מחלקות בתוך מחלקות - inner classes
חלק ראשון

Nested-Class או Static-Inner-Class

ב- Java ניתן להגדיר מחלקה בתוך מחלקה אחרת. לדוגמה:

```
class A {
   int _a=7;
   static class B {
         int _b=8;
   }
}
```

המשמעות של הגדרת המחלקה B בתוך A היא שהצורך במושג B עולה מתוך הקיום של A. ללא A, אין ל - B משמעות. מבחינת קוד הנמצא מחוץ ל - A , הוגדרו כאן למעשה המחלקות: A ו - A.B. לדוגמה:

```
class Check {
   static public void main(String[] args) {
         A a = new A();
         A.B b = new A.B();
         System.out.println(a._a+" "+b._b);
   }
}
```

שימו לב שהעובדה שהמחלקה B מוגדרת בתוך המחלקה A איננה אומרת שכל אובייקט מסוג A מחזיק אובייקט מסוג B - לא מדובר על הכלה של אובייקטים. לאובייקט a שב - main יש רק data member אחד - a_.

השימוש במילה השמורה static בהצהרת B נועדה להגיד שמדובר במחלקה נפרדת שניתן ליצור אובייקטים מסוגה ללא תלות בקיום אובייקטים מסוג A. בהמשך נראה הגדרות של inner classes ללא שימוש ב - static. כאשר מחלקה פנימית מוגדרת תוך שימוש במילה static, היא נקראת מחלקה מקוננת - Nested class.

למחלקות המוגדרות בתוך מחלקות, ניתן לתת הרשאות גישה כמו ל - members רגילים של המחלקה. לדוגמה:

```
class A {
   int _a=7;
   static private class B {
         int _b=8;
   }
}
```

אם ננסה כעת לקמפל את הקוד של Check, כל אזכור של המחלקה A.B יגרום לשגיאת קומפילציה. כאשר הגדרנו את המחלקה B כ - private אמרנו בעצם שרק קוד של A יוכל להתייחס לטיפוס B. מדובר בעצם על **אנקפסולציה של טיפוס**.

הנה דוגמה שימושית מאד:

```
class List {
   static private class Node {
         int _data;
         Node _next;
         Node(int data, Node next) { _data = data; _next = next; }
   }
```

```
   private Node _head = null;
   public void add(int data) { _head = new Node(data,_head); }
   public String toString() {
         String s = "";
         for(Node p=_head; p!=null; p = p._next)
            s+=p._data+"->";
         return s;
   }
   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]);
         System.out.println(l);
   }
}
```

בדוגמה זו הוגדר Node כ - inner class של List - **private**.

הסיבה להגדרתו כ - private היא **אנקפסולציה**. העובדה שקיים טיפוס בשם List.Node הוא פרט מימוש של List. לקוד חיצוני אין שום צורך לדעת שקיים טיפוס כזה וחבל שהוא יסתמך על כך. אם יום אחד נרצה לשנות את המימוש של List והשם Node כבר לא יתאר היטב את המהות בה אנחנו משתמשים, נוכל לשנות את השם ללא צורך לשנות את הקוד המשתמש ב - List. יתרה מכך, בהפיכת הטיפוס Node לפרטי אנחנו מונעים מהקוד המשתמש לבצע פעולות לא רצויות עם קודקוד של הרשימה. אלה בעצם שתי הסיבות הרגילות לאנקפסולציה והסתרת מימוש: מודולריות ושמירת עקביות באובייקט. אלה הסיבות גם לאנקפסולציה של משתנים ופונקציות.

הקוד הבא, לדוגמה, לא יעבור קומפילציה:

```
class Check {
   static public void main(String[] args) {
         List.Node p; // c.error: List.Node is private
   }
}
```

אם היינו משנים את הרשאת הגישה של Node ל - public, הקוד היה עובר קומפילציה.

אנקפסולציה של הטיפוס Node מאפשרת מתירנות לגבי הרשאות הגישה של ה - members שלו. מאחר ורק קוד של List ו - Node יכול להתייחס לטיפוס מסוג Node, אין צורך להגן על _data ו- _next משימוש ע"י קוד חיצוני. זאת הסיבה שלא הצהרנו עליהם כ - private.

ניתן לתת ל - inner class את כל ההרשאות הרגילות - public,protected,private ו - package-access. ההרשאות לטיפוסים פנימיים מתנהגות בדיוק כמו הרשאות למשתנים ופונקציות. אם לדוגמה יש לטיפוס פנימי הרשאת protected , ניתן יהיה להתייחס עליו גם מיורשים השייכים לחבילה אחרת אבל לא ממחלקות שאינן יורשות ונמצאות בחבילה אחרת.

ישנה סיבה חשובה נוספת להגדרת Node כ - inner class של List והיא **אי-זיהום של מרחב השמות**. השם Node יכול להיות שם נח גם לקודקוד של עץ בינארי. אם Node היה class רגיל, לא היינו יכולים לקרוא לקודקוד של עץ באותו השם, היינו נאלצים לבחור שם אחר. אם קודקוד של רשימה מוגדר כ - inner-class ב - List וקודקוד של עץ מוגדר להיות inner-class ב - Tree. ניתן לקרוא לשניהם בשם הקצר והנח - Node . בתוך הקוד של Tree אפשר יהיה לדבר על Node ואז יהיה ברור שמדובר בקודקוד של עץ. בתוך הקוד של List, יהיה ברור שמדובר בקודקוד של

רשימה. בקוד חיצוני יהיה צריך להתייחס על הקדקודים בשמם המלא: List.Node או Tree.Node - וזאת בתנאי שהרשאות הגישה לטיפוסים מאפשרות זאת. במצב זה, השם Node יכל עדיין לשמש לייצוג מושג אחר - אולי קודקוד כללי.
כאשר מגדירים מחלקה שהוצרך בה קיים רק בהקשר מחלקה אחרת, חבל לתפוס בשבילה שם שעשוי לשמש לצרכים אחרים.

הרשאות גישה מיוחדות ל - inner class

למרות שמחלקה פנימית היא מחלקה נפרדת מהמחלקה בתוכה היא מוגדרת, יש לשתי המחלקות הרשאות מיוחדות זו לזו. מחלקה פנימית יכולה לגשת לכל ה - private members של המחלקה החיצונית ולהפך. הקוד הבא מדגים זאת:

```
class A {
   private int _a=7;
   void f1() {
         B b = new B();
         b._b++;
   }
   static class B {
         private int _b=8;
         void f2() {
            A a = new A();
            a._a++;
         }
   }
}
```

מחלקות פנימיות שאינן סטטיות

עד כה יצרנו מחלקות פנימיות שהיוו בעצם מחלקות נפרדות לחלוטין. ה"פנימיות" של אותן מחלקות התבטא רק בשמן ובהרשאות הגישה אליהן. מכל בחינה אחרת הן היו שקולות למחלקות המוגדרות בנפרד. כל ההשלכות שהיו להגדרה הפנימית היו על תהליך הקומפילציה, בזמן ריצה, לא היה שום הבדל בין הגדרה פנימית להגדרה רגילה.
כל זה מכיוון שהגדרנו את המחלקה הפנימית כסטטית. אם נגדיר אותה כ לא-סטטית **ייווצר קשר** בזמן ריצה בין כל אובייקט מהמחלקה הפנימית לאובייקט מהמחלקה החיצונית **שיצר** אותו.
לדוגמה:

```
class A {
   private int _a=7;
   B getB() { return new B(); }
   class B {
         void foo() {
            System.out.println(_a);
         }
   }
}
```

כפי שניתן ליראות, הפונקציה foo() של המחלקה הפנימית, מתייחסת לשדה a_ השייך בכלל למחלקה החיצונית. מדובר למעשה בשדה a_ של אותו אובייקט מסוג A שדרכו האובייקט מסוג A.B נוצר. הנה דוגמה לשימוש:

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

```
        A a = new A();
        A.B b = a.getB();
        b.foo();
    }
}
```

אם היינו מנסים ליצור אובייקט מסוג A.B ללא אובייקט מסוג A הקוד לא היה עובר קומפילציה. לדוגמה:

```
class Check {
    static public void main(String[] args) {
        A.B b = new A.B(); // compilation error
    }
}
```

אם מעונינים ליצור אובייקט מסוג A.B באמצעות אובייקט קיים מסוג A, אפשר לעשות זאת ע"י שימוש בתחביר המיוחד הבא:

```
class Check {
    static public void main(String[] args) {
        A a = new A();
        A.B b = a.new B();
    }
}
```

מכיוון שיש צורך באובייקט מסוג A על מנת ליצור אובייקט מסוג A.B, גם הקוד הבא לא יעבור קומפילציה:

```
class A {
    static B getB() { return new B(); }
    class B {}
}
```

מכיוון שפונקציה סטטית איננה עובדת על אובייקט מסוים, אין לה this וחסר לה אובייקט דרכו היא יכולה ליצור את האובייקט מסוג A.B.

מכאן אולי ברורה ההחלטה של מעצבי השפה לבחור במילה השמורה static בשביל לתאר Nested class. מחלקה פנימית שאיננה מקבלת את ה - this של האובייקט שיצר אותה היא מחלקה פנימית סטטית. מחלקה פנימית רגילה (ללא שימוש ב - static) מקבלת בזמן יצירתה את ה - this של האובייקט שיצר אותה ושומרת אותו לשימושה.

כאמור, לכול אובייקט מסוג המחלקה הפנימית יש אפשרות לגשת לאובייקט שדרכו נוצר. השם המפורש של ה - reference (ה- pointer) לאותו אובייקט הוא שם המחלקה החיצונית ,נקודה ואז this. לדוגמה:

```
class A {
    private int _a=7;
    class B {
        void foo() {
            A p = A.this;
            System.out.println(p._a);
        }
    }
}
```

הפעלת foo() תגרום להדפסת המספר 7.