מחלקות בתוך מחלקות - inner classes
חלק שני

דוגמה שימושית ל - inner class שאינו סטטי

בשיעור הקודם ראינו ש Java מאפשרת הגדרת מחלקה בתוך מחלקה כך שכל אובייקט מסוג המחלקה הפנימית מכיר בזמן ריצה את האובייקט מסוג המחלקה החיצונית שיצר אותו.
נשאלת השאלה מתי יש צורך במנגנון כזה.
דוגמה טובה לצורך כזה הוא איטרטור של מבנה נתונים. איטרטור הוא אובייקט המכיר מבנה נתונים ומספק שירותים שונים לעבודה עליו. השרות הבסיסי שמספק כל איטרטור הוא מעבר סידרתי על כל אברי מבני הנתונים.
נוסיף את הקוד הבא לדוגמת הרשימה המשורשרת מהשיעור הקודם:

```
class List {
 … all the previous code …
 public class Iterator {
        private Node _place=_head;
        boolean hasNext() { return _place!=null;}
        int next() {
           int ret = _place._data;
           _place = _place._next;
           return ret;
        }
   }
  public Iterator iterator() { return new Iterator(); }
}
```

למחלקת הרשימה הוספנו פונקציה ציבורית בשם iterator(), המחזירה אובייקט מסוג המחלקה הפנימית List.Iterator. List.Interator היא מחלקה פנימית, ציבורית ואיננה סטטית. העובדה שאיננה סטטית, מאפשרת לה להתייחס ל - member של אותו אובייקט שיצר אותה. בדוגמה זו יכולת זו מתבטאת רק בפניה ל - _head של List באתחול של List.Iterator.

ניתן שימוש אפשרי באיטרטור:

```
class Check {
   static public void main(String[] args) {
        List l = new List();
        int[] array = {2,7,1,3,9};
        for(int i=0; i<array.length; ++i)
           l.add(array[i]);
        List.Iterator it = l.iterator();
        while(it.hasNext())
           System.out.print(it.next()+",");
        System.out.println();
   }
}
```

פלט:

```
9,3,1,7,2,
```

כל קוד המשתמש בקינון מחלקות לא סטטי, אפשר לממש גם ע"י קינון סטטי ושליחה של המצביע לאובייקט היוצר. את האיטרטור שמימשנו עם מחלקה פנימית לא סטטית, ניתן היה לממש כך:

```
class List {
        … original code without the iterator …

public static class Iterator {
        private Node _place;
        private List _theList;
        boolean hasNext() { return _place!=null;}
        int next() {
           int ret = _place._data;
           _place = _place._next;
           return ret;
        }
        private Iterator(List theList) {
           _theList = theList;
           _place = _theList._head;
        }
   }
}
   public Iterator iterator() { return new Iterator(this); }
```

למעשה, בדוגמה הספציפית הזאת אין הכרח לשמור את המצביע לאובייקט היוצר ואפשר להסתפק בשליחת ה - head_ ואתחול ה - place_. אם מעונינים להוסיף לאטרטור פונקציונליות נוספת הדורשת גישה ל - members נוספים של הרשימה, שליחת ה - head_ כמובן לא תספיק.

Inner class המוגדר בתוך פונקציה

ב - Java יש אפשרות להגדיר מחלקה גם בתוך פונקציה (שיטה):

```
class A {
   void f() {
        class B {
           void foo() {System.out.println("B.foo(),  B in A.f() ");}
        }
        B b = new B();
        b.foo();
   }
   static public void main(String[] args) {
        A a = new A();
        a.f();
   }
}
```

מכיוון שהפונקציה f() בה מוגדרת המחלקה B איננה סטטית, המחלקה הפנימית B איננה סטטית והיא יכולה לגשת ל - members של A. אם f() הייתה פונקציה סטטית, אז B הייתה מחלקה פנימית סטטית ולא הייתה לה אפשרות לגשת ל - members של A.
המחלקה B יכולה לגשת גם למשתנים לוקליים **קבועים** של הפונקציה אבל לא למשתנים לוקאליים אחרים.

למרות שהמחלקה מוכרת רק בתוך הפונקציה, ניתן ליצור אובייקט מסוגה שיהיה רלוונטי גם מחוץ לפונקציה. בכדי לעשות זאת, הפונקציה צריכה להחזיר התייחסות יותר אבסטרקטית אליו. התייחסות כזאת יכולה להיות מחלקת אב או ממשק כמו בדוגמה הבאה:

```
class A {
   HasFoo f() {
         class B implements HasFoo {
            public void foo() { System.out.println("A.f().B.foo()");}
         }
         return new B();
   }

   static public void main(String[] args) {
         A a = new A();
         (a.f()).foo();
   }
}

interface HasFoo {
   public void foo();
}
```

דוגמה לשימוש ב class המוגדר בתוך פונקציה

לפעמים class מסוים מוגדר רק עבור אלגוריתם ספציפי ולא רוצים שיהיה מוכר ממקומות אחרים בתוכנית או ששמו יזהם את מרחב השמות. במקרים כאלה אפשר להשתמש ב - class במוגדר בתוך פונקציה.

הקוד הבא מדגים שימוש כזה עבור אלגוריתם של מעבר אקראי בגרף אקראי:

```
import java.util.*;

class A {
   static public void main(String[] args) {
         randomWalk();
   }
   static void randomWalk() {
         final int SIZE = 50;
         final int MAXDEG = 2;
         final int MAXPATH = 10;
         class Node {
            double _data;
            ArrayList _contacts;
            Node(double data) { _data = data; _contacts = new ArrayList();}
         }
         Node[] array = new Node[SIZE];
         for(int i=0; i<array.length; ++i) {
            array[i] = new Node(i);
            int n = (int)(Math.random()*(MAXDEG+1));
            for(int j=0; j<n; ++j)
                  array[i]._contacts.add(
                     new Integer((int)(Math.random()*SIZE))
                  );
         }
         for(int i=0; i<7; ++i) {
            Node p = array[(int)(Math.random()*SIZE)];
```

```
            int path=0;
            while(p._contacts.size() > 0 && path < MAXPATH) {
                System.out.print("( "+p._data+" )->");
                Object next =
                   p._contacts.get(
                                    (int)(Math.random()*p._contacts.size())
                                    );
                p = array[((Integer)next).intValue()];
                ++path;
            }
            System.out.println("|| length: "+path);
        }
    }
}
```

פלט אפשרי:

```
( 21.0 )->( 40.0 )->( 37.0 )->( 13.0 )->( 18.0 )->|| length: 5
( 36.0 )->( 3.0 )->( 19.0 )->|| length: 3
|| length: 0
( 13.0 )->( 32.0 )->|| length: 2
|| length: 0
( 11.0 )->( 40.0 )->( 37.0 )->( 13.0 )->( 32.0 )->|| length: 5
( 47.0 )->( 37.0 )->( 13.0 )->( 18.0 )->|| length: 4
```

כאן Node הוא class הנחוץ רק לצורכי האלגוריתם הזה ולכן הוא מוגדר כ - class פנימי לפונקציה.

סיבה נוספת להגדרה של class בתוך פונקציה היא רצון להחזיר אובייקט העונה על ממשק מסוים בלי להוסיף מחלקה חדשה למרחב הכללי. הדוגמה הסינתטית שהבאנו עם הממשק HasFoo מדגימה אפשרות כזאת.