(Set other){ ???
    }
```

בעייה: אין שום דרך לקבל את האיברים של other, כי לא מוגדר איזה מבנה נתונים זה. java נותן פתרון כללי לבעיה ע"י שני ממשקים שתמיד הולכים ביחד: **Iterable**, **Iterator**.

לא נדבר עליו בשיעור הזה

```
public interface Iterator{
    public boolean hasNext();
    public Object next();
    public Object remove();
}

public interface Iterable{
    public Iterator iterator();
}

public interface Set extends Iterable {
    public void add(Object o){
    public boolean contains(Object o){
    public void remove(Object o){
    public int size();
}
```

גם מיישם Iterable

מייצר איטרטור שמתאים למחלקה שלנו

```
public class SetAsArray implements Set{
    private Array arr;
    private int size;
    …
    public Iterator iterator(){
        return new ArrayIterator(arr, size);
    }
}
```

איטרטור למערך

```
public class ArrayIterator implements Iterator {
    private Array arr;
    private int nextIndx, size;

    public ArrayIterator(Array arr, int size) {
        this.arr = arr;
        this.size = size;
        nextIndx = 0;
    }

    public boolean hasNext() {
        return nextIndx < size;
```

```
    }
    public Object next() {
        if (!hasNext())
            throw new RuntimeException("No more
            elements");
        Object ans = arr.get(nextIndx);
        nextIndx = nextIndx+1;
        return ans;
    }
}

public class MySet implements Set {
    …
    public MySet(Set other){
        Iterator iter = other.iterator();
        while(iter.hasNext())
            add(iter.next());
    }
}
```

דוגמא לשימוש באיטרטור - אנחנו לא חייבים לדעת באיזה מבנה נתונים other משתמש בשביל לכתוב בנאי שמעתיק לשלנו

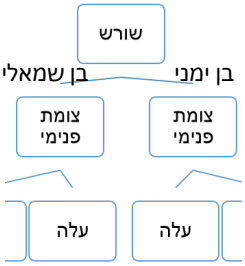
אפשר לקרוא לשיטה מקומית בלי this

מה עשינו?

1. Set הרחיב את Iterable.
2. כל מחלקה שמממשת Set חייבת לממש שיטה Iterator שמחזירה איזה אובייקט איטרטור (בחרנו איטרטור של מערך כי זה מה שאנחנו צריכים פה).
3. כתבנו מחלקה עם מימוש מתאים (במקרה זה, איטרטור על מערך).

עצים בינאריים

עץ הוא מבנה שמייצג היררכיה בין אובייקטים. עץ מורכב מקבוצה של **צמתים וצלעות**. הצומת הכי עליון נקראת **שורש** והכי תחתונים נקראים **עלים**. העץ מתאר **יחסי אב-בן**. צומת שאין לו אבא נקרא שורש וצומת שאין לו בנים נקרא עלה. יכולים להיות רק שני בנים לאב אחד (בעץ בינארי).



צומת X נקרא **צאצא** של צומת Y אם:
X הוא בן שמאלי או ימני של Y **או** X הוא צאצא של אחד הבנים של Y.

אם X הוא צאצא של Y, אז Y נקרא אב קדמון של X.

בדומה לשעשינו ברשימה מקושרת, נגדיר שתי מחלקות:

1. **BinaryNode** - מחלקה שמייצגת צומת.
2. **BinaryTree** - מחלקה שמייצגת עץ.

```
public class BinaryNode {

    protected Object data;
    protected BinaryNode left;
    protected BinaryNode right;

    public BinaryNode(Object data) {
        if (data == null)
            throw new NullPointerException();
        this.data = data;
        left = null;
        right = null;
    }
```

```
public class BinaryTree {

    protected BinaryNode root;

    public BinaryTree(){
        root = null;
    }

    public boolean isEmpty(){
        return root == null;
    }
    public void insert(Object element) {

        if(isEmpty()) {
            root = new BinaryNode(element);
        }else
            root.insert(element);
```

```
    if(isEmpty()) {
        root = new BinaryNode(element);
    }else
        root.insert(element);
}
```

<-בתור דוגמא, נכתוב שיטה שיוצרת רנדומאלית בן בימין או בשמאל

```
public void insert(Object element) {
    if (Math.random() < 0.5) {
        if (left == null)
            left = new BinaryNode(element);
        else
            left.insert(element);
    }
    else {
        if (right == null)
            right = new BinaryNode(element);
        else
            right.insert(element);
    }
}
```

חיפוש איבר בעץ

```
public boolean contains(Object element) {
    if(isEmpty())
        return false;
    else
        return root.contains(element);
}
```

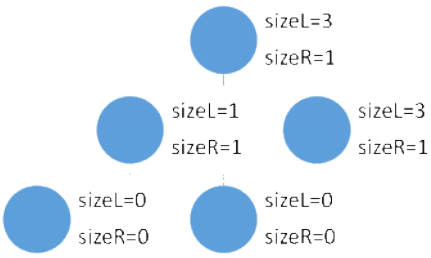
<- פונקציה רקורסיבית

הסבר ל else if - אם הוא לא מצא בענף שמאל הוא יכנס לחפש בימין ואם כן אז לא ינסה בכלל את ימין

```
public boolean contains(Object element) {
    boolean found = false;
    if(data.equals(element))
        found = true;
    else if(left != null && left.contains(element))
         found = true;
    else if(right!= null && right.contains(element))
         found = true;
    return found;
}
```

גודל העץ מוגדר להיות מספר הצמתים בעץ. גודל של עץ ריק מוגדר להיות 0.

```
public int size(){
    if(isEmpty())
        return 0;
    else
        return root.size();
}
```




```
public int size() {
    int sizeL = 0, sizeR = 0;
    if(left != null)
        sizeL = left.size();
    if(right != null)
        sizeR = right.size();
    return sizeL + sizeR + 1;
}
```

גובה הצומת מוגדר להיות המסלול הכי ארוך (של צלעות) מהצומת לעלה שצאצא שלו.
גובה העץ = גובה שורש העץ.
גובה של עץ ריק מוגדר להיות 1-.

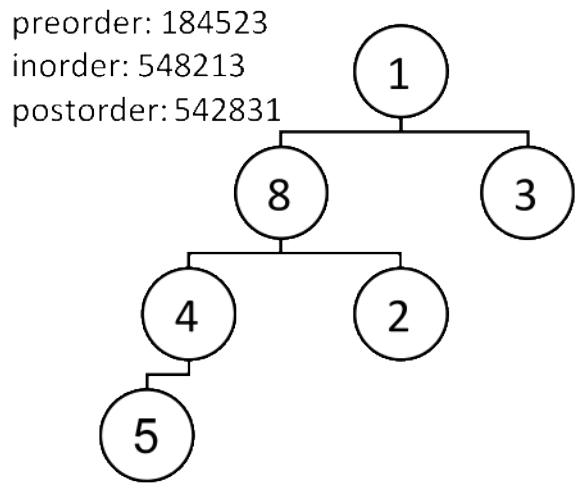
```
public int height(){
    if(isEmpty())
        return -1;
    else
        return root.height();
}
```

```
public int height() {
    int heightL = -1, heightR = -1;
    if(left != null)
        heightL = left.height();
    if(right != null)
        heightR = right.height();
    return Math.max(heightL, heightR) + 1;
}
```

עומק הצומת מוגדר להיות מספר הצלעות מהצומת לשורש

<u>סריקה</u> היא מעבר על צמתים של העץ בצורה שיטתית. שלוש שיטות:

preorder - **צומת**, שמאל, ימין. (תחילי)
inorder - שמאל, **צומת**, ימין. (תוכי)
postorder - שמאל, ימין, **צומת**. (סופי)




```
    public void preorder() {
        if(!isEmpty())
            root.preorder();
    }

    public void inorder() {
        if(!isEmpty())
            root.inorder();
    }

    public void postorder() {
        if(!isEmpty())
            root.postorder();
    }

} //class BinaryTree
```

```
    public void preorder() {
        System.out.println(data.toString());
        if(left != null)
            left.preorder();
        if(right != null)
            right.preorder();
    }

    public void inorder() {
        if(left != null)
            left.inorder();
        System.out.println(data.toString());
        if(right != null)
            right.inorder();
    }

    public void postorder() {
        if (left != null)
            left.postorder();
        if (right != null)
            right.postorder();
        System.out.println(data.toString());
    }

} //class BinaryNode
```

עץ חיפוש בינארי (BST)

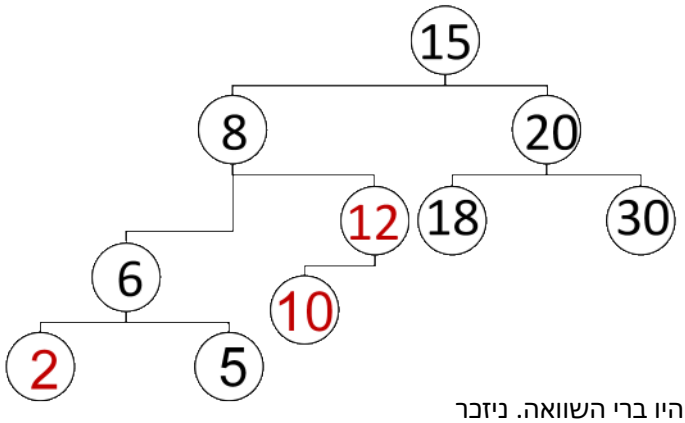
09:15 24 January 2017

בשיעור הזה:

- עץ חיפוש בינארי - Binary Search Tree (BST)

המשך לקוד מאתמול...

עץ חיפוש בינארי



עץ חיפוש בינארי הוא עץ בינארי המקיים את התכונה הבאה:

יהי: **X** - צומת בעץ

Y - צומת בתת עץ השמאלי של X

Z - צומת בתת עץ הימני של X

אז: **Y**.data<**X**.data<**Z**.data

על מנת להחזיק את האיברים באופן ממויין, יש לדאוג שהאיברים יהיו ברי השוואה. ניזכר בממשק Comparable:

```
public interface Comparable{
    public int compareTo(Object other);
}
```

נרחיב את המחלקות מהשיעור הקודם:

```
public class BSN extends BinaryNode {

    public BSN (Comparable data) {
        super(data);
    }

    public boolean contains(Object data) {
        if (!(data instanceof Comparable))
            return false;
        else
            return contains((Comparable) data);
    }

    private boolean contains(Comparable data) {
        int compareResult = this.data.compareTo(data);
        if (compareResult == 0) return true;
        else if (compareResult > 0){
            return
            (left != null && left.contains(data));
        }
        else return
            (right != null && right.contains(data));
    }

    public void insert(Object data) {
        if (!(datainstanceof Comparable))
            throw new RuntimeException("Only a Comparable
            data may be inserted!");
```

נדרוס את השיטה contains שירשנו עם שיטה שיעילה יותר עבור BST

פונקציה רקורסיבית שתעשה את העבודה

	תזכורת/
o1=o2	0
o1>o2	>0
o1<o2	<0

דריסה. מוודא שקיבלנו אובייקט שניתן להשוואה

```
        else
            insert((Comparable) data);
    }

    private void insert(Comparable data) {
        if (this.data.compareTo(data) > 0){
            if (left == null)
                left = new BSN (data);
            else
                left.insert(data);
        }
        else if (this.data.compareTo(data) < 0){
            if (right == null)
                right = new BSN (data);
            else
                right.insert(data);
        }
    }
}
```

נכניס משמאל או מימין ע"פ ההשוואה בין האובייקטים

העשרה(לא בחומר)/ אם היינו רוצים להוסיף שיטה שמוחקת:

יש שלושה מקרים:

1. צומת שרוצים למחוק הוא עלה.
2. לצומת שרוצים למחוק יש בן אחד.
3. לצומת שרוצים למחוק יש שני בנים.

שני המקרים הראשונים פשוטים. השלישי יותר מורכב. ניקח את המינימלי בענף ונשים אותו במקום הצומת שהיא אב של הצומת שאנחנו מוחקים.