קורס JAVA
מרצה: שאדי עסאקלה

מחלקה אנונימית (Anonymous class):

מחלקה אנונימית היא מחלקה מקומית , חסרת שם המשמשת ליצירת עצם בו זמנית עם הגדרת המחלקה.
לדוגמא:

```
public class firstClass {
	int x;
	public int func(int n)
	{
		x += n;
		System.out.println("firstClass - x is: \n"+ x);
		return x;
	}
	public void print()
	{
		System.out.println("firstClass: "+ x);

	}
	public firstClass(){}

}

public class lastClass extends firstClass{
	public int func(int n)
	{
		x -= n;
		System.out.println("lastClass-x: \n"+ x);
		return x;
	}
	public void print()
	{
		System.out.println("lastClass: "+ x);

	}
	public lastClass(){}

}

public class Main {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		firstClass st1 = new firstClass();
		firstClass st2 = new lastClass();
		firstClass st3 = new lastClass()
		{
			public int func(int n)
			{
				x = n*2;
				System.out.println("Main-x: \n"+ x);
				return x;
			}
```

```
        };

            //st1.func(st2.func(st3.func(st2.func(st1.func(3)))));
            st1.func(3);
            st2.func(4);
            st3.func(4);
            st1.print();
            st2.print();
            st3.print();
        }
}
```

פלט:

```
firstClass - x is:
3
lastClass-x:
-4
Main-x:
8
firstClass: 3
lastClass: -4
lastClass: 8
```

מחסנית:
היא מבנה נתונים המבוסס עקרון LIFO, מחלקה זו ממושת ב java.util.Stack
היא תומכת ואף מומלצת במקרים של אחזור נותנים באופן הפוך.
למשל, במשחקים שבהם השחקנים נכנסים ויוצאים בסדר מסוים, אלגוריתמים לבדיקות למשל בדיקת תקינות סוגריים. ועוד.

Method Summary	
boolean	**empty**() Tests if this stack is empty.
Object	**peek**() Looks at the object at the top of this stack without removing it from the stack.
Object	**pop**() Removes the object at the top of this stack and returns that object as the value of this function.
Object	**push**(Object item) Pushes an item onto the top of this stack.
int	**search**(Object o) Returns the 1-based position where an object is on this stack.

חשוב לזכור כי במחסנית הגישה היא אך ורק לאיבר שבראש המחסנית ולא ניתן לשלוף איבר שלא בראש המחסנית בפעולה אחת.
דוגמא:

```
import java.util.Stack;
public class Stack1 {

	public static void main(String[] args) {

		Stack s = new Stack();
		System.out.println("s.size() = " + s.size());

		s.push("Obj1");
		s.push("Obj2");
		System.out.println("s.pop() = " + s.pop());
		System.out.println("s.peek() = " + s.peek());
		System.out.println("s.pop() = " + s.pop());
	}
}
```

כאשר ממסים לבצע peek().pop() על מחסנית ריקה, נזרקת חריגה (יורחב בהמשך).

דוגמא לתוכנית שהופכת את המחסנית, (איבר שהיה בראש הופך להיות בתחתית המחסנית ולהיפך)

```
import java.util.Stack;
public class Stack1 {

	public static void main(String[] args) {

		Stack s = new Stack();

		s.push("z");
		s.push("y");
		s.push("x");
		System.out.println("the orignal stack is = " + s);
		reverse(s);
		System.out.println("the reversed stack is = " + s);
	}
	private static void reverse(Stack s)
	{
		Stack temp = new Stack(), temp2 =  new Stack();
		while(!s.empty())
			temp.push(s.pop());
		while(!temp.empty())
			temp2.push(temp.pop());
		while(!temp2.empty())
			s.push(temp2.pop());
	}
}
```

ArrayList

מבנה נתונים דמוי מערך הגדל בצורה דינמית בעת הצורך.

מספר מתודות חשובות:

Method Summary	
void	**add**(int index, Object element) Inserts the specified element at the specified position in this list.
boolean	**add**(Object o) Appends the specified element to the end of this list.
boolean	**addAll**(Collection c) Appends all of the elements in the specified Collection to the end of this list, in the order that they are returned by the specified Collection's Iterator.
boolean	**addAll**(int index, Collection c) Inserts all of the elements in the specified Collection into this list, starting at the specified position.
void	**clear**() Removes all of the elements from this list.
Object	**clone**() Returns a shallow copy of this ArrayList instance.
boolean	**contains**(Object elem) Returns true if this list contains the specified element.
void	**ensureCapacity**(int minCapacity) Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
Object	**get**(int index) Returns the element at the specified position in this list.
int	**indexOf**(Object elem) Searches for the first occurence of the given argument, testing for equality using the equals method.
boolean	**isEmpty**() Tests if this list has no elements.
int	**lastIndexOf**(Object elem) Returns the index of the last occurrence of the specified object in this list.
Object	**remove**(int index) Removes the element at the specified position in this list.
protected void	**removeRange**(int fromIndex, int toIndex) Removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.

Object	**set**(int index, Object element) Replaces the element at the specified position in this list with the specified element.
int	**size**() Returns the number of elements in this list.
Object[]	**toArray**() Returns an array containing all of the elements in this list in the correct order.
Object[]	**toArray**(Object[] a) Returns an array containing all of the elements in this list in the correct order; the runtime type of the returned array is that of the specified array.
void	**trimToSize**() Trims the capacity of this ArrayList instance to be the list's current size.

דוגמאות:

```
import java.util.*;

public class ArrayListExample {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		int counter = 0;
		ArrayList listA = new ArrayList();
		ArrayList listB = new ArrayList();
		for(int i=0;i<5;i++)
		{
			listA.add(new String(" " + i));
		}
		listA.add("Alex");
		listA.add("Jeff");
		listA.add("Alex");

		for(int i=0;i < listA.size();i++)
		{
			System.out.println("[" + i + "] - " + listA.get(i));
		}

		int locationIndex = listA.indexOf("Jeff");
		System.out.println("Location of Jeff is: " + locationIndex);
		System.out.println("First occuration of Alex  is: " +
listA.indexOf("Alex"));
		System.out.println("Last occuration of Alex  is: " +
listA.lastIndexOf("Alex"));

		System.out.println("Is List A empty?   " + listA.isEmpty());
		System.out.println("Is List B empty?   " + listB.isEmpty());
```

```
            System.out.println();

            listB  = new ArrayList();
            for(int i=0;i < listA.size();i++)
            {
                  listB.add(listA.get(i));
            }

            System.out.println("Are List's A and B equal?" + listA.equals(listB));

            Object[] objArray = listA.toArray();
            for(int i=0;i < objArray.length;i++)
            {
                  System.out.println("Array Element [" + i + "] - " + objArray[i]);
            }

            listA.clear();
            System.out.println("Is List A empty?   " + listA.isEmpty());
            System.out.println("Is List B empty?   " + listB.isEmpty());
            System.out.println();


      }

}
```

פלט

```
[0] -  0
[1] -  1
[2] -  2
[3] -  3
[4] -  4
[5] - Alex
[6] - Jeff
[7] - Alex
Location of Jeff is: 6
First occuration of Alex  is: 5
Last occuration of Alex  is: 7
Is List A empty?   false
Is List B empty?   true

Are List's A and B equal?true
Array Element [0] -  0
Array Element [1] -  1
Array Element [2] -  2
Array Element [3] -  3
Array Element [4] -  4
Array Element [5] - Alex
Array Element [6] - Jeff
Array Element [7] - Alex
Is List A empty?   true
Is List B empty?   false
```

הגדרת סוג מבני נתונים - ניתן להגדיר איזה סוג ספציפי של מבני נתונים שנרצה.

```
import java.util.ArrayList;
import java.util.Stack;
public class Main {
public static void main(String[] args) {

            Person p = new Person("Shadi",12345,'M');
            Person queen = new Person("Queen",33333,'F');
            String str="hello";


                  ArrayList list = new ArrayList();
                  list.add(p);
                  list.add(queen);
                  list.add(str);
                  while(!list.isEmpty())
                  {
                        if(list.get(0) instanceof Person)
                              System.out.println(((Person)
list.get(0)).getName());
                        else if(list.get(0) instanceof String)
                              System.out.println(list.get(0));
                        list.remove(0);
                  }



                  Stack s  = new Stack();
                  ArrayList <Person> l1 = new ArrayList();

                  s.push(new Person("Yossef",25,'M'));
                  l1.add(new Person("Yoni",30,'M'));
                  Object o1 = s.peek();

                  System.out.println(((Person) o1).getName());
                  System.out.println(l1.get(0).getName());
            }

}
```

פלט

```
Shadi
Queen
hello
Yossef
Yoni
```

תרגיל כיתה

כתוב מתודה public Object bottom שמקבלת מחסנית כפרמטר, ומחזירה אובייקט אחרון בתחתית המחסנית (לעומת peek). מתודה זו לא תהרוס את המחסנית, כלומר מצב המחסנית יהיה בדיוק כמו לפני ביצוע המתודה.

אוספים

- מבנה נתונים המאפשר לנו להחזיק כמות כלשהי של איברים בעלי מכנה משותף.
- המחלקה Vector היא סוף של אוסף שתפקידה לשמש כמערך שאינו מוגבל בגודלו
- במימוש האוספים ע"י JAVA ישנו שימוש במנגנון הנקרא generics שבו למעשה כותבים את המחלקה פעם אחת, ומעבירים כפרמטר ב- < > את הטיפוס של אוסף הנתונים
- הטיפוס חייב להיות שם של מחלקה ולא טיפוס בסיסי
- למשל: Integer ולא int
- נהוג גם לקרוא לאוסף בשם container

המחלקה Vector

מחלקת וקטור מיועדת למערך הגדל לפי הצורך.הפונקציות הבסיסיות של המחלקה הן :

addElement() – הוספת איבר למערך

elementAt() – קבלת איבר שנמצא במקום מסוים במערך

setElementAt() – הכנסת איבר למקום מסוים במערך

indexOf() – החזרת המקום שך איבר מסוים במערך (אם קיים)

removeAllElement() – איפוס הוקטור

יש לצרף בתחילת התוכנית: **import java.util.Vector;**

דוגמא:

```
import java.util.Vector;

public class Vec {

	public static void main(String[] args) {
	Vector <String> vec = new Vector<String>();
	vec.addElement("Sunday");
	vec.addElement("Monday");
	vec.addElement("Tuesday");
	System.out.println(vec); //// [Sunday,Monday,Tuesday]
	System.out.println(vec.elementAt(1));//Monday
	System.out.println(vec.indexOf("Monday")); //1

	System.out.println(vec.capacity()+ " " + vec.size()); //1

	vec.addElement("Sunday");
	vec.addElement("Monday");
	vec.addElement("Tuesday");
	vec.addElement("Sunday");
	vec.addElement("Monday");
	vec.addElement("Tuesday");
	vec.addElement("Sunday");
	vec.addElement("Monday");
	vec.addElement("Tuesday");
```

```
        System.out.println(vec.capacity()+ " " + vec.size());
}
}
[Sunday, Monday, Tuesday]
Monday
1
10 3
20 12
```

לשים לב שמגדיל את הגודל הפיזי ב 2 ותמיד מתחיל ב 10.

עוד דוגמא:

לשים לב שצריך מעטפת למשתנה בסיסי.

```
import java.util.Vector;

public class Vec {

    public static void main(String[] args) {
    Vector <Integer> vec = new Vector<Integer>();
    Integer a;
    vec.addElement(1);
    vec.addElement(2);
    vec.addElement(3);
    a = vec.elementAt(1);
    System.out.println(a);
}
}
```

מחלקת Hashtable

מייצגת טבלה, שמקשרת מפתח מסוים לערך מסוים אחר.הפונקציות הבסיסיות של המחלקה הן:

put()– הוספת מפתח וערך נוסף לטבלה

get()– קבלת ערך בעזרת מפתח

remove()– הסרת מפתח בטבלה

size()– קבלת מספר האיברים בטבלה

דוגמא:

```
Hashtable numerology = new Hashtable();
numerology.put("a", new Integer(1));
numerology.put("b", new Integer(2));
numerology.put("c", new Integer(3));
….
numerology.put("z", new Integer(26));
Integer letter;
letter = (Integer) numerology.get(x);
```

בנינו טבלה של ערכי האותיות בגימטרייה.

מאחר והטבלה מקבלת רק עצמים (לא משתנים) במקום להשתמש במשתנה מסוג int נעשה שימוש בעצם מסוג Integer שהיא המחלקה העוטפת את int (בהתאם יש מחלקות עוטפות גם לשאר סוגי המשתנים הבסיסיים)

יש לצרף למעלה: import java.util.Hashtable;

Java Exceptions

Exceptions הן תקלות שעלולות להתרחש בזמן ריצה. ניתן לטפל בהם בצורה מסודרת , הגיונית ולמנוע קריסה של התכנית . מנגנון החריגות מאפשר לכתוב קוד אלגנטי יותר, ובטוח יותר ע"י הגדרת אזורים מיוחדים לטיפול בבעיות חריגות.

<u>דוגמאות לבעיות שקורות בזמן ריצה:</u>

1. חלוקה ב- 0,
2. פנייה לקובץ לא קיים,
3. עבודה עם null,
4. חריגה מגבולות מערך וכו'.

ב- Java קיים מנגנון המטפל בחריגות בצורה מסודרת.

קיימות 2 סוגי שגיאות:

Error - שגיאה שלא ניתן להתגבר עליה והתכנית באה אל קיצה המר...

Exception - חריגה בה ניתן לטפל.

2 אבחנות אלה מגדירות 2 מחלקות אלו הנורשות ממחלקת Throwable קיימות חריגות תקניות רבות שהוגדרו עבורן מחלקות.

לכל מחלקה הנגזרת מ-Throwable יש בנאי בעל פרמטר אחד פרמטר מחרוזת המקבל את מחרוזת ההודעה (או פירוט) של החריגה. כל מחלקה הנגזרת מ- Throwable יש את השיטות הבאות:

String getMessage()

void printStackTrace()

String toString()

השיטה getMessage משיגה את ההודעה של החריגה, toString מדפיסה אותו, ו-printStackTrace שופך תמונת זיכרון של המחסנית (אמצעי דיבוג למומחים).

<u>דוגמאות למחלקות הנורשות מ- : Error</u>

OutOfMemory – כשלון בהקצאה זיכרון.

<u>דוגמאות למחלקות הנורשות מ- :Exception</u>

IOException חריגת קלט/פלט כללית (למשל קובץ לא קיים).

FileNotFoundException – נורשת מהקודמת.

הגדרה תחבירית של אופן הטיפול בחריגה:

```
Try
{
      //try to do something which may cause exception
}
catch(Some1Exception1 e1)
{
}
catch(Some1Exception2 e2)
{
}
```

דוגמא:

```
public class Example1 {
      public static void main(String[] args) {
            try
            {
                  throw new Exception("First Example");
            }
            catch(Exception e){
                  System.out.println("caught an Exception" + e.getMessage());
            }
```

הפונקציה יוצרת חריגה ותופסת אותה באותה הפונקציה.

הערות:

1. בלוק בו עלולה להתרחש חריגה נמצא בתוך:

```
try{
      //code
}
```

2. האזור בו מטופלת החריגה נמצא בתוך:

```
Catch{
}
```

כדי ליזום זריקה של חריגה יש ליצור אובייקט מסוג חריגה להשתמש בפקודה **throw** (נדון בה בהמשך).

דוגמא:

```
public class EvenException {
      void isEven(int number) throws Exception{
            if(number%2==0)
                  throw new Exception("Number is Even");
      }
}



public class Example2 {
      public static void main(String[] args) {
         EvenException ex = new EvenException();
         int number = 5;
         try{
             ex.isEven(number);
             //Only if the func "isEven(...)" didn't throw exception
```

```
            //the next line will done:
            System.out.println("If we are here, then the number is odd");
        }
        //If the func "isEven(...)" throw an exception, it is
        //catched by the next block:
         catch(Exception e){
            System.out.println("if we are here" + e.getMessage());
         }
    }
}
```

הערות:

1. כאשר פונקציה אינה רוצה לטפל בחריגה היא מצהירה שהיא זורקת חריגה.
2. שימו לב לזרימה של הפונקציה. אם לא מתרחשת חריגה, אזור הcatch אינו מתבצע. אם יש חריגה אזור ה catch מתבצע במקום האזור שאחרי החריגה.

מה קורה בזמן החריגה? ישנו דמיון רב בין פקודת return לבין זריקת Exception:

1. פעולת הפונקציה נעצרת.
2. בשני המקרים משתחרר הזיכרון של אותה הפונקציה.

בניית מחלקות של חריגות

על מנת לבנות Exception משמעותי, המתאים לתוכנית שאנחנו כותבים, ניתן לייצר Exception משלנו. במקרה זה יש לרשת מהמחלקה Exception. מקובל לתת למחלקה שם שמתאר את סוג ה Exception.

דוגמא:

```
public class MyException extends Exception {
      MyException(){}
      MyException(String msg)
      {
            super(msg);
      }
}

public class Example3 {

      void run()
      {
            Try
            {
                  throw new MyException("This is my ex");
            }
            catch(MyException e)
            {
                  System.out.println(e.getMessage());
            }
      }
      public static void main(String[] args) {
            Example3 test = new Example3();
            test.run();
```

```
        }
}
```

הערות:

1. מחכה לאובייקט מסוג Exception או למחלקה נורשת של Exception.

```
   void run()
   {
      try{
            throw new MyException("This is my ex");
      }
      catch(MyException e1){
            System.out.println("caught my exception 1");
      }
      catch(Exception e2){
            System.out.println("caught my exception 2");
      }
   }
```

תשובה: יודפס `caught my exception 1`

מסקנה: אזור הטיפול בחריגות עובד בדומה ל case ומטפל בחריגה הראשונה המתאימה.

המילה finally:

לפעמים נרצה לבצע קוד שיתרחש בכל מקרה אם הייתה חריגה או לא. במקרה זה נשתמש ב finally

דוגמא:

```
void run()
{
      try{
            throw new MyException("This is my ex");
      }
      catch(MyException e1){
            System.out.println("caught my exception 1");
      }
      catch(Exception e2){
            System.out.println("caught my exception 2");
      }
      finally{
            System.out.println("End of exception handling");
      }
}
```

מתי יש צורך אמיתי בfinally? נניח שבתוך הפונקציה פתחנו חלון שאותו נרצה לסגור בכל מקרה! את סגירת החלון נבצע באזור זה

דוגמא:

נתונה המחלקה הבאה:

```
abstract class Shape {
      int x,y;
      String name;

      Shape(int x,int y, String name)
      {
            this.x = x;
            this.y = y;
            this.name = name;
      }

      abstract void Show();
      int getSum()
      {
            return(x+y);
      }
      abstract int getTypicalNum();
      public String toString()
      {
            return "name =" + name + ",x =" +x+ ",y=" +y;
      }
}
```

א. יש להגדיר ולממש 2 מחלקות נוספות:

- Flow: שתכיל בנוסף לחברי המחלקה המוגדרים ב Shape את חבר המחלקה int num וערכם האופייני של אובייקטים של Flowיהיה num.
- Figure: שתכיל בנוסף לחברי המחלקה המוגדרים ב Shape את חברי המחלקה int w,h וערכם האופייני של האובייקטים של Figure יהיה w+h.

ב. יש להגדיר מחלקה Structure שתכיל מערך של אובייקטים של Flow ו Figure ובתוכה מתודה add להוספת אובייקט.

ג. יש להגדיר מחלקה לטיפול בחריגות בשם OutOfRangeException אשר מכילה מספר קבוע static final int MAX = 100. יש להוסיף לתוכנית יזום של החריגה כך שזה ימנע קיום של אובייקט במערך שערכו האופייני גדול מMAX.

```
public class Flow extends Shape {
      int num;
      Flow(int x, int y, int num, String name)
      {
            super(x,y,name);
            this.num = num;
      }
      void Show()
      {
            System.out.println("Flow name"+ name);
      }
      int getSum()
      {
```

```
            return(super.getSum() +num);
        }
        int getTypicalNum()
        {
            return num;
        }
        public String toString()
        {
            return super.toString() + "num = "+ num;
        }
    }



public class Figure extends Shape {

        int w,h;
        Figure(int x,int y,int w, int h, String name)
        {
            super(x,y,name);
            this.w = w;
            this.h = h;
        }
        void Show(){}
        int getSum()
        {
            return (super.getSum() +w +h);
        }
        int getTypicalNum()
        {
            return w+h;
        }
        public String toString()
        {
            return super.toString() + "w=" +w + ",h=" +h;
        }
    }

public class Structure {

        Shape[] shape;
        int cerrentNum; //the number of the object in the array
        public Structure(int max) {
            super();
            this.shape = new Shape[max];
            this.cerrentNum = 0;
        }

        void add(Shape x) throws OutOfRangeException
        {
            if(cerrentNum == shape.length)
                System.out.println("the array is full");
            else
                if(x.getTypicalNum() > OutOfRangeException.MAX)
                    throw new OutOfRangeException();
```

```
                else
                        shape[cerrentNum ++] =x;
        }


}public class OutOfRangeException extends Exception {
        static final int MAX = 100;
        OutOfRangeException()
        {
                super("you are out of range");
        }
}


public class Exercise1 {

        public static void main(String[] args) {
                Structure st = new Structure(5);
                Shape s1 = new Figure(2,3,4,5,"figure1");
                Shape s2 = new Flow(1,2,101,"flow1");

                try
                {
                        st.add(s1);
                        st.add(s2);
                }
                catch(OutOfRangeException e)
                {
                        System.out.println(e.getMessage());
                }
        }
}
```

דוגמא נוספת:

```
public class MyException extends Exception {
      MyException(){}
      MyException(String msg)
      {
            super(msg);
      }

}


public class Example3 {

    public static int division(int totalSum, int totalNumber)
    {
        int res = -1;
        System.out.println("Computing Division.");
        try
        {
                res = totalSum/totalNumber;

        } //try
        catch(ArithmeticException  e){
                System.out.println("ArithmeticException : "+ e.getMessage());
        }
        finally
        {
                if(res!= -1)
                {
                        System.out.println("Good...");
                        System.out.println("Result : " + res);
                } // if
                else
                {
                        System.out.println("Not Good");
                        return res;
                } // else

        } // finally
        return res;
    } // division

      void run()
      {
            try
            {
                  throw new MyException("This is my ex");
            }
            catch(MyException e)
            {
                  System.out.println(e.getMessage());
            }
      }
      public static void main(String[] args) {
```

```
            // TODO Auto-generated method stub
            Example3 test = new Example3();
            test.run();

            int result  = division(100,0);        // Line 2
        System.out.println("result : "+result);

//////////////////////////////////

        try{
            int a[] = new int[2];
            System.out.println("Access element three :" + a[3]);
         }catch(ArrayIndexOutOfBoundsException e){
            System.out.println("Exception thrown  :" + e);
         }



      }

}
```

output:

```
This is my ex
Computing Division.
ArithmeticException : / by zero
Not Good
result : -1
          Exception thrown  :java.lang.ArrayIndexOutOfBoundsException: 3
```

תרגיל:

1. הוסיפו לתרגיל הקודם של המחסנית טיפול ב Exception לידיעתכם בניסיון לבצע peek או pop על מחסנית ריקה נזרקת חריגה emptyStackException. הוסיפו את בלוק ה catch שידפיס הודעה מתאימה עבור חריגה זו ובלוק catch שיתפוס כול חריגה אחרת וידפיס הודעה שנתפסה חריגה.
2. בתרגיל זה נגריל מספרים שלמים לתוך מערך אך כל מספר חייב להיות זוגי.

נטפל בחריגות הבאות:

- מוגרל מספר שאינו זוגי – תודפס הודעת שגיאה מתאימה.
- הכנסת איבר מעבר לקיבולת המערך. – יוקצה מערך חדש בגודל כפול, תוכן המערך הקודם יועתק אליו ולאחר מכן האיבר הדש יוכנס למערך החדש.
- גישה לאיבר שלא בגבולות המערך – הדפסת הודעה מתאימה