ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS
ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

Οι πίνακες ως Συλλογές Βασικές Πράξεις Filter, Map, Reduce και Predicates

Αθ. Ανδρούτσος

Πίνακες - Συλλογές

- Οι πίνακες είναι μία γραμμική δομή δεδομένων, αλλά γενικότερα είναι μία συλλογή (συσχετιζόμενων) στοιχείων. Στα στοιχεία συλλογών μπορούμε να κάνουμε κάποιες βασικές πράξεις όπως
  - **Filtering**, για παράδειγμα να εξάγουμε ένα νέο πίνακα μόνο με τους ζυγούς
  - **Mapping**, για παράδειγμα να εξάγουμε ένα νέο πίνακα όπου σε κάθε στοιχείο έχουμε προσθέσει το ένα (1)
  - **Reducing**, όπου βρίσκουμε για παράδειγμα το άθροισμα ή τον μέσο όρο, κλπ. των στοιχείων του πίνακα
  - **Predicates**, τα predicates είναι boolean functions (methods) για παράδειγμα αν ο πίνακας περιέχει έστω ένα άρτιο ή αν όλα τα στοιχεία είναι άρτιοι, κλπ.

Filtering

- Έστω ότι έχουμε ένα array με τους βαθμούς μία τάξης και θέλουμε να εξάγουμε ένα άλλο array με τους βαθμούς που είναι PASS, δηλαδή >= 5
- Το filtering γίνεται με διάσχιση και συγκρίσεις

Pass Grades


```java
package gr.aueb.cf.testbed.ch6;

/**
 * Demonstrates Filtering with arrays.
 */
public class Filtering {

    public static void main(String[] args) {
        int[] grades = new int[] {4, 9, 9, 8, 7, 2, 1, 4, 10};
        int PASS = 5;

        int[] passed = getPassGrades(grades, PASS);
        for (int pass : passed) {
            System.out.print(pass + " ");
        }
    }
```

Get pass grades


```java
/**
 * Returns an array with the grades that are
 * greater or equal to a certain limit.
 *
 * @param grades    the source array of grades.
 * @param limit     the pass grade limit.
 * @return          a new array with the 'pass' elements.
 */
public static int[] getPassGrades(int[] grades, int limit) {
    int count = 0;
    if (grades == null) return null;
    for (int grade : grades) {
        if (grade >= limit) {
            count++;
        }
    }
    int[] passedOut = new int[count];
    int pivot = -1;
    for (int grade : grades) {
        if (grade >= limit) {
            passedOut[++pivot] = grade;
        }
    }
    return passedOut;
}
```

- Κάνει δύο περάσματα (διασχίσεις, traverse)
- Στο 1ο πέρασμα μετράει πόσα στοιχεία pass είναι μέσα στον πίνακα, ώστε να δημιουργήσει ένα νέο πίνακα (gradesOut) με διάσταση όσα και τα στοιχεία που θα επιστρέψουμε
- Στο 2ο πέρασμα εισάγει στον πίνακα gradesOut τα στοιχεία αυτά και επιστρέφει τον gradesOut

Mapping (1)


```java
package gr.aueb.cf.testbed.ch6;

/**
 * Demonstrates Array Mapping.
 */
public class Mapping {

    public static void main(String[] args) {
        int[] salaries = {1000, 1200, 900, 1700};
        double BONUS = 0.25;
        double[] wages = getWagesWithBonus(salaries, BONUS);
        for (double wage : wages) {
            System.out.printf("%.2f  ", wage);
        }
    }
```

Mapping (2)


```java
17      /**
18       * Returns a new array of the wages plus the bonus.
19       *
20       * @param wages     the array of initial wages.
21       * @param bonus     the bonus, i.e. 0.10, as a
22       *                  percentage of the wage.
23       * @return          the updated waged
24       */
25      public static double[] getWagesWithBonus(int[] wages, double bonus) {
26          if (wages == null) return null;
27          double[] passedOut = new double[wages.length];
28
29          for (int i = 0; i < wages.length; i++) {
30              passedOut[i] = wages[i] + wages[i] * bonus;
31          }
32          return passedOut;
33      }
34  }
```

Reducing (1)


```java
package gr.aueb.cf.testbed.ch6;

/**
 * Demonstrates Reducing an array to
 * single value, e.g. total, avg.
 */
public class Reducing {

    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5, 6, 7};
        System.out.println("Total: " + getTotal(arr));
        System.out.printf("Average: %.2f", getAvg(arr));
    }
```

- Η λογική του reducing είναι λογική aggregation

Reducing (2)


```java
/**
 * Returns the sum of the array elements.
 *
 * @param arr   the source array.
 * @return      the total (sum).
 */
public static int getTotal(int[] arr) {
    if (arr == null)  return 0;
    int total = 0;
    for (int item : arr) {
        total += item;
    }
    return total;
}
```

```java
/**
 * Returns the average of the array elements.
 *
 * @param arr   the source array.
 * @return      the average.
 */
public static double getAvg(int[] arr) {
    if (arr == null)  return 0;
    int total = 0;
    int avg = 0;
    for (int item : arr) {
        total += item;
    }
    return (double) total / arr.length;
}
```

- Υπολογίζει (πάνω αριστερά) το άθροισμα των τιμών του array που δέχεται ως είσοδο και (πάνω δεξιά) τον μέσο όρο των τιμών του array που δέχεται ως είσοδο

Predicates

- Τα predicates είναι boolean functions (methods). **Επιστρέφουν true ή false.**
- Διασχίζουν μία συλλογή όπως ένας πίνακας και ελέγχουν αν τα στοιχεία του συμμορφώνονται προς τη λογική που εκφράζουν, π.χ. αν υπάρχει ένας άρτιος, αν υπάρχουν 4 άρτιοι το πολύ, αν όλοι είναι άρτιοι, αν είναι συνεχόμενοι, αν έχουν τον ίδιο λήγοντα, αν είναι στην ίδια δεκάδα, κλπ

Άρτιοι


```java
/**
 * Traverses an array to decide if the array
 * involves more than two even numbers.
 *
 * @param arr    the source array.
 * @return       true, if the array contains
 *               more than two evens, false otherwise.
 */
public static boolean moreThanTwoEvens(int[] arr) {
    if (arr == null) return false;
    int evens = 0;
    for (int item : arr) {
        if (item % 2 == 0) {
            evens++;
        }
    }
    return evens > 2;
}
```

- Ελέγχει τον πίνακα για το αν περιέχει περισσότερους από δύο άρτιους
- Διασχίζει τον πίνακα και ελέγχει κάθε στοιχείο αν το mod 2 == 0 (άρα είναι άρτιος) και αυξάνει κατά 1 την evens
- Τελικά επιστρέφει true αν evens > 2, αν δηλαδή βρέθηκαν περισσότεροι από δύο άρτιοι

Περιττοί


```java
/**
 * Traverses an array to decide if the array
 * involves more than two odd numbers.
 *
 * @param arr   the source array.
 * @return      true, if the array contains
 *              more than two odds, false otherwise.
 */
public static boolean moreThanTwoOdds(int[] arr) {
    if (arr == null) return false;
    int odds = 0;
    for (int item : arr) {
        if (item % 2 != 0) {
            odds++;
        }
    }
    return odds > 2;
}
```

- Ελέγχει τον πίνακα για το αν περιέχει περισσότερους από δύο περιττούς
- Διασχίζει τον πίνακα και ελέγχει κάθε στοιχείο αν το mod 2 != 0 (άρα είναι περιττός) και αυξάνει κατά 1 την odds (odds++)
- Επιστρέφει true αν odds > 2, αν δηλαδή βρέθηκαν περισσότεροι από δύο περιττοί

Συνεχόμενοι


```java
/**
 * Traverses an array to decide if the array
 * involves more than two consecutive numbers.
 *
 * @param arr   the source array.
 * @return      true, if the array contains
 *              more than two consecutive, false otherwise.
 */
public static boolean moreThanTwoConsecutive(int[] arr) {
    if (arr == null) return false;
    int cons = 0;
    for (int i = 0; i < arr.length - 2; i++) {
        if ((arr[i] == arr[i+1] - 1) && (arr[i] == arr[i+2] - 2)) {
            cons++;
        }
    }
    return cons >= 1;
}
```

- Ελέγχει τον πίνακα για το αν περιέχει περισσότερους από δύο συνεχόμενους αριθμούς (π.χ. 5, 6, 7 – τρεις συνεχόμενοι)
- Διασχίζει τον πίνακα μέχρι i < length - 2 (μιας και κοιτάμε 2 στοιχεία μπροστά) και ελέγχει για κάθε στοιχείο αν έχει σχέση με το επόμενο (μικρότερο κατά 1) και το μεθεπόμενο (μικρότερο κατά 2)
- Τελικά επιστρέφει true αν cons >= 1, αν δηλαδή βρέθηκε μία έστω συνεχόμενη τριάδα

Λήγοντες


```java
/**
 * Checks if the array contains more than two
 * number with same ending.
 *
 * @param arr    the source array.
 * @return       true, if the array contains more than
 *               two numbers with same ending, false otherwise.
 */
public static boolean moreThanTwoSameEndings(int[] arr) {
    if (arr == null) return false;
    int[] endings = new int[10];
    boolean hasSameEndings = false;

    for (int item : arr) {
        endings[item % 10]++;
    }
    for (int item : endings) {
        if (item >= 3) {
            hasSameEndings = true;
            break;
        }
    }
    return hasSameEndings;
}
```

- Ορίζει ένα βοηθητικό πίνακα (endings) 10 θέσεων από 0 έως 9 όσοι και λήγοντες
- Διασχίζει τον αρχικό πίνακα και σε κάθε αντίστοιχη θέση του endings -- endings[item % 10] -- αυξάνει κατά 1. Άρα για κάθε στοιχείο του αρχικού πίνακα βρίσκει τον λήγοντα (item % 10) και μετά πάει στην αντίστοιχη θέση του endings και αυξάνει κατά 1
- Διατρέχει στην συνέχεια τον endings και αν σε κάποια θέση (λήγοντα) υπάρχει τιμή >= 3 σημαίνει ότι είχαν βρεθεί 3 ή περισσότερα στοιχεία με τον ίδιο λήγοντα και επιστρέφει true, αλλιώς false

Άσκηση

- Ζητήστε από τον χρήστη να εισάγει έξι ακεραίους από 1 έως 49 και ελέγξτε αν αυτή η εξάδα περνάει από τα παρακάτω φίλτρα (predicates)
    1. Δεν έχει πάνω από 3 άρτιους
    2. Δεν έχει πάνω από 3 περιττούς
    3. Δεν έχει πάνω από 3 συνεχόμενους
    4. Δεν έχει πάνω από 3 αριθμούς με τον ίδιο λήγοντα
    5. Δεν έχει πάνω από 3 αριθμούς στην ίδια δεκάδα