ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Γενικευμένες κλάσεις

Συλλογές

Stack

- Θυμηθείτε πως ορίσαμε μια στοίβα ακεραίων

```
public class IntStack
{
        private IntStackElement head;
        private int size = 0;

        public int pop(){
                if (size == 0){ // head == null
                        System.out.println("Pop from empty stack");
                        System.exit(-1);
                }
                int value = head.getValue();
                head = head.getNext();
                size --;
                return value;
        }

        public void push(int value){
                IntStackElement element = new IntStackElement(value);
                element.setNext(head);
                head = element;
                size ++;
        }

}
```

```
public class IntStackElement
{
        private int value;
        private IntStackElement next = null;

        public IntStackElement(int value){
                this.value = value;
        }

        public int getValue(){
                return value;
        }

        public IntStackElement getNext(){
                return next;
        }

        public void setNext(IntStackElement element){
                next = element;
        }

}
```

Stack

- Αν θέλουμε η στοίβα μας να αποθηκεύει αντικείμενα της κλάσης Person θα πρέπει να ορίσουμε μια διαφορετική Stack και διαφορετική StackElement.

```
class PersonStackElement
{
        private Person value;
        private PersonStackElement next;

        public PersonStackElement(Person val){
                value = val;
        }

        public void setNext(PersonStackElement element){
                next = element;
        }

        public PersonStackElement getNext(){
                return next;
        }

        public Person getValue(){
                return value;
        }

}
```

```
public class PersonStack
{
        private PersonStackElement head;
        private int size = 0;

        public Person pop(){
          if (size == 0){ // head == null
                System.out.println("Pop from empty stack");
                System.exit(-1);
          }
          Person value = head.getValue();
          head = head.getNext();
          size --;
          return value;
        }

        public void push(Person value){
          PersonStackElement element = new PersonStackElement(value);
          element.setNext(head);
          head = element;
          size ++;
        }

}
```

Stack

- Θα ήταν πιο βολικό αν μπορούσαμε να ορίσουμε μία μόνο κλάση Stack που να μπορεί να αποθηκεύει αντικείμενα οποιουδήποτε τύπου.
  - Πώς μπορούμε να το κάνουμε αυτό?
- Μια λύση: Η ObjectStack που κρατάει αντικείμενα Object, την πιο γενική κλάση
- Τι πρόβλημα μπορεί να έχει αυτό?

```
class ObjectStackElement
{
        private Object value;
        private ObjectStackElement next;

        public ObjectStackElement(Object val){
                value = val;
        }

        public void setNext(ObjectStackElement element){
                next = element;
        }

        public ObjectStackElement getNext(){
                return next;
        }

        public Object getValue(){
                return value;
        }

}
```

```
public class ObjectStack
{
        private ObjectStackElement head;
        private int size = 0;

        public Object pop(){
          if (size == 0){ // head == null
                System.out.println("Pop from empty stack");
                System.exit(-1);
          }
          Object value = head.getValue();
          head = head.getNext();
          size --;
          return value;
        }

        public void push(Object value){
          ObjectStackElement element = new ObjectStackElement(value);
          element.setNext(head);
          head = element;
          size ++;
        }

}
```

```
public class StackTest
{
  public static void main(String[] args){
    ObjectStack stack = new ObjectStack();

    Person p = new Person("Alice", 1);
    Integer i = new Integer(10);
    String s = "a random string";

    stack.push(p);
    stack.push(i);
    stack.push(s);
  }
}
```

Δεν μπορούμε να ελέγξουμε τι αντικείμενα μπαίνουν στην στοίβα. Κατά την εξαγωγή θα πρέπει να γίνει μετατροπή και θέλει προσοχή να μετατρέπουμε το σωστό αντικείμενο στον σωστό τύπο.

Θέλουμε να δημιουργούμε στοίβες συγκεκριμένου τύπου.

Γενικευμένες (Generic) κλάσεις

- Οι γενικευμένες κλάσεις περιέχουν ένα τύπο δεδομένων T που ορίζεται παραμετρικά
- Όταν χρησιμοποιούμε την κλάση αντικαθιστούμε την παράμετρο T με τον τύπο δεδομένων (την κλάση) που θέλουμε
- Συντακτικό:
  - `public class Example<T> {…}`
- Ορίζει την γενικευμένη κλάση Example με παράμετρο τον τύπο T
  - Μέσα στην κλάση ο τύπος T χρησιμοποιείται σαν τύπος δεδομένων
- Όταν χρησιμοποιούμε την κλάση Example αντικαθιστούμε το T με κάποια συγκεκριμένη κλάση
  - `Example<String> ex = new Example<String>();`

Ένα πολύ απλό παράδειγμα

```
public class Example<T>{
   T data;

   public Example(T data){
         this.data = data;
   }

   public void setData(T data){
      this.data = data;
   }

   public T getData(){
      return data;
   }

   public static void main(String[] args){
      Example<String> ex = new Example<String>("hello world");
      System.out.println(ex.getData());
   }
}
```

Όταν ορίζουμε το αντικείμενο ex η κλάση String αντικαθιστά τις εμφανίσεις του T στον κώδικα

Ο ορισμός του constructor γίνεται χωρίς το <T> παρότι στην δημιουργία του αντικειμένου χρησιμοποιούμε το <String>

Γενικευμένη Στοίβα

- Μπορούμε τώρα να φτιάξουμε μια στοίβα για οποιοδήποτε τύπο δεδομένων

```
class StackElement<T>
{
        private T value;
        private StackElement<T> next;

        public StackElement(T val){
                value = val;
        }

        public void setNext(StackElement<T> element){
                next = element;
        }

        public StackElement<T> getNext(){
                return next;
        }

        public T getValue(){
                return value;
        }

}
```

```
public class Stack<T>
{
        private StackElement<T> head;
        private int size = 0;

        public T pop(){
          if (size == 0){ // head == null
                System.out.println("Pop from empty stack");
                System.exit(-1);
          }
          T value = head.getValue();
          head = head.getNext();
          size --;
          return value;
        }

        public void push(T value){
          StackElement<T> element = new StackElement<T>(value);
          element.setNext(head);
          head = element;
          size ++;
        }

}
```

```
public class StackTest
{
    public static void main(String[] args){
        Stack<Person> personStack = new Stack<Person>();

         personStack.push(new Person("Alice", 1));
        personStack.push(new Person("Bob",2));
        System.out.println(personStack.pop());
        System.out.println(personStack.pop());

        Stack<Integer> intStack = new Stack<Integer>();
        intStack.push(new Integer(10));
        intStack.push(new Integer(20));
        System.out.println(intStack.pop());
        System.out.println(intStack.pop());

        Stack<String> stringStack = new Stack<String>();
        stringStack.push("string 1");
        stringStack.push("string 2");
        System.out.println(stringStack.pop());
        System.out.println(stringStack.pop());
    }
}
```

Δημιουργούμε στοίβες συγκεκριμένου τύπου.

Πολλαπλές παράμετροι

- Μπορούμε να έχουμε πάνω από ένα παραμετρικούς τύπους

```
public class KeyValuePair<K,V>{
    private K key;
    private V value;
  …
}
```

Παγίδες

1. Ο τύπος T δεν μπορεί να αντικατασταθεί από ένα πρωταρχικό τύπο δεδομένων (π.χ. int, double, boolean – πρεπει να χρησιμοποιήσουμε τα wrapper classes γι αυτά, Integer, Boolean, Double)
2. Δεν μπορούμε να ορίσουμε ένα πίνακα από αντικείμενα γενικευμένης κλάσης.

   Π.χ., `StackElement<String>[] A;`

   Δεν είναι αποδεκτό!
3. Δεν μπορούμε να χρησιμοποιούμε τον τύπο T όπως οποιαδήποτε άλλη κλάση.

   Π.χ., `T obj = new T();`

   `T[] a = new T[10];`

   Δεν είναι αποδεκτά!

Γενικευμένες κλάσεις με περιορισμούς

- Ας υποθέσουμε ότι θέλουμε να ορίσουμε μία γενικευμένη κλάση Pair η οποία κρατάει ένα ζεύγος από δυο αντικείμενα οποιουδήποτε τύπου.

```
public class Pair<T>{
    private T first;
    private T second;
  …
}
```

Γενικευμένες κλάσεις με περιορισμούς

- Θέλουμε επίσης να μπορούμε να διατάσουμε τα ζεύγη
  - Για να γίνει αυτό θα πρέπει να υπάρχει τρόπος να συγκρίνουμε τα στοιχεία first και second.
  - Περιορίζουμε την T να υλοποιεί το interface Comparable

```
public class Pair<T extends Comparable>{
    private T first;
    private T second;

    public void order(){
        if (first.compareTo(second) > 0){
          T temp = first; first = second; second = temp;
        }
    }
}
```

extends όχι implements

Γενικευμένες κλάσεις με περιορισμούς

- Μπορούμε να περιορίσουμε τον παραμετρικό τύπο να κληρονομεί οποιαδήποτε κλάση, ή οποιοδήποτε interface ή συνδυασμό από τα παραπάνω.
  - `public class SomeClass <T extends Employee> { … }`
  - `public class SomeClass <T extends Employee & Comparable> { … }`

Δέχεται μόνο απογόνους της Employee

Δέχεται μόνο απογόνους της Employee που υλοποιούν το interface Comparable

Γενικευμένες κλάσεις και κληρονομικότητα

- Μια γενικευμένη κλάση μπορεί να έχει απογόνους άλλες γενικευμένες κλάσεις.
  - Οι απόγονοι κληρονομούν και τον τύπο T.
  - `public class OrderedPair<T> extends Pair<T> { … }`

- Δεν ορίζεται κληρονομικότητα ως προς τον παραμετρικό τύπο T
  - Δεν υπάρχει καμία σχέση μεταξύ των κλάσεων `Pair<Employee>` και `Pair<HourlyEmployee>`

Wildcard

- Αν θέλουμε να ορίσουμε ένα γενικό παραμετρικό τύπο χρησιμοποιούμε την παράμετρο μπαλαντέρ ?, η οποία αναπαριστά ένα οποιοδήποτε τύπο T.

- `public void someMethod(Pair<?>){ … }`

- Μπορούμε να περιοριστούμε σε ένα τύπο που είναι απόγονος της Employee.

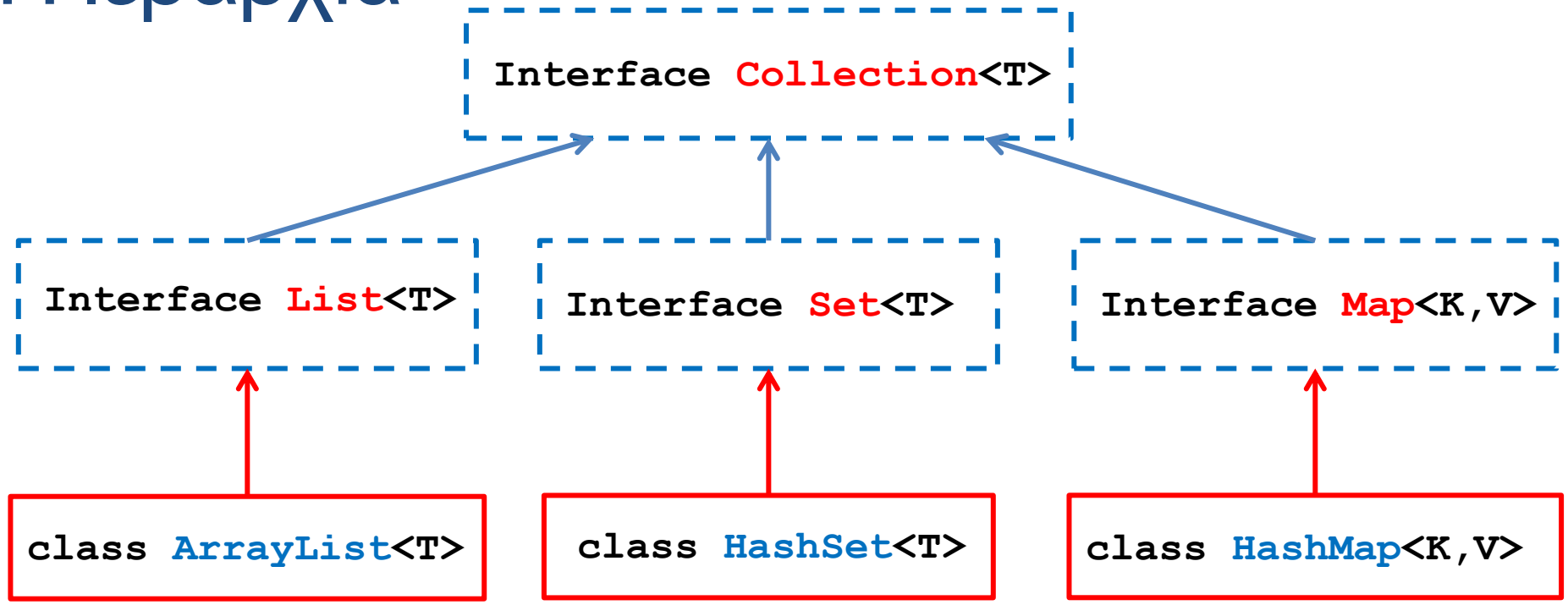
- ```
  public void someMethod(
      Pair<? extends Employee>){ … }
  ```
  ```

ΣΥΛΛΟΓΕΣ

ArrayList

- Η κλάση ArrayList<T> είναι μια περίπτωση γενικευμένης κλάσης
  - Ένας δυναμικός πίνακας που ορίζεται με παράμετρο τον τύπο των αντικειμένων που θα κρατάει.
- Η ArrayList<T> είναι μία από τις συλλογές (Collections) που είναι ορισμένες στην Java.
  - Υπάρχουσες δομές δεδομένων που μας βοηθάνε στην αποθήκευση και ανάκτηση των δεδομένων.

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)

- Constructors
  - `ArrayList<T> myList = new ArrayList<T>();`
  - `ArrayList<T> myList = new ArrayList<T>(10);`
    `//λιστα με χωρητικότητα 10`
- 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`
  - `set(int i, T x)`: αλλάζει την τιμή της θέσης `i` με την τιμή `x`
  - `get(int i)`: επιστρέφει την τιμή στη θέση `i`.
  - `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 casting μετά).
- Διατρέχοντας τα στοιχεία του συνόλου:
  - `HashSet<T> mySet = new HashSet<T>();`
  - `for(T x: mySet){…}`

Παράδειγμα Ι

- Διαβάζουμε μια σειρά από Strings και θέλουμε να βρούμε όλα τα μοναδικά Strings
  - Π.χ. να φτιάξουμε το λεξικό ενός βιβλίου

- Πώς θα το υλοποιήσουμε αυτό?
  - Με ArrayList?
    - Πρέπει να κάνουμε πάρα πολλές συγκρίσεις
  - Με HashSet?
    - Η αναζήτηση ενός string γίνεται πολύ πιο γρήγορα.

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

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

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

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

        Object[] array = mySet.toArray();
        for (int i = 0; i < array.length; i ++){
            String name = (String)array[i];
            System.out.println(name);
        }
    }
}
```

Δήλωση μιας μεταβλητής HashSet από Strings.

Τοποθετούμε στο HashSet μόνο τα Strings τα οποία δεν έχουμε ήδη δει (δεν είναι ήδη στο σύνολο)

Ένας τρόπος για να διατρέξουμε και να τυπώσουμε τα στοιχεία του HashSet

Ένας άλλος τρόπος για να διατρέξουμε το HashSet χρησιμοποιώντας την εντολή toArray().
Ο πίνακας είναι πίνακας από Objects, και πρέπει να κάνουμε downcasting

HashMap (JavaDocs link)

- Constructors
  - `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

Παράδειγμα ΙΙ

- Διαβάζουμε μια σειρά από Strings και θέλουμε να βρούμε όλα τα μοναδικά Strings και να τους δώσουμε ένα μοναδικό id.
  - Π.χ. να δώσουμε αριθμούς σε μία λίστα με ονόματα
- Πώς θα το υλοποιήσουμε αυτό?
- Τι γίνεται αν θέλουμε να δημιουργήσουμε ένα αντικείμενο Person για κάθε μοναδικό όνομα?

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

public class HashMapExample1
{
   public static void main(String[] args){
        HashMap<String,Integer> myMap = new HashMap<String,Integer>();
        Scanner input = new Scanner(System.in);

        int counter = 0;
        while(input.hasNext()){
           String name = input.next();
           if (!myMap.containsKey(name)){
                myMap.put(name,counter);
                counter ++;
           }
        }

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

Δήλωση μιας μεταβλητής HashMap που συσχετίζει Strings (κλειδια) και Integers (τιμές) Για κάθε όνομα (String) το id (Integer)

Αν το όνομα δεν είναι ήδη στο HashMap τότε ανάθεσε στο όνομα αυτό τον επόμενο αύξοντα αριθμό και πρόσθεσε ένα νέο ζευγάρι (όνομα αριθμός) στο HashMap.

Διατρέχοντας το HashMap

Διέτρεξε το σύνολο με τα κλειδιά (ονόματα) στο HashMap

Για κάθε κλειδί (όνομα) πάρε το id που αντιστοιχεί στο όνομα αυτό και τύπωσε το.

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

public class HashMapExample2
{
   public static void main(String[] args){
        HashMap<String,Person> myMap = new HashMap<String,Person>();
        Scanner input = new Scanner(System.in);

        int counter = 0;
        while(input.hasNext()){
           String name = input.next();
           if (!myMap.containsKey(name)){
                Person p = new Person(name,counter);
                myMap.put(name,p);
                counter ++;
           }
        }

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

Δημιουργούμε ένα HashMap το οποίο σε κάθε διαφορετικό όνομα αντιστοιχεί ένα αντικείμενο Person.

Καλείται η toString της κλάσης Person

Iterators

- Ένα interface που μας δίνει τις λειτουργίες για να διατρέχουμε ένα Collection
- Μέθοδοι
  - `hasNext()`: boolean αν ο iterator έχει φτάσει στο τέλος ή όχι.
  - `T next()`: επιστρέφει την τιμή επόμενη τιμή
  - `remove()`: αφαιρεί το στοιχείο το οποίο επέστρεψε η τελευταία next()
    - Προσοχή, δεν μπορούμε να καλέσουμε την remove ενώ συνεχίζεται το iteration.
- Μέθοδος της Set:
  - `Iterator iterator()`: επιστρέφει ένα iterator για το set.

```
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 it = mySet.iterator();
        while (it.hasNext()){
           if (it.next().equals("a")){
                it.remove();
                break;
           }
        }
        it = mySet.iterator();
        while (it.hasNext()){
           System.out.println(it.next());
        }
   }
}
```

Διατρέχει το σύνολο και αν βρει το String “a” το αφαιρεί από το σύνολο.

Πρέπει να κάνουμε break γιατί αν αφαιρέσουμε μία τιμή ενώ διατρέχουμε το σύνολο μπορεί να προκληθεί λάθος.

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

Συλλογές

- 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
  - Π.χ., στο παράδειγμα με το τμήμα του πανεπιστημίου πως θα ανακτήσουμε την καρτέλα ενός φοιτητή?
  - Πως μπορείτε να αποθηκεύσετε τα χαρτιά για κάθε χρώμα στο CrazyEights?