Структури от Данни и Обектно-Ориентирано Програмиране
спец. Компютърни Науки
Упражнение № 8
14.04.2009г.

ТЕМА: Струкутура от данни МНОЖЕСТВО

Задача: Да се създаде клас **VectorSet,** който реализира интерфейса **Set** и представя множество, като за съхранение на елементите на множеството се използва данна тип **Vector.**

```
import java.util.Iterator;

public interface Set<E> extends Iterable<E> {
    /*
     * Adds the specified element to this set if it is not
     * already present (optional operation). More formally,
     * adds the specified element e to this set if the set contains
     * no element e2 such that (e==null ? e2==null : e.equals(e2)).
     * If this set already contains the element, the call leaves
     * the set unchanged and returns false.
     * In combination with the restriction on constructors,
     * this ensures that sets never contain duplicate elements.
     */
    boolean add(E e);

     /*
      * Removes all of the elements from this set (optional operation).
      * The set will be empty after this call returns.
      */
     void clear();

    /*
     * Returns true if this set contains the specified element.
     * More formally, returns true if and only if this set contains an element
     * e such that (o==null ? e==null : o.equals(e)).
     */
    boolean contains(E e);

    /*
     * Returns the difference of this set and the other set.
     */
    Set<E> difference(Set<E> other);
```

```
/*
 * Compares the specified object with this set for equality.
 * Returns true if the specified object is also a set, the two sets
 * have the same size, and every member of the specified set is contained in this set
 * (or equivalently, every member of this set is contained in the specified set).
 * This definition ensures that the equals method works properly across different
 * implementations of the set interface.
 */
boolean equals(Object o);

/*
 * Returns the intersection of this set and the other set.
 */
Set<E> intersection(Set<E> other);

/*
 * Returns true if this set contains no elements.
 */
boolean isEmpty();

/*
 * Returns true if this set is a subset of the other set.
 */
boolean isSubsetOf(Set<E> other);

/*
 * Returns an iterator over the elements in this set.
 * The elements are returned in no particular order
 * (unless this set is an instance of some class that provides a guarantee).
 */
Iterator<E> iterator();

/*
 * Removes the specified element from this set if it is present
 * (optional operation). More formally, removes an element e such
 * that (o==null ? e==null : o.equals(e)), if this set contains such an element.
 * Returns true if this set contained the element (or equivalently,
 * if this set changed as a result of the call). (This set will not contain the element
 * once the call returns.)
 */
boolean remove(E e);
```

```
    /*
     * Returns the number of elements in this set (its cardinality).
     * If this set contains more than Integer.MAX_VALUE elements,
     * returns Integer.MAX_VALUE
     */
    int size();

    /*
     * Returns the symmetric difference of this set and the other set.
     */
    Set<E> symDifference(Set<E> other);

    /*
     * Returns an array containing all of the elements in this set.
     * If this set makes any guarantees as to what order its elements
     * are returned by its iterator, this method must return the elements
     * in the same order.
     */
    Object[] toArray();

    /*
     * Returns an array containing all of the elements in this set;
     * the runtime type of the returned array is that of the specified array.
     * If the set fits in the specified array, it is returned therein.
     * Otherwise, a new array is allocated with the runtime type of
     * the specified array and the size of this set.
     */
    <T> T[] toArray(T[] a);

    /*
     * Returns the union of this set and the other set.
     */
    Set<E> union(Set<E> other);
}



import java.util.Vector;
import java.util.Iterator;

public class VectorSet<E> implements Set<E>, Cloneable {

    private Vector<E> body;
```

```
public VectorSet() {
        body = new Vector<E>();
}

public boolean add(E e) {
        if(body.contains(e))
                return false;
        return body.add(e);
}

 public void clear() {
         body.clear();
 }

 public Object clone() throws CloneNotSupportedException {
         VectorSet<E> result = (VectorSet<E>)super.clone();
         result.body = (Vector<E>)body.clone();
         return result;
 }

public boolean contains(E e) {
        return body.contains(e);
}

public Set<E> difference(Set<E> other) {
        VectorSet<E> result = new VectorSet<E>();
        E temp;
        Iterator<E> it = this.iterator();
        while(it.hasNext())
                if(!other.contains(temp = it.next()))
                        result.add(temp);
        return result;
}

public boolean equals(Object o) {
        if(o == null)
                return false;
        if(this == o)
                return true;
        if(!(o instanceof VectorSet))
                return false;
        VectorSet<E> other = (VectorSet<E>)o;
        return this.isSubsetOf(other) && other.isSubsetOf(this);
}
```

```
public Set<E> intersection(Set<E> other) {
        VectorSet<E> result = new VectorSet<E>();
        E temp;
        Iterator<E> it = this.iterator();
        while(it.hasNext())
                if(other.contains(temp = it.next()))
                        result.add(temp);
        return result;
}

public boolean isEmpty() {
        return body.isEmpty();
}

public boolean isSubsetOf(Set<E> other) {
        Iterator<E> it = this.iterator();
        while(it.hasNext())
                if(!other.contains(it.next()))
                        return false;
        return true;
}

public Iterator<E> iterator() {
        return body.iterator();
}

public boolean remove(E e) {
        return body.remove(e);
}

public int size() {
        return body.size();
}

public Set<E> symDifference(Set<E> other) {
        VectorSet<E> result = new VectorSet<E>();
        E temp;
        Iterator<E> it = this.iterator();
        while(it.hasNext())
                if(!other.contains(temp = it.next()))
                        result.add(temp);
        it = other.iterator();
        while(it.hasNext())
                if(!this.contains(temp = it.next()))
```

```
                    result.add(temp);
        return result;
    }

    public Object[] toArray() {
        Object[] array = new Object[body.size()];
        Iterator<E> it = body.iterator();
        int i = 0;
        while(it.hasNext())
            array[i++] = it.next();
        return array;
    }

    public <T> T[] toArray(T[] a) {
        throw new UnsupportedOperationException();
    }

    public String toString() {
        StringBuffer stb = new StringBuffer("[ ");
        if(!isEmpty()) {
            Iterator<E> it = iterator();
            while(it.hasNext())
                stb.append(it.next().toString() + " ; ");
            stb.delete(stb.length()-2, stb.length()-1);
        }
        stb.append("]");
        return stb.toString();
    }

    public Set<E> union(Set<E> other) {
        VectorSet<E> result = null;
        try {
            result = (VectorSet<E>)this.clone();
        }
        catch(CloneNotSupportedException e){}
        Iterator<E> it = other.iterator();
        E temp;
        while(it.hasNext())
            if(!result.contains(temp = it.next()))
                result.add(temp);
        return result;
    }
}
```