שימוש בקבצים חיצוניים (המשך)

בסוף השיעור הקודם הבאנו דוגמה לקוד המאפשר לקרוא עץ בינארי מקובץ. במבט חוזר על הקוד, ניתן לראות שתנאי העצירה של הרקורסיה מתבצע תמיד לפני שמגיעים ל - null. מכיוון שמצביעים ב - Java נשלחים תמיד by value, כל שינוי שלהם לא משפיע על הפונקציה הקוראת. בכדי לבצע שינוי שישפיע גם על הפונקציה הקוראת, יש לשנות את האובייקט המוצבע ע"י המצביע. הדבר גורם לקוד מסורבל במקצת. שימו לב שבגלל אותה הסיבה היינו צריכים לכתוב גם קוד מיוחד עבור הפונקציה שקוראת לפונקציה הרקורסיבית.

ניתן לכתוב קוד פשוט יותר, ע"י שימוש בערך המוחזר מהפונקציה:

```
public void load(FileReader fr) throws Exception {
        _root = load_rec(fr);
   }

   private Node load_rec(FileReader fr) throws Exception {
        char c = (char)fr.read();
        if(c=='n')
           return null;
        Node ret = new Node(readInt(fr));
        ret._ls = load_rec(fr);
        fr.read(); // ","
        ret._rs = load_rec(fr);
        fr.read(); // ")"
        return ret;
}
```

שימו לב שאם פורמט הקידוד היה פשוט יותר, ללא הפסיקים והסוגר הימני, ניתן היה לכתוב קוד קצר יותר.

```
private Node load_rec(FileReader fr) throws Exception {
        return new Node(readInt(fr),load_rec(fr),load_rec(fr));
}
```

כמובן, יש צורך להוסיף ctor מתאים ל - Node.

אפשרות אחרת להתגבר על הבעיה, היא לשלוח את המצביע אותו רוצים לשנות, עטוף באובייקט. לשם הפשטות, ניתן לעטוף אותו במערך בעל תא בודד:

```
public void load(FileReader fr) throws Exception {
        Node[] arr = new Node[1];
        load_rec(fr,arr);
        _root = arr[0];
   }

   private void load_rec(FileReader fr, Node[] arr) throws Exception {
        char c = (char)fr.read();
        if(c=='n') {
           arr[0] = null;
           return;
        }
        Node node = new Node(readInt(fr));
        load_rec(fr,arr);
        node._ls = arr[0];
        fr.read(); // ","
```

```
        load_rec(fr,arr);
        node._rs = arr[0];
        fr.read(); // ")"
        arr[0]=node;
    }
```

טכניקה זו חשובה כאשר צריך לשנות מספר מצביעים שנשלחים לפונקציה. בשפות תכנות אחרות כמו C++ או pascal יש תמיכה מובנת בשפה לשליחה של פרמטרים המשתנים ע"י הפונקציה הנקראת בצורה המשפיעה על הפונקציה הקוראת. לשליחת פרמטרים כזאת, קוראים לעיתים העברת פרמטרים by reference בניגוד להעברה by value.

בכל הגרסאות, פלט התוכנית יהיה זהה:

(2,(1,n,n),(6,(5,(4,(3,n,n),n),n),(7,n,n)))

serialization

עד כה, ראינו כיצד ניתן לקודד אובייקטים מורכבים, לשמור אותם בזיכרון הפרמננטי ולאחר מכן לשחזר אותם חזרה לזיכרון הראשי. הכלים בהם השתמשנו היו הכלים הבסיסיים של עבודה עם קבצים חיצוניים, את כל מלאכת הקידוד והפענוח, בצענו בעצמנו.

ב - Java קיים מנגנון מובנה בשפה המאפשר קידוד ופענוח אוטומטיים. שימוש במנגנון זה חוסך את הצורך לקבוע פורמט קידוד ואת הצורך לכתוב קוד מקודד וקוד מפענח. מכיוון שמדובר במנגנון כללי, הקידוד עלול להיות פחות קומפקטי מפורמט קידוד שהומצא במיוחד עבור סוג מסוים של אובייקט.

בכדי לאפשר קידוד אוטומטי של מחלקה, יש להצהיר אליה כניתנת לקידוד ע"י מימוש הממשק Serializable. ממשק זה איננו מכיל אף פונקציה והוא משמש אך ורק לתיוג (בדומה לתפקיד התיוג שיש ל - Cloneable). הנה דוגמה בסיסית לשימוש במנגנון:

```
import java.io.*;
class Check {
    public static void main(String[] args) throws Exception {
        A a = new A(7,17),a1;
        ObjectOutputStream
            oos = new ObjectOutputStream(new FileOutputStream("check.txt"));
        oos.writeObject(a);
        oos.close();
        ObjectInputStream
            ois = new ObjectInputStream(new FileInputStream("check.txt"));
        a1 = (A)ois.readObject();
        a1.print();
    }

}

class A implements Serializable {
    B _b1, _b2;
    public A(int d1, int d2) { _b1 = new B(d1); _b2 = new B(d2); }
    public void print() {
        System.out.println("<"+_b1._data+","+_b2._data+">");
    }
}
```

```
class B implements Serializable {
   int _data;
   B(int data) {_data = data;}
}
```

כפי שניתן לראות, בשביל לקודד אובייקט לתוך קובץ, יש לעטוף אובייקט מסוג FileOutputStream ע"י אובייקט מסוג ObjectOutputStream. הקידוד עצמו נעשה ע"י הפונקציה writeObject(); של ObjectOutputStream. הפענוח מבוצע באופן דומה ע"י הפונקציה readObject() של ObjectInputStream העוטף אובייקט מסוג FileInputStream. שימו לב שגם המחלקה A אותה מקודדים, וגם המחלקה B המהווה חלק ממנה, חייבות להיות מוצהרות כ - Serializable.

ניתן כעת דוגמה לשימוש במנגנון ה - serialization על מנת לשמור ולשחזר את העץ הבינארי שקודדנו קודם בצורה ידנית :

```
import java.io.*;

class Tree implements Serializable {
   static private class Node implements Serializable {
         int _data;
         Node _ls, _rs;
         Node(int data) {_data = data; _ls = _rs = null; }
   }
   Node _root=null;
   public void insert(int data) {
         if(_root==null) {
            _root = new Node(data);
            return;
         }
         Node p = _root;
         while(true) {
            if(data < p._data)
                  if (p._ls == null) {
                     p._ls = new Node(data);
                     return;
                  } else
                     p = p._ls;
            else
                  if (p._rs == null) {
                     p._rs = new Node(data);
                     return;
                  } else
                     p = p._rs;
         }
   }
   public String toString() {
         return toString(_root);
   }

   private String toString(Node p) {
         if(p == null)
            return "n";
         return "("+p._data+","+ toString(p._ls)+ ","+toString(p._rs)+")";
   }
```

```
   public void save(FileOutputStream fos) throws Exception {
        (new ObjectOutputStream(fos)).writeObject(this);
   }

   static public Tree load(FileInputStream fis) throws Exception {
        return (Tree)(new ObjectInputStream(fis)).readObject();
   }

   static private Tree createTree() throws Exception {
        int[] array = {2,6,1,5,4,7,3};
        Tree t = new Tree();
        for(int i=0; i<array.length; ++i)
           t.insert(array[i]);
        return t;
   }

   public static void main(String[] args) throws Exception {
        Tree t = createTree(), t1;
        FileOutputStream fos = new FileOutputStream("t.tree");
        t.save(fos);
        fos.close();
        FileInputStream fis = new FileInputStream("t.tree");
        t1 = load(fis);
        fis.close();
        System.out.println(t1);
   }
}
```

פלט:

```
(2,(1,n,n),(6,(5,(4,(3,n,n),n),n),(7,n,n)))
```

שימו לב שבקוד זה, לא היינו צריכים לדאוג כלל לקידוד ולפענוח. כל העבודה נעשתה בידי מנגנון ה - serialization.

מנגנון ה - serialization חייב להיות מסוגל להתמודד עם מעגלי הצבעות. מנגנון נאיבי העוקב אחרי ההצבעות של האובייקט ומקודד אותן, היה נקלע ללולאות אין סופיות. התוכנית הבאה בודקת התמודדות עם מעגל הצבעות פשוט:

```
import java.io.*;
class Check {
   public static void main(String[] args) throws Exception {
        A a = new A(7,17),a1;
        ObjectOutputStream
           oos = new ObjectOutputStream(new FileOutputStream("check.txt"));
        oos.writeObject(a);
        oos.close();
        ObjectInputStream
           ois = new ObjectInputStream(new FileInputStream("check.txt"));
        a1 = (A)ois.readObject();
        a1.print();
   }

}

class A implements Serializable {
   B _b1, _b2;
```

```
   public A(int d1, int d2) { _b1 = new B(d1,this); _b2 = new B(d2,this); }
   public void print() {
         System.out.println("<"+_b1._data+","+_b2._data+">");
   }
}

class B implements Serializable {
   A _a;
   int _data;
   B(int data, A a) {_data = data; _a=a; }
}
```

פלט:

<7,17>

תוכנית זו מדגימה שהמנגנון אכן ערוך לטפל במעגלי הצבעות.

דרישה נוספת של מנגנון ה - Serialization

בכדי שמנגנון הפענוח יעבוד, התוכנית צריכה לדעת מה המבנה המחלקה של האובייקט השמור בקובץ. האינפורמציה של המבנה נמצאת בקובץ ה - byte code (.class) של המחלקה ולכן קובץ זה חייב להיות נגיש לתוכנית.

לדוגמה, נניח שבתיקייה אחת נמקם את הקבצים הבאים:

file A.java:
```
import java.io.*;
class A implements Serializable {}
```

file Check.java:
```
import java.io.*;
class Check {
   public static void main(String[] args) throws Exception {
         ObjectOutputStream
            oos = new ObjectOutputStream(new FileOutputStream("A.txt"));
         oos.writeObject(new A());
         oos.close();
   }
}
```

אחרי שנקפל את הקבצים ונריץ את Check, ניצור תת תיקייה, ובה נמקם את הקובץ הבא:

file Check.java:
```
import java.io.*;
class Check {
   public static void main(String[] args) throws Exception {
         ObjectInputStream
            oos = new ObjectInputStream(new FileInputStream("../A.txt"));
         oos.readObject();
         oos.close();
   }
}
```

כאשר נקפל ונריץ את Check, בזמן ניסיון הקריאה של האובייקט, ייזרק ClassNotFoundException מכיוון ש - A.class לא נמצא ע"י התוכנית שרצה. אם נעתיק את A.class לתת התיקייה, התוכנית תעבוד.