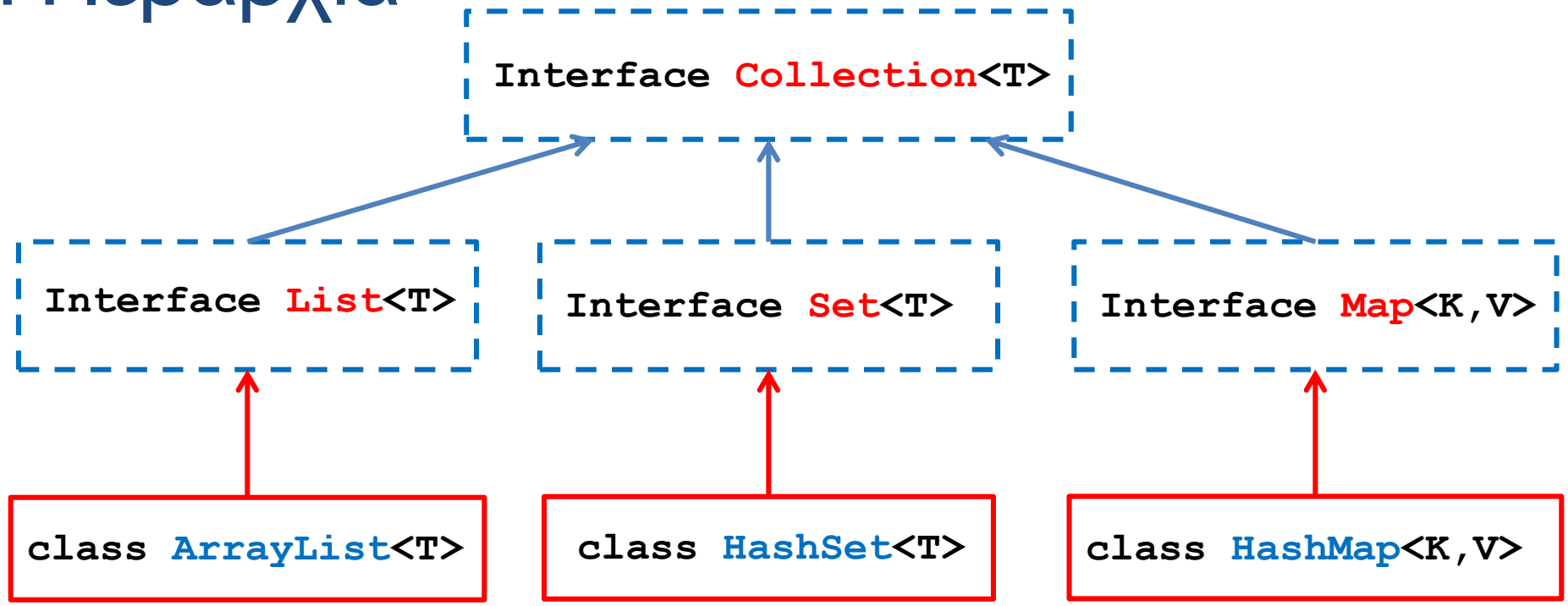
ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Συλλογές

ΣΥΛΛΟΓΕΣ

Η ιεραρχία



Αποθηκεύει δεδομένα σε σειριακή μορφή. Υπάρχει η έννοια της διάταξης. Καλό αν θέλουμε να διατρέχουμε τα δεδομένα συχνά και γρήγορα.

Αποθηκεύει δεδομένα σαν σύνολο χωρίς διάταξη. Καλό αν θέλουμε να βρίσκουμε γρήγορα αν ένα στοιχείο ανήκει στο σύνολο

Αποθηκεύει (key,value) ζεύγη. Παρόμοια δομή με το HashSet για την αποθήκευση των κλειδιών, αλλά τώρα κάθε κλειδί (key) σχετίζεται με μία τιμή (value).

ArrayList (JavaDocs link)

- Constructor
  - `ArrayList<T> myList = new ArrayList<T>();`
- Μέθοδοι
  - `add(T x)`: προσθέτει το στοιχειό `x` στο τέλος του πίνακα.
  - `add(int i, T x)`: προσθέτει το στοιχειό `x` στη θέση `i` και μετατοπίζει τα υπόλοιπα στοιχεία κατά μια θέση.
  - `remove(int i)`: αφαιρεί το στοιχείο στη θέση `i` και το επιστρέφει.
  - `remove(T x)`: αφαιρεί το στοιχείο
  - `set(int i, T x)`: θέτει στην θέση `i` την τιμή `x` αλλάζοντας την προηγούμενη
  - `get(int i)`: επιστρέφει το αντικείμενο τύπου `T` στη θέση `i`.
  - `contains(T x)`: boolean αν το στοιχείο `x` ανήκει στην λίστα ή όχι.
  - `size()`: ο αριθμός των στοιχείων του πίνακα.
- Διατρέχοντας τον πίνακα:
  - `ArrayList<T> myList = new ArrayList<T>();`
  - `for(T x: myList){…}`

HashSet (JavaDocs link)

- Constructors
  - `HashSet<T> mySet = new HashSet<T>();`
- Μέθοδοι
  - `add(T x)`: προσθέτει το στοιχειό `x` αν δεν υπάρχει ήδη στο σύνολο.
  - `remove(T x)`: αφαιρεί το στοιχείο `x`.
  - `contains(T x)`: boolean αν το σύνολο περιέχει το στοιχείο `x` ή όχι.
  - `size()`: ο αριθμός των στοιχείων στο σύνολο.
  - `isEmpty()`: boolean αν έχει στοιχεία το σύνολο ή όχι.
  - `Object[] toArray()`: επιστρέφει πίνακα με τα στοιχεία του συνόλου (επιστρέφει πίνακα από Objects – χρειάζεται downcasting μετά).
- Διατρέχοντας τα στοιχεία του συνόλου:
  - `HashSet<T> mySet = new HashSet<T>();`
  - `for(T x: mySet){…}`

HashMap (JavaDocs link)

- Αποθηκεύει ζευγάρια από τιμές και κλειδιά.
- Constructor
  - `HashMap<K,V> myMap = new HashMap<K,V>();`
- Μέθοδοι
  - `put(K key, V value)`: προσθέτει το ζευγάρι (`key,value`) (δημιουργεί μία συσχέτιση)
  - `V get(K key)`: επιστρέφει την τιμή για το κλειδί `key`.
  - `remove(K key)`: αφαιρεί το ζευγάρι με κλειδί `key`.
  - `containsKey(K key)`: boolean αν το σύνολο περιέχει το κλειδί `key` ή όχι.
  - `containsValue(V value)`: boolean αν το σύνολο περιέχει την τιμή `value` ή όχι. (αργό)
  - `size()`: ο αριθμός των στοιχείων (κλειδιών) στο map.
  - `isEmpty()`: boolean αν έχει στοιχεία το map ή όχι.
  - `Set<K> keySet()`: επιστρέφει ένα Set με τα κλειδιά.
  - `Collection<V> values()`: επιστρέφει ένα Collection με τις τιμές
  - `Set<Map.entry<K,V>> entrySet()`: επιστρέφει μία Set αναπαράσταση των key-value εγγραφών στο HashMap

Iterators

- Ένα interface που μας δίνει τις λειτουργίες για να διατρέχουμε ένα Collection
  - Ιδιαίτερα χρήσιμοι αν θέλουμε να αφαιρέσουμε στοιχεία από ένα Collection.
- Μέθοδοι του Iterator<T>
  - `hasNext()`: boolean αν ο iterator έχει φτάσει στο τέλος ή όχι.
  - `T next()`: επιστρέφει την επόμενη τιμή (αναφορά όχι αντίγραφο)
  - `remove()`: αφαιρεί το στοιχείο το οποίο επέστρεψε η τελευταία next()
- Μέθοδος του Collection :
  - `Iterator iterator()`: επιστρέφει ένα iterator για μία συλλογή. Π.χ.:
    - `HashSet<String> mySet = new HashSet<String>();`
    - `Iterator<String> iter = mySet.iterator();`

```
import java.util.HashSet;
import java.util.Scanner;

public class WrongIteratorExample
{
   public static void main(String[] args){
        HashSet<String> mySet = new HashSet<String>();
        Scanner input = new Scanner(System.in);

        while(input.hasNext()){
           if (!mySet.contains(name)){
                mySet.add(input.next());
           }
        }

        for (String s: mySet){
           if (s.length() <= 2){
                mySet.remove(s);
           }
        }

        for (String s:mySet){
           System.out.println(s);
        }
   }
}
```

Θέλω να αφαιρέσω από το σύνολο τα Strings με λιγότερους από 2 χαρακτήρες

Αν διατρέξουμε το set με την for-each εντολή θα πάρουμε (συνήθως) **λάθος**.

Δεν μπορούμε να αλλάζουμε το Collection ενώ το διατρέχουμε!

```
import java.util.HashSet;
import java.util.Iterator;
import java.util.Scanner;

public class IteratorExample
{
   public static void main(String[] args){
        HashSet<String> mySet = new HashSet<String>();
        Scanner input = new Scanner(System.in);

        while(input.hasNext()){
           if (!mySet.contains(name)){ mySet.add(input.next()); }
        }

        Iterator<String> it = mySet.iterator();
        while (it.hasNext()){
           if (it.next().length() <= 2){
                it.remove();
           }
        }

        it = mySet.iterator();
        while (it.hasNext()){
           System.out.println(it.next());
        }
   }
}
```

Θέλω να αφαιρέσω από το σύνολο τα Strings με λιγότερους από 2 χαρακτήρες

Ο Iterator μας επιτρέπει να διατρέχουμε την συλλογή και να διαγράφουμε στοιχεία.

Ξανα-διατρέχουμε τον πίνακα. Ο iterator πρέπει να ξανα-οριστεί για να ξεκινήσει από την αρχή του συνόλου.

```java
import java.util.Map;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Scanner;

class IteratorExample2
{
    public static void main(String[] args){
        HashMap<String, Integer> myMap = new HashMap<String,Integer>();
        Scanner input = new Scanner(System.in);

        while(input.hasNext()){
            String name = input.next();
            if (!myMap.containsKey(name)){myMap.put(name,1);}
            else{ myMap.put(name,myMap.get(name)+1);}
        }

        Iterator<Map.Entry<String,Integer>> iter = myMap.entrySet().iterator();
        while(iter.hasNext()){
            if (iter.next().getValue() <=2){
                iter.remove();
            }
        }

        for(String key: myMap.keySet()){
            System.out.println(key + ":" + myMap.get(key));
        }
    }
}
```

Θέλω να αφαιρέσω από το σύνολο τα Strings με λιγότερες από 2 εμφανίσεις

Η entrySet επιστρέφει μια συλλογή από Map.entry αντικείμενα (γι αυτό πρέπει να κάνουμε import το Map) τα οποία παραμετροποιούμε με τους τύπους που κρατά το HashMap

ListIterator<T>

- Ένας Iterator ειδικά για την συλλογή List
  - Κύριο πλεονέκτημα ότι επιτρέπει διάσχιση της λίστας προς τις δύο κατευθύνσεις και αλλαγές στη λίστα ενώ την διατρέχουμε.
- Επιπλέον μέθοδοι της ListIterator
  - `hasPrevious()`: boolean αν υπάρχουν κι άλλα στοιχεία πριν από αυτό στο οποίο είμαστε.
  - `T previous()`: επιστρέφει την προηγούμενη τιμή
  - `set(T)`: Θέτει την τιμή του στοιχείου που επέστρεψε η τελευταία next()
  - `add(T)`: Προσθέτει ένα στοιχείο στη λίστα αμέσως μετά από αυτό στο οποίο βρισκόμαστε
- Μέθοδος της List :
  - `ListIterator listIterator()`: επιστρέφει ένα iterator για μία συλλογή.

Κάθε a μετέτρεψε το σε b και πρόσθεσε μετά και ένα c

```
import java.util.*;

public class ListIteratorExample
{
        public static void main(String[] args){
                ArrayList<String> array = new ArrayList<String>();
                Scanner input = new Scanner(System.in);

                while(input.hasNext()){
                        String name = input.next();
                        array.add(name);
                }

                ListIterator<String> it = array.listIterator();
                while (it.hasNext()){
                        if (it.next().equals("a")){
                                it.set("b");
                                it.add("c");
                        }
                }
                it = array.listIterator();
                while (it.hasNext()){
                        System.out.println(it.next());
                }

        }
}
```

```
import java.util.*;

public class ListIteratorExample
{
    public static void main(String[] args){
       ArrayList<String> myList = new ArrayList<String>();
       Scanner input = new Scanner(System.in);

       while(input.hasNext()){
            String name = input.next();
            myList.add(name);
       }

       myList.remove("a");

       for (String s: myList){
            System.out.println(s);
       }
    }
}
```

Θέλω να αφαιρέσω από τις εμφανίσεις του String “a” από την λίστα μου

Η κλήση της remove θα αφαιρέσει μόνο την πρώτη εμφάνιση του “a”

Πως θα τις αφαιρέσουμε όλες?

Υπενθύμιση: η remove επιστρέφει boolean αν έγινε επιτυχώς αφαίρεση (αν άλλαξε δηλαδή η λίστα).

```
import java.util.*;

public class ListIteratorExample
{
    public static void main(String[] args){
       ArrayList<String> myList = new ArrayList<String>();
       Scanner input = new Scanner(System.in);

       while(input.hasNext()){
            String name = input.next();
            myList.add(name);
       }

       while(myList.remove("a"));

       for (String s: myList){
            System.out.println(s);
       }
    }
}
```

Θέλω να αφαιρέσω από τις εμφανίσεις του String “a” από την λίστα μου

Καλεί την remove μέχρι να επιστρέψει false

Η υλοποίηση αυτή όμως **δεν είναι αποδοτική** γιατί κάθε φορά που καλούμε την remove διατρέχουμε την λίστα από την αρχή. Είναι καλύτερα να χρησιμοποιήσουμε ένα iterator.

Χρήση των συλλογών

- Οι τρεις συλλογές που περιγράψαμε είναι πάρα πολύ χρήσιμες για να κάνετε γρήγορα προγράμματα
  - Συνηθίσετε να τις χρησιμοποιείτε και μάθετε πότε βολεύει να χρησιμοποιείτε την κάθε δομή
- Το HashMap είναι ιδιαίτερα χρήσιμο γιατί μας επιτρέπει πολύ γρήγορα να κάνουμε lookup: να βρίσκουμε ένα κλειδί μέσα σε ένα σύνολο και την συσχετιζόμενη τιμή

Παραδείγματα

- Έχουμε ένα πρόγραμμα που διαχειρίζεται τους φοιτητές ενός τμήματος. Ποια συλλογή πρέπει να χρησιμοποιήσουμε αν θέλουμε να λύσουμε τα παρακάτω προβλήματα?
  1. Θέλουμε να μπορούμε να εκτυπώσουμε τις πληροφορίες για τους φοιτητές που παίρνουν ένα μάθημα.
     - `ArrayList<Student> allStudents`
  2. Θέλουμε να μπορούμε να τυπώσουμε τις πληροφορίες για ένα συγκεκριμένο φοιτητή (χρησιμοποιώντας το ΑΜ του φοιτητή)
     - `HashMap<Integer,Student> allStudents`
  3. Θέλουμε να ξέρουμε ποιοι φοιτητές έχουν ξαναπάρει το μάθημα και να μπορούμε να ανακτήσουμε αυτή την πληροφορία για κάποιο φοιτητή
     - `HashSet<Integer> repeatStudents`
     - `HashSet<Student> repeatStudents`

Αναζήτηση με ΑΜ

Αναζήτηση με αντικείμενο

Χρήση δομών

- ArrayList: όταν θέλουμε να διατρέχουμε τα αντικείμενα ή όταν θέλουμε διάταξη των αντικείμενων, και δεν θα χρειαστούμε αναζήτηση κάποιου αντικείμενου
  - Π.χ., μια κλάση Course περιέχει μια λίστα από αντικείμενα τύπου Students
    - Εφόσον μας ενδιαφέρει να τυπώνουμε μόνο.

- HashSet: όταν θέλουμε να έχουμε μια συλλογή από μοναδικά αντικείμενα και θέλουμε γρήγορη αναζήτηση για να μάθουμε αν κάποιο αντικείμενο ανήκει σε αυτή
  - Π.χ., να βρούμε αν ένας φοιτητής (ΑΜ) ανήκει στη λίστα των φοιτητών που ξαναπαίρνουν το μάθημα
  - Π.χ., να βρούμε τα μοναδικά ονόματα από μια λίστα με ονόματα με επαναλήψεις

- HashMap: Ίδια λειτουργικότητα με το HashSet αλλά μας επιτρέπει να συσχετίσουμε μια τιμή με κάθε στοιχείο του συνόλου
  - Π.χ. θέλω να ανακαλέσω γρήγορα τις πληροφορίες για ένα φοιτητή χρησιμοποιώντας το ΑΜ του
  - Το HashMap είναι πιο χρήσιμο απ’ ότι ίσως θα περιμένατε

Περίπλοκες δομές

- Έχουμε μάθει τρεις βασικές δομές
  - `ArrayList`
  - `HashSet`
  - `HashMap`
- Μπορούμε να δημιουργήσουμε αντικείμενα που συνδιάζουν αυτές τις δομές
  - `HashMap<String,ArrayList<String>>`
  - `ArrayList<HashSet<String>>`
  - `HashMap<Integer,HashMap<String,String>>`

Παράδειγμα

- Θέλουμε για καθένα από τα μοναδικά Strings που διαβάζουμε να κρατάμε τις θέσεις στις οποίες εμφανίστηκαν.
  - Π.χ., αν έχουμε είσοδο “a b a c b a”, για το “a” θα τυπώσουμε τις θέσεις 0,2,5, για το “b” θα τυπώσουμε τις θέσεις 1,4 και για το “c” τη θέση 3.

```
import java.util.HashMap;
import java.util.ArrayList;
import java.util.Scanner;

class HashMapArrayListExample
{
    public static void main(String[] args){
        HashMap<String,ArrayList<Integer>> myMap =
                                            new HashMap<String,ArrayList<Integer>>();
        Scanner input = new Scanner(System.in);

        int counter = 0;
        while(input.hasNext()){
            String name = input.next();
            if (!myMap.containsKey(name)){
                myMap.put(name,new ArrayList<Integer>());
            }
            myMap.get(name).add(counter);
            counter ++;
        }

        for(String name: myMap.keySet()){
            System.out.print(name + ":");
            for (Integer i:myMap.get(name)){
                System.out.print(" "+i);
            }
            System.out.println();
        }
    }
}
```

Παράδειγμα

- Στο πρόγραμμα της γραμματείας ενός πανεπιστημίου που κρατάει πληροφορία για τους φοιτητές, θέλω γρήγορα με το ΑΜ του φοιτητή να μπορώ να βρω το βαθμό για ένα μάθημα χρησιμοποιώντας τον κωδικό του μαθήματος. Τι δομή πρέπει να χρησιμοποιήσω?

Υλοποίηση

- Χρειάζομαι ένα HashMap με κλειδί το ΑΜ του φοιτητή ώστε να μπορούμε γρήγορα να βρούμε πληροφορίες για τον φοιτητή.
  - Τι τιμές θα κρατάει το HashMap?

- Θα πρέπει να κρατάει άλλο ένα HashMap το οποίο να έχει σαν κλειδί τον κωδικό του μαθήματος και σαν τιμή τον βαθμό του φοιτητή.

Ορισμός

```
HashMap<Integer,HashMap<Integer,double>> StudentCoursesGrades;
```

Χρήση

```
StudentCoursesGrades = new HashMap<Integer,HashMap<int,double>>();
StudentCoursesGrades.put(469,new HashMap<Integer,double>());
StudentCoursesGrades.get(469).put(205,9.5);
StudentCoursesGrades.get(469).get(205);
```

Προσθέτει το βαθμό

Διαβάζει το βαθμό

Διαφορετική υλοποίηση

- Στο πρόγραμμα μου να έχω μια κλάση Student που κρατάει τις πληροφορίες για ένα φοιτητή και μία κλάση StudentRecord που κρατάει την καρτέλα του φοιτητή για το μάθημα. Πως αλλάζει η υλοποίηση?

Ορισμός

```
HashMap<Integer,Student> allStudents;
```

Ορισμός

```
class Student
{
    private int AM;
    private HashMap<Integer,StudentRecord> courses;
    ...

    pubic StudentRecord getCourseRecord(int CourseId){
        return courses.get(courseId);
    }
}
```

Ορισμός

```
class StudentRecord
{
    private double grade;
    ...

    public double getGrade{
        return grade;
    }
}
```

Χρήση

```
allStudents.get(469).getCourseRecord(205).getGrade();
```

Ορισμός

```
HashMap<Integer,Student> allStudents;
```

Ορισμός

```
class Student
{
    private int AM;
    private HashMap<Integer,StudentRecord> courses;
    ...

    pubic HashMap<Integer,StudentRecord> getCourses(){
        return courses;
    }
}
```

Διαφορετική υλοποίηση
Μπορούμε να επιστρέφουμε ένα HashMap

Ορισμός

```
class StudentRecord
{
    private double grade;
    ...

    public double getGrade{
        return grade;
    }
}
```

Χρήση

```
allStudents.get(469).getCourses().get(205).getGrade();
```

Χρονική πολυπλοκότητα

- Έχει τόσο μεγάλη σημασία τι δομή θα χρησιμοποιήσουμε? Όλες οι δομές μας δίνουν περίπου την ίδια λειτουργικότητα.
  - ΝΑΙ!

- Αν κάνουμε αναζήτηση για μια τιμή σε ένα ArrayList πρέπει να διατρέξουμε τη λίστα για να δούμε αν ένα στοιχείο ανήκει ή όχι στη λίστα.
  - Κατά μέσο όρο θα συγκρίνουμε με τα μισά στοιχεία της λίστας
  - Η χρονική πολυπλοκότητα είναι γραμμική ως προς τον αριθμό των στοιχείων

- Σε ένα HashSet ή HashMap αυτό γίνεται σε χρόνο σχεδόν σταθερό (ή λογαριθμικό ως προς τον αριθμό των στοιχείων)
  - Αν έχουμε πολλά στοιχεία, και κάνουμε πολλές αναζητήσεις αυτό κάνει διαφορά

```
import java.util.*;

class ArrayHashComparison
{
    public static void main(String[] args){
        ArrayList<Integer> array = new ArrayList<Integer>();
        for (int i =0; i < 100000; i ++){
            array.add(i);
        }
        HashSet<Integer> set = new HashSet<Integer>();
        for (int i =0; i < 100000; i ++){
            set.add(i);
        }
        ArrayList<Integer> randomNumbers = new ArrayList<Integer>();
        Random rand = new Random();
        for (int i = 0; i < 100000; i ++){
            randomNumbers.add(rand.nextInt(200000));
        }

        long startTime = System.currentTimeMillis();
        for (Integer x:randomNumbers){
            boolean b = array.contains(x);
        }
        long endTime = System.currentTimeMillis();
        long duration = (endTime - startTime);
        System.out.println("Array took "+ duration + " millisecs");

        startTime = System.currentTimeMillis();
        for (Integer x:randomNumbers){
            boolean b = set.contains(x);
        }
        endTime = System.currentTimeMillis();
        duration = (endTime - startTime);
        System.out.println("Set took "+duration + " millisecs");
    }
}
```

Με το ArrayList κάνουμε περίπου 100000*100000/2 συγκρίσεις

Με το HashSet κάνουμε περίπου 100000 συγκρίσεις