מגדל של קוביות

פתרון

1. א. כתוב את המחלקה **קוביה Cube** המגדירה עצם מסוג קוביה שתכונותיו:
   - אורך צלע הקוביה side, מספר שלם
   - צבע הקוביה color, מחרוזת

   ב. כתוב במחלקה את כל הפעולות הדרושות לטיפול בקוביה:
   פעולה בונה, פעולות אחזור (get) לכל תכונה, הפעולה toString.
   בקוביה שניצור לא נשנה את התכונות, לכן לא נכתוב פעולות עדכון (set).

   ג. הוסף למחלקה פעולה בונה מעתיקה.

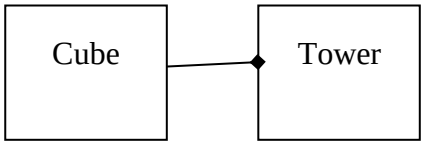
   ד. הוסף למחלקה את הפעולה equal המחזירה אמת אם הקוביה הנוכחית זהה בתכונותיה לקוביה שהתקבלה, ושקר אחרת.

```
/*
 *  Cube  -  המחלקה קוביה
 */
public class Cube
{
    //--- תכונות המחלקה ---
    private int side;        //  אורך הצלע
    private String color;    //  צבע הקוביה

    //--- פעולה בונה ---
    public Cube(int side, String color)

    //--- בנאי מעתיק ---
    public Cube(Cube cb)

    //--- פעולות אחזור לכל תכונה ---
    public String getColor()

    public int getSide()

    //--- לא נכתוב פעולות set כי הקוביה אינה יכולה לשנות את תכונותיה ---

    //--- פעולה המחזירה מחרוזת המתארת את מצב הקוביה ---
    public String toString()

    //--- פעולה המחזירה אמת אם שתי הקוביות זהות ושקר אחרת ---
    public boolean equals (Cube cb)
}
```

2. **מגדל של קוביות Tower** הינו אוסף של קוביות בגדלים וצבעים שונים.
האוסף מיוצג על ידי שרשרת חוליות מסוג <Node<Cube.

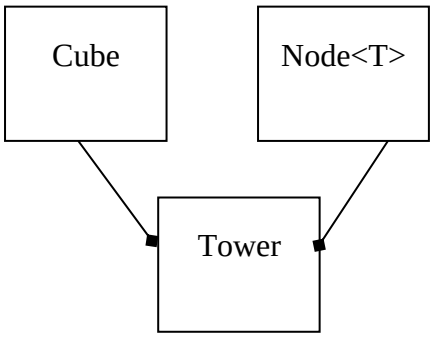
עבור מגדל של קוביות, נבחין בין מימוש המגדל באמצעות מערך למימוש באמצעות שרשרת חוליות.

המחלקות בפרויקט הממומש באמצעות מערך:



המחלקה Tower הינה מחלקה מורכבת,
כי היא מורכבת מעצמים מסוג קוביה - Cube

המחלקות בפרויקט הממומש באמצעות שרשרת חוליות:



המחלקה Tower הינה מחלקה מורכבת,
כי היא מורכבת מעצמים מסוג חוליה - <Node<T
וגם מסוג קוביה - Cube

התכנית הראשית תהיה זהה בשני הפרויקטים.
הרעיון - **הסתרת מידע** - הפרדה בין הממשק למימוש. לא חושפים את מבנה האוסף.
התכנית צריכה לעבוד נכון גם אם נחליף את המחלקה Tower שבה היא משתמשת למימוש אחר של האוסף.
ולכן התכנית תשתמש אך ורק בפעולות הממשק של המחלקות Tower ו- Cube.

המחלקה Tower הממומשת בשרשרת החוליות מכילה הפניה לחוליה הראשונה מסוג <Node<Cube, ותשתמש בפעולות המחלקה <Node<T .
המחלקה <Node<T הינה מחלקה כללית לכל סוגי השרשראות שנרצה לכתוב. (בתוך המחלקה מתייחסים למידע ולהפניות כ- T ו- <Node<T ופונים לתכונות כ- this.).
המחלקה Tower יוצרת שרשרת קונקרטית של קוביות (השרשרת היא מטיפוס <Node<Cube).

להלן מימוש חלקי של המחלקה Tower בשני המימושים.
שימו לב שאותה תכנית מריצה את שתי המחלקות.

כותרת הפעולה	תיאור הפעולה
Tower ()	פעולה בונה המחזירה מגדל שאין בו קוביות.
void add (Cube cb)	פעולה המוסיפה קובייה לראש מגדל. (ראש המגדל = החוליה הראשונה בשרשרת).
void insert (Cube cb)	פעולה המוסיפה קוביה למגדל קוביות הממוין לפי אורך הצלע, כך שהמגדל יישאר ממוין. (הקוביה הקטנה ביותר נמצאת בראש המגדל, בתחילת השרשרת, והקוביה הגדולה ביותר בתחתית המגדל, בסוף).
String toString()	פעולה המחזירה מחרוזת המתארת את מבנה המגדל.

```java
/*
 *         מגדל של קוביות - Tower המחלקה
 *              ממומשת במערך מצופף
 *       כך שכל הקוביות מרוכזות בתחילת המערך
 */
public class Tower
{
    //--- תכונות ---
    public static final int SIZE = 20;  //  גודל המגדל
    private Cube [] arr;     //  מערך הקוביות
    private int last;        // מספר הקוביות במגדל
                             //  וגם - מספר התא הפנוי הראשון במערך

    //--- פעולה בונה המחזירה מגדל ריק ---
    public Tower ()
    {
        this.arr = new Cube [SIZE];
        this.last = 0;
    }

    //--- פעולה המוסיפה קוביה לראש המגדל ---
    //      (ראש המגדל נמצא בתחילת השרשרת) ---
    public void add (Cube cb)
    {
        this.arr[this.last] = cb;
        this.last ++;
    }
    //--- פעולה המוסיפה קוביה למגדל קוביות הממויין לפי גודל הקוביה ---
    //--- בסיום הפעולה ישאר המגדל ממוין ---
    public void insert (Cube cb)
    {
        int i = 0;
        while (i < this.last &&  arr[i].getSide() < cb.getSide())
            i ++ ;

        //--- הזזת כל הקוביות מהמקום שנמצא ועד הקוביה האחרונה ---
        //--- מקום אחד ימינה כדי לפנות מקום לקוביה החדשה ---
        for (int j = this.last ; j > i ; j--)
            this.arr[j] = this.arr[j-1];

        //--- הכנסה למקום שהתפנה ---
        this.arr[i] = cb;

        //--- הגדלת last כי נוספה קוביה ---
        this.last ++;
    }

     //--- פעולה המחזירה מחרוזת המתארת את המגדל ---
     public String toString()
     {
         String str = "[";
         int i = 0;

         while (i < this.last)
         {
             str += arr[i].toString();
             if (i < this.last - 1)
                 str += ", ";
             i ++;
         }
         str += "]";
         return str;
     }
}
```

```java
/*
 *   המחלקה Tower - מגדל של קוביות
 *      ממומשת בשרשרת חוליות
 */
public class Tower
{
    //--- תכונה ---
    private Node<Cube> first;        //  הפניה לקוביה הראשונה במגדל

    //--- פעולה בונה המחזירה מגדל ריק ---
    public Tower ()
    {
        this.first = null;
    }

    //--- פעולה המוסיפה קוביה לראש המגדל ---
    //--- (ראש המגדל נמצא בתחילת השרשרת) ---
    public void add (Cube cb)
    {
        this.first = new Node<Cube>(cb, this.first);
    }
    //--- פעולה המוסיפה קוביה למגדל קוביות הממויין לפי גודל הקוביה ---
    //--- בסיום הפעולה ישאר המגדל ממוין ---
    public void insert (Cube cb)
    {
        Node<Cube> pos = this.first;     //  אסור להזיז את הראשון
        Node<Cube> prev = null;          // pos -הפנייה לחולה הקודמת ל
        while (pos != null &&  pos.getValue().getSide() < cb.getSide())
        {
            prev = pos;
            pos = pos.getNext();
        }

        //--- הכנסה להתחלה ---
        if (prev == null)
            this.first = new Node<Cube>(cb, this.first);     // this.add(cb);
        else
            prev.setNext(new Node<Cube>(cb, pos)); // prev הוספה אחרי
    }

    //--- פעולה המחזירה מחרוזת המתארת את המגדל ---
    public String toString()
    {
        String str = "[";
        Node<Cube> pos = this.first;

        while (pos != null)
        {
            str += pos.getValue().toString();
            if (pos.hasNext())
                str += ", ";
            pos = pos.getNext();
        }
        str += "]";
        return str;
    }
}
```

התכנית הבודקת את המחלקות:

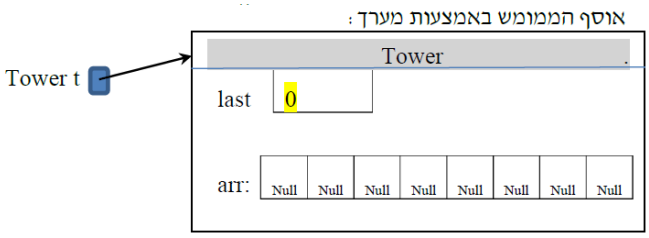
```
public class ChkTower
{
    /**
     * תכנית לבדיקת המחלקה מגדל של קוביות
     */
    public static void main(String[] args)
    {
        Tower tw1 = build();
        System.out.println("tw1 : " + tw1.toString());

        Tower tw2 = buildSorted();
        System.out.println("tw2 : " + tw2.toString());
    }

    //--- פעולה הבונה ומחזירה שרשרת של קוביות לא ממוינת ---
    public static Tower build()
    {
        Tower t = new Tower();
        t.add(new Cube (5, "red"));
        t.add(new Cube (2, "green"));
        t.add(new Cube (3, "yellow"));
        return t;
    }

    //--- פעולה הבונה ומחזירה שרשרת של קוביות ממוינת ---
    public static Tower buildSorted()
    {
        Tower t = new Tower();
        t.insert(new Cube (5, "red"));
        t.insert(new Cube (2, "green"));
        t.insert(new Cube (3, "yellow"));
        return t;
    }
}

/*

tw1 : [(3, yellow), (2, green), (5, red)]
tw2 : [(2, green), (3, yellow), (5, red)]

*/
```

נעקוב באמצעות תרשים עצמים אחרי הפעולה build בשני המימושים:

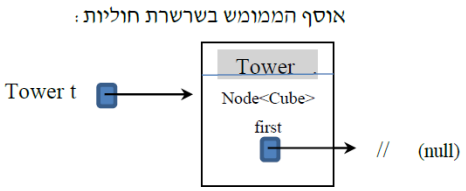
```
Tower t = new Tower();
```

יצירת האוסף:

אוסף הממומש באמצעות מערך:

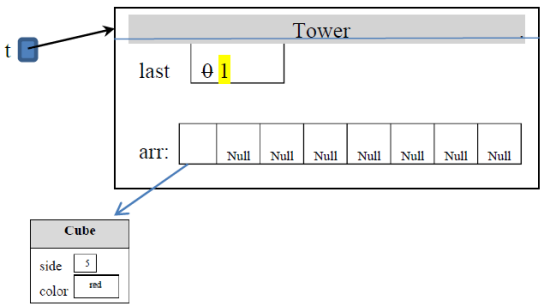


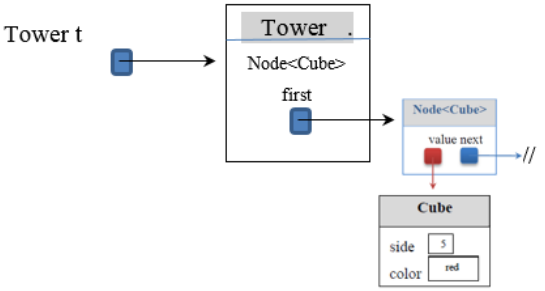
אוסף הממומש בשרשרת חוליות:



הוספת חוליה ראשונה לאוסף:

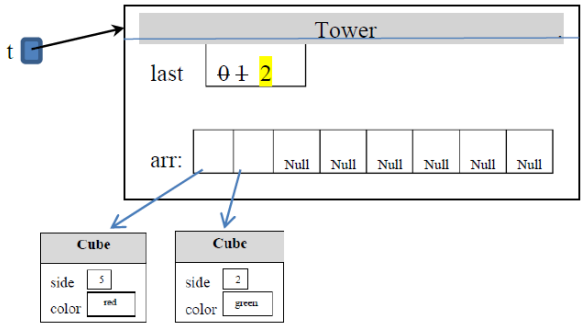
```
t.add(new Cube (5, "red"));
```

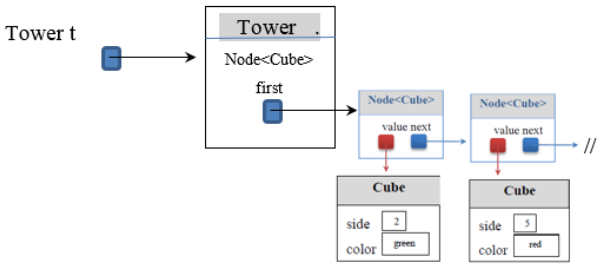




הוספת חוליה שנייה לאוסף:

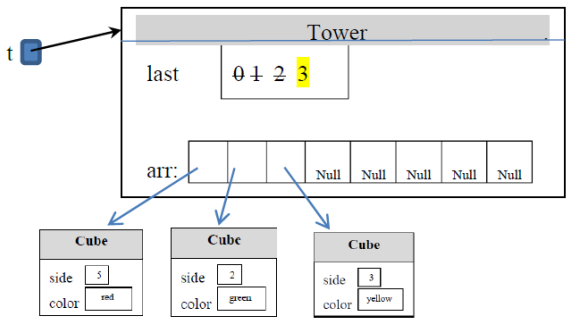
```
t.add(new Cube (2, "green"));
```

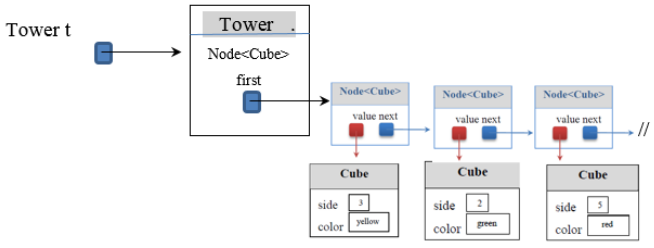




הוספת חוליה שלישית לאוסף:

```
t.add(new Cube (3, "yellow"));
```





במימוש מערך: arr[i] הינו הפנייה לעצם

במימוש שרשרת חוליות: pos הוא הפניה לחוליה שה- value שלה הוא הפנייה לעצם