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

Σύνθετοι Τύποι Δεδομένων Πίνακες (Arrays)

Αθ. Ανδρούτσος

Primitives vs References

- Η Java αλλά και όλες οι γλώσσες προγραμματισμού κάνουν διάκριση ανάμεσα στους απλούς ή **πρωταρχικούς τύπους δεδομένων (primitive data types)** και στους **σύνθετους τύπους δεδομένων (reference types)**
- Οι **πρωταρχικοί τύποι** δεδομένων στην Java είναι οι: int, byte, short, long, float, double, char, boolean
- Οι ***σύνθετοι τύποι δεδομένων*** αποτελούνται από πρωταρχικούς τύπους ή από σύνθετους

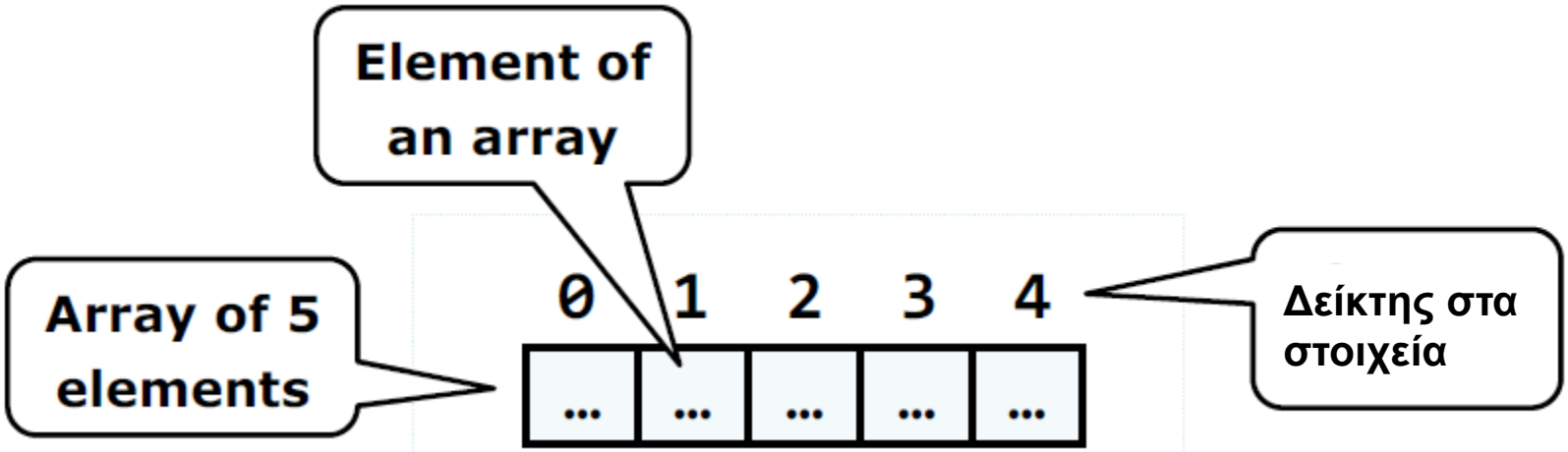
Πίνακες (1)

- Οι Πίνακες (arrays) είναι ένας σύνθετος τύπος δεδομένων στην Java που περιλαμβάνει μία **ακολουθία τιμών του ίδιου τύπου** (π.χ. πίνακας ακεραίων ή πίνακας από boolean ή πίνακας από πίνακες –δισδιάστατος πίνακας- κλπ.)
- Σε πιο τεχνικούς όρους, **Πίνακας (array)** είναι μία **διατεταγμένη ακολουθία τιμών του ίδιου τύπου δεδομένων**

Πίνακες (2)

- Διατεταγμένη ακολουθία σημαίνει ότι **υπάρχει διάταξη,** δηλαδή υπάρχει 1ο στοιχείο του πίνακα, που βρίσκεται στην 1η θέση του πίνακα, 2ο στοιχείο του πίνακα που βρίσκεται στη 2η θέση του πίνακα, κλπ.
- Η αναφορά στα στοιχεία του πίνακα γίνεται με **ακέραιους δείκτες**, δηλαδή ακέραιες τιμές που συμβολίζουν τη θέση του στοιχείου μέσα στον πίνακα (ξεκινούν από το 0, που είναι η 1η θέση του πίνακα)

Σύνθετες δομές – Πίνακες



- Όλα τα στοιχεία του πίνακα ***είναι του ίδιου τύπου*** (πρωταρχικού ή σύνθετου)
- Κάθε στοιχείο είναι σε μία θέση του πίνακα. **Οι θέσεις των πινάκων στην Java ξεκινούν από το 0**

Πίνακες

- Οι πίνακες είναι επομένως γραμμικές (linear) δομές δεδομένων (είναι διατεταγμένη ακολουθία, υπάρχει σχέση προηγούμενου-επόμενου στοιχείου)
- Το πλήθος των ακέραιων δεικτών ενός πίνακα ονομάζεται διάσταση (dimension) του πίνακα
- **Το πρώτο στοιχείο κάθε πίνακα στην Java ξεκινάει από τη θέση μηδέν**
- Η προσπέλαση ενός στοιχείου σε ένα πίνακα μπορεί να γίνει άμεσα με τη χρήση του ακέραιου δείκτη του

Μήκος του Πίνακα

- Έστω πίνακας με πέντε στοιχεία, έστω δηλαδή ότι το μήκος του πίνακα είναι ***length = 5***
- Η αρίθμηση των στοιχείων ενός πίνακα ***ξεκινάει πάντα από το μηδέν*** και πάει μέχρι το **length – 1**
- Επομένως αν έχουμε 5 στοιχεία, το 1ο στοιχείο, όπως είπαμε βρίσκεται στη θέση 0 και το τελευταίο στοιχείο βρίσκεται στη θέση 4 του πίνακα

Μνήμη – Stack και Heap (1)

- Το JVM (Java Virtual Machine) χωρίζει από λογική σκοπιά τη μνήμη που διαχειρίζεται σε δύο τμήματα: τη ***Στοίβα (Stack)*** και το ***Σωρό (Heap)***
- Η Στοίβα χρησιμοποιείται για στατική δέσμευση χώρου δηλαδή δέσμευση κατά το χρόνο μεταγλώττισης (@compile time) και ο Σωρός για δυναμική δέσμευση χώρου δηλαδή κατά το χρόνο εκτέλεσης (@runtime)

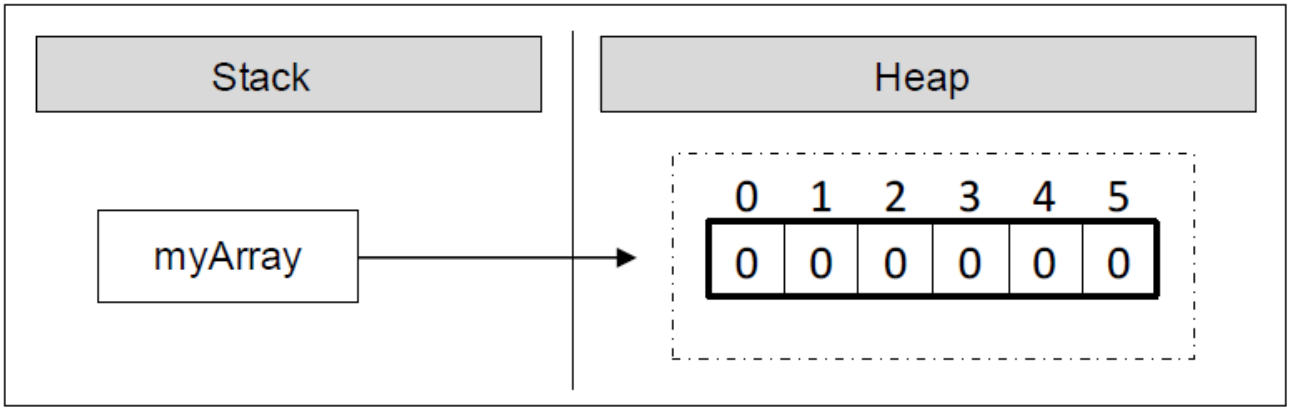
Μνήμη – Stack και Heap (2)

- Οι πρωταρχικοί τύποι δεδομένων (primitives) στη Java **δεσμεύουν χώρο στατικά, δηλαδή κατά το χρόνο μεταγλώττισης, στο Stack**
- Οι σύνθετοι (ή αναφορικοί – references) τύποι δεσμεύουν χώρο στο Stack μόνο για την αναφορική μεταβλητή, **ενώ τα περιεχόμενα αποθηκεύονται δυναμικά, δηλαδή κατά το χρόνο εκτέλεσης, στο Σωρό (Heap)**

Δήλωση και Δημιουργία Πίνακα


```
int[] myArray = new int[6];
```

Δηλώνουμε μία μεταβλητή ***myArray*** τύπου ***πίνακα ακεραίων. Ο πίνακας συμβολίζεται με [] μετά το όνομα του τύπου*** Δηλώνουμε τη ***myArray*** και αρχικοποιούμε με τη ***new***

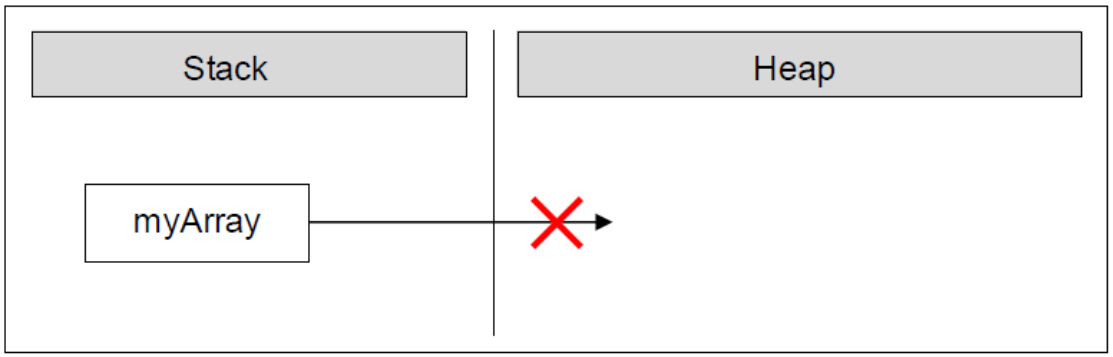


- Η ***new*** δεσμεύει (allocates) στο Heap χώρο για τον πίνακα (εδώ 6 θέσεις) και επιστρέφει στη ***myArray ως τιμή τη θέση μνήμης*** του πίνακα (για την ακρίβεια τη θέση μνήμης της 1ης θέσης του πίνακα, δηλαδή της θέσης 0)

Δήλωση Πίνακα


```
int[] myArray;
```

Έστω η δήλωση χωρίς αρχικοποίηση, δηλαδή χωρίς new.

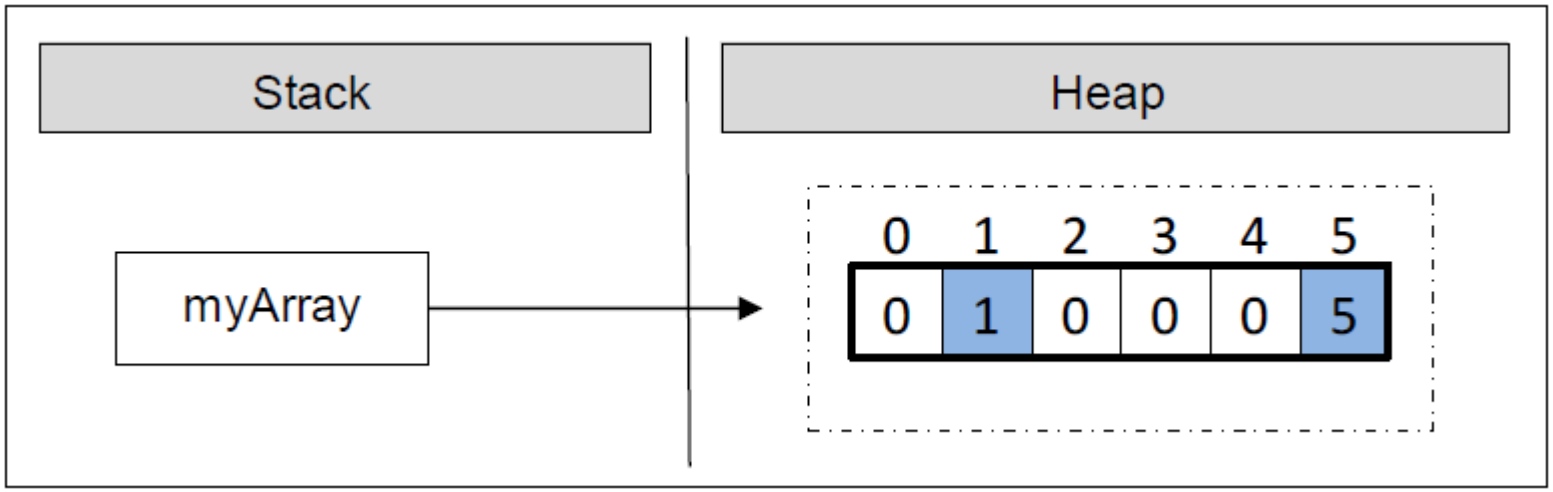


- Η παραπάνω δήλωση δεσμεύει στο Stack την αναφορική μεταβλητή (δείκτη) ***myArray,*** κατά το χρόνο μεταγλώττισης. Τα περιεχόμενα των αναφορικών μεταβλητών είναι διευθύνσεις μνήμες (γιαυτό λέγονται και δείκτες γιατί 'δείχνουν' σε μία θέση μνήμης)
- Η παραπάνω δήλωση ***δηλώνει αλλά δεν αρχικοποιεί*** την ***myArray***, οπότε η τιμή της είναι **null** κατά τη στιγμή της δήλωσης, δηλαδή δεν 'δείχνει' πουθενά

Πρόσβαση σε θέσεις Πίνακα

- Πρόσβαση σε μία θέση ενός πίνακα έχουμε με τον τελεστή indexer [], π.χ. **myArray[θέση]**
- Οι θέσεις ξεκινάνε από το 0

```
int[] myArray = new int[6];
myArray[1] = 1;
myArray[5] = 5;
```



Δήλωση πίνακα και populate

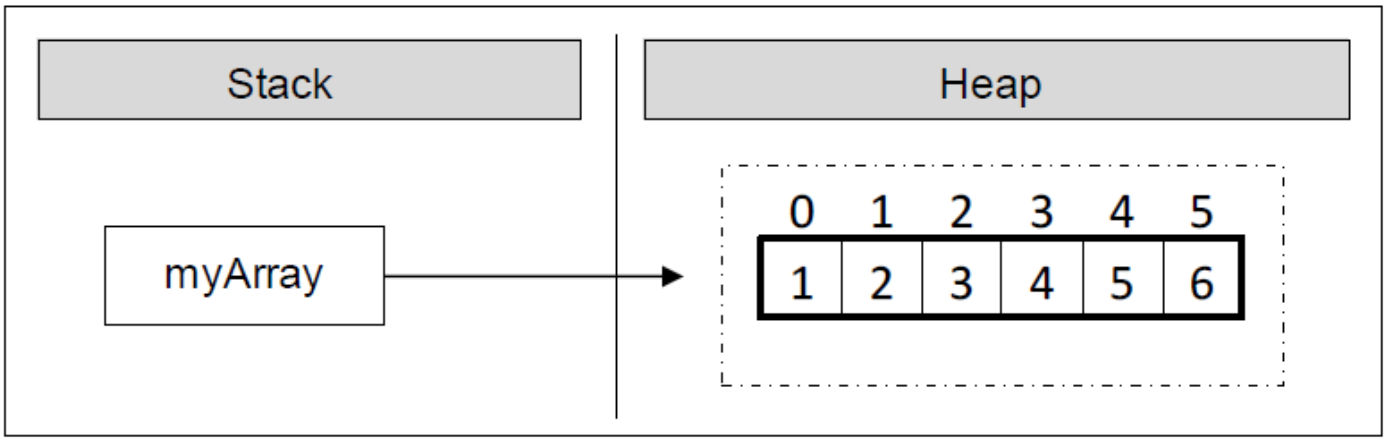

```java
package gr.aueb.cf.ch6;

/**
 * Δήλωση πίνακα ακεραίων 3 θέσεων,
 * populate (αρχικοποίηση τιμών) και
 * εκτύπωση τιμών.
 */
public class ArrayApp {

    public static void main(String[] args) {
        int[] arr = new int[3];
        arr[0] = 5;
        arr[1] = 7;
        arr[2] = 12;

        System.out.println("arr[0] = " + arr[0] + ", arr[1] = "
                + arr[1] + ", arr[2] = " + arr[2]);
    }
}
```

- Δηλώνουμε ένα πίνακα 3 θέσεων ακεραίων και αρχικοποιούμε (populate) δίνοντας τις τιμές 5, 7, και 12
- Οι default αρχικές τιμές σε πίνακα ακεραίων είναι μηδέν

Unsized δημιουργία πίνακα (1)

- «Έμμεση» δέσμευση χώρου μπορεί να γίνει με unsized initialization δηλαδή ***δεν δηλώνουμε διάσταση (άρα δεν χρησιμοποιούμε τη new), παρά μόνο αρχικοποιούμε μέσα σε { }***
- Αυτόματα ο μεταγλωττιστής δημιουργεί πίνακα n θέσεων, όσα και τα στοιχεία (6 θέσεων στο παρακάτω παράδειγμα)

```
int[] myArray = { 1, 2, 3, 4, 5, 6 };
```



Unsized Array Init (2)


```
1  package gr.aueb.cf.ch6;
2
3  /**
4   * Unsized array init.
5   */
6  public class UnsizedInitApp {
7
8      public static void main(String[] args) {
9          int[] arr = {1, 4, 8, 2, 12};
10
11         System.out.println(arr[0]);
12         System.out.println(arr[1]);
13         System.out.println(arr[2]);
14         System.out.println(arr[3]);
15         System.out.println(arr[4]);
16     }
17 }
```

- Μέσα σε {} δίνουμε λίστα τιμών (int literals) διαχωρισμένα με κόμμα

Array Initializer


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

/**
 * Array initializer.
 */
public class ArrayInit {

    public static void main(String[] args) {
        int[] arr = new int[] {1, 2, 3, 4, 5};

        int[] arr2;
        arr2 = new int[] {6, 7, 8, 9, 10};
    }
}
```

- Με array initializer μπορούμε και **σε μία γραμμή αλλά και σε δύο γραμμές** να αρχικοποιήσουμε ένα array
- Ενώ με unsized init μπορούμε μόνο σε μία γραμμή

Πίνακες Αναλυτικά

- Πίνακες χρησιμοποιούμε όταν υπάρχουν δεδομένα που έχουν συνάφεια μεταξύ τους, όπως για παράδειγμα οι πωλήσεις ανά μήνα μίας επιχείρησης ή οι βαθμοί των μαθητών μιας τάξης, τότε χρειαζόμαστε μία δομή δεδομένων που να μπορεί να χειριστεί αυτά τα δεδομένα συνολικά ως μια δομή και όχι ως μεμονωμένες μεταβλητές

Πίνακες – Σύνθετη δομή

- Οι πίνακες **σε αντίθεση με τους πρωταρχικούς τύπους δεδομένων**, είναι σύνθετος τύπος δεδομένων
- Οι σύνθετοι τύποι δεδομένων καταλαμβάνουν πολύ χώρο στην μνήμη και στην Java αναφέρονται ως **Αναφορικοί Τύποι (Reference Types) σε αντίθεση με τους πρωταρχικούς τύπους δεδομένων (Primitive Types)** και όπως είδαμε χρησιμοποιούμε την new για να δεσμεύσουμε χώρο καθώς και αναφορικές μεταβλητές (δείκτες) για να αναφερθούμε σε αυτούς

Μονοδιάστατοι Πίνακες

- Όπως είδαμε δηλώνουμε ένα πίνακα
  σαν κανονική μεταβλητή
  αλλά με τη χρήση του τελεστή []
  είτε `int[] a;` – που είναι προτιμότερο
  ή `int a[];` (που είναι C-Style)

```
int[] a = new int[3];
a[0]=0;
a[1]=1;
a[2]=2;
```

- Δέσμευση χώρου στη μνήμη
  - (1) Με τον **τελεστή new** ακολουθούμενο από τον τύπο δεδομένων και το πλήθος των στοιχείων που θέλουμε να δεσμεύσουμε (Τρία στοιχεία σε αυτό το παράδειγμα)
  - Αναφορά στα περιεχόμενα των θέσεων και εκχώρηση τιμών με a[0], a[1], ..., a[9]
  - **1η θέση πάντα η θέση 0 και το 1ο στοιχείο a[0]**

Ο τελεστής new

- Επειδή οι πίνακες είναι μία δομή που περιλαμβάνει πολλές θέσεις στη μνήμη (στοιχεία του πίνακα) θα πρέπει οι θέσεις αυτές να δεσμευτούν
- Η **δέσμευση χώρου στη μνήμη γίνεται με την εντολή new**
- Η εντολή new δεσμεύει χώρο στη μνήμη **δυναμικά δηλαδή κατά το χρόνο εκτέλεσης**

Δείκτες και αναφορές (1)

- Δείκτης ή αναφορά στις γλώσσες προγραμματισμού ονομάζεται μία μεταβλητή που περιέχει ως περιεχόμενο μία διεύθυνση μνήμης, άρα αναφέρεται ή 'δείχνει' σε κάποιο σημείο της μνήμης

Δείκτες και αναφορές (2)

- Οι δείκτες/αναφορές έχουν το πλεονέκτημα ότι μπορούμε να αναφερθούμε σε μεγάλες δομές δεδομένων **μόνο με έναν δείκτη στην δομή** βελτιστοποιώντας το χώρο που απαιτείται για να γίνουν διάφορες πράξεις

Δείκτες και αναφορές (3)

- Στην Java παρότι δεν υπάρχουν δείκτες με τον τρόπο που υπάρχουν στην C, δηλαδή δεν μπορούμε να δηλώσουμε μεταβλητές δείκτη άμεσα, ωστόσο υπάρχουν και χρησιμοποιούνται έμμεσα
- Αυτό συμβαίνει στις αναφορικές δομές (reference types) όπως οι πίνακες

Reference Types

- Το όνομα κάθε πίνακα είναι στην πραγματικότητα δείκτης που δείχνει στην 1η θέση του πίνακα
- Αυτό έχει το πλεονέκτημα ότι **για να αναφερθούμε σε ολόκληρη τη δομή του πίνακα, μπορούμε να το κάνουμε αναφερόμενοι απλά στο όνομά του**, δηλαδή στον δείκτη που δείχνει στον πίνακα

Πράξεις επί Πινάκων

- Αρχικοποίηση ή αλλιώς Γέμισμα (**Populate**)
- Διάβασμα (**Read**) των στοιχείων και εκτύπωση
- Ενημέρωση (**Update**) ενός στοιχείου
- Αναζήτηση (**search**) ενός στοιχείου

Αρχικοποίηση με εισαγωγή τιμών - Populate

- **Άμεσα** μέσα στον κώδικα
  - Με ***new[n] και στη συνέχεια με εκχωρήσεις στοιχείων στις θέσεις του Πίνακα*** (η 1η θέση είναι η θέση 0)
  - Με ***unsized initialization***
  - Με **array initializer**, δηλαδή με new χωρίς διάσταση και αρχικοποίηση με { }

Αρχικοποίηση με εκχωρήσεις


```java
package testbed.ch6;

public class ArrayPopulate {

    public static void main(String[] args) {

        // Array declaration and init
        int[] ages = new int[5];

        // Populate array
        ages[0] = 20;
        ages[1] = 40;
        ages[2] = 27;
        ages[3] = 17;
        ages[4] = 22;

        // Print the array elements
        for (int i = 0; i < ages.length; i++) {
            System.out.print(ages[i] + " ");
        }
    }
}
```

- Ο τελεστής ***new*** δεσμεύει χώρο στη μνήμη (για την ακρίβεια σε μια περιοχή της μνήμης που ονομάζεται "Σωρός" (Heap))
- Η δέσμευση γίνεται «δυναμικά» δηλ. κατά τον χρόνο εκτέλεσης
- Την στιγμή που εκτελείται η new όλα τα στοιχεία του Πίνακα έχουν τιμή 0
- Στη συνέχεια τα αρχικοποιούμε σε νέες τιμές με ρητό τρόπο μέσω πρόσβασης στη θέση του κάθε στοιχείου

Αρχικοποίηση με unsized init


```
1  package gr.aueb.than.ch6;
2
3  /**
4   * Αρχικοποιεί ένα πίνακα ακραίων
5   * με unsized initialization.
6   *
7   * @author A. Androutsos
8   */
9  public class ArrayPopulateUnsized {
10
11     public static void main(String[] args) {
12         // Initialize - Populate the array
13         int[] ages = {19, 20, 23, 22, 30};
14
15         // Print the array elements
16         for (int i = 0; i < ages.length; i++) {
17             System.out.print(ages[i] + " ");
18         }
19     }
20 }
```

```
Run:  ArrayPopulateUnsized
"C:\Program Files\Java\jdk1.8.0_40\bin\java.exe" ...
19 20 23 22 30
Process finished with exit code 0
```

- Αρχικοποιούμε τον πίνακα ή αλλιώς τον γεμίζουμε (**populate**) άμεσα με unsized initialization
- Στη συνέχεια εκτυπώνουμε με for
- Το μήκος (μέγεθος) του πίνακα επιστρέφεται από την ιδιότητα ***.length*** <όνομα_πίνακα>.length
- Παρατηρούμε ότι η for ξεκινάει από το 0 μιας και η 1η θέση του πίνακα είναι η ages[0] και φτάνει μέχρι το < ages.length δηλαδή μέχρι το ages.length – 1 που είναι το τελευταίο στοιχείο του πίνακα

Array Initializer


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

/**
 * Το unsized init μπορεί να χρησιμοποιηθεί
 * μόνο κατά τη στιγμή της δήλωσης ενώ το
 * array initializer είναι πιο ευέλικτο.
 */
public class ArrayInitializerApp {

    public static void main(String[] args) {
        int[] ages;

        // Array initializer
        ages = new int[] {1, 2, 3, 4};

        for (int i = 0; i < ages.length; i++) {
            System.out.print(ages[i]);
        }
    }
}
```

- Η αρχικοποίηση με array initializer είναι πιο γενικός μηχανισμός που μπορεί να χρησιμοποιηθεί μετά τη δήλωση σε αντίθεση με το array initialization που γίνεται κατά τη στιγμή της δήλωσης

Εκτύπωση στοιχείων Πίνακα

- Για να διασχίσουμε (traverse) **ένα πίνακα και γενικά μια γραμμική δομή**, χρησιμοποιούμε μία ***FOR***
- Ένα πίνακα τον διασχίζουμε ***από το στοιχείο 0 μέχρι το στοιχείο length – 1*** και εκτυπώνουμε κάθε στοιχείο στην κονσόλα
- Η κλασική FOR **μπορεί να οδηγήσει σε λάθη αν πάμε να προσπελάσουμε έξω από τα όρια του πίνακα** (για παράδειγμα αν διασχίσουμε μέχρι length αντί μέχρι length - 1)
- Υπάρχει και μία άλλη μορφή της FOR που μπορούμε να χρησιμοποιήσουμε ώστε **να αναφερόμαστε στις θέσεις των στοιχείων με ασφάλεια** (βλ. επόμενη διαφάνεια)

For-Each Loop


```java
package testbed.ch6;

/**
 * Διασχίζει και εκτυπώνει τα στοιχεία ενός
 * πίνακα με for-each.
 */
public class ArrayForeachApp {

    public static void main(String[] args) {
        int[] ages = {19, 29, 22, 40};

        for (int age : ages) {
            System.out.print(age + " ");
        }
    }
}
```

- Η for-each είναι παρόμοια με την κλασσική FOR αλλά πιο **ασφαλής** στις περιπτώσεις που δεν χρειαζόμαστε τη θέση κάθε στοιχείου αλλά το στοιχείο καθαυτό
- Σημαίνει πως για ***κάθε τιμή - age τύπου int- μέσα στον πίνακα ages*** εκτύπωσε κάθε τιμή
- Το age είναι μία μεταβλητή που μετακινείται αυτόματα και δείχνει μία-μία σε όλες τις τιμές/θέσεις του πίνακα

Αρχικοποίηση με Scanner από το πληκτρολόγιο


```java
package gr.aueb.than.ch6;

import java.util.Scanner;

/**
 * Αρχικοποιεί με Scanner από το πληκτρολόγιο.
 *
 * A. Androutsos
 */
public class ArrayPopulateKeyboard {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int[] ages = new int[5];

        for (int i = 0; i<ages.length; i++) {
            System.out.println("Δώστε αριθμό για τη θέση: " + (i+1));
            /* Δίνουμε i+1 για να είναι πιο φιλικό προς τον χρήστη
             * Ο χρήστης δεν καταλαβαίνει τις θέσεις από 0 - 4 αλλά
             * από 1 - 5
             */
            ages[i] = sc.nextInt();
        }

        for (int i = 0; i < ages.length; i++) {
            System.out.print(ages[i] + " ");
        }

        sc.close();
    }
}
```

- Εδώ κάνουμε populate το array διαβάζοντας τις τιμές από τον χρήστη μέσω Scanner
- Στη συνέχεια εκτυπώνουμε

Αρχικοποίηση από αρχείο


```java
package gr.aueb.than.ch6;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

/**
 * Αρχικοποιεί τον πίνακα διαβάζοντας από ένα αρχείο.
 *
 * @author A. Androutsos
 */
public class ArrayPopulateFile {

    public static void main(String[] args) throws FileNotFoundException {
        int[] ages = new int[5];
        File intFile = new File("C:/THANASSIS/myIntFile.txt");
        Scanner sc = new Scanner(intFile);

        for (int i = 0; i<ages.length; i++) {
            ages[i] = sc.nextInt();
        }

        for (int age : ages) {
            System.out.print(age + " ");
        }

        sc.close();
    }
}
```

- Αντιστοιχούμε τον Scanner σε μια μεταβλητή αρχείου (*intFile*), που πιο πριν έχουμε δηλώσει (ως File) και αντιστοιχήσει σε ένα φυσικό αρχείο, στο **C:/THANASSIS/myIntFile.txt**
- Κάνουμε populate τον πίνακα με τις τιμές του αρχείου και εκτυπώνουμε
- Τα file paths στην Java μπορούμε να τα δίνουμε σε platform-independent format, με forward slash (“/”)
- Επίσης για Windows θα μπορούσαμε να δώσουμε C:\\THANASSIS\\myIntFile.txt
- Δηλαδή σε Windows format κάνοντας escape το \ επειδή είναι ειδικός χαρακτήρας

Πίνακες ως παράμετροι μεθόδων


```
public static void printArray(int[] arr) {
    for (int item : arr) {
        System.out.print(item + " ");
    }
}
```

- Κατά τον ορισμό της μεθόδου περνάμε ως παράμετρο ένα πίνακα int[]. Όπως έχουμε πει στην Java **όλες οι παράμετροι μεθόδων περνάνε πάντα κατά τιμή (by value)**, δηλαδή αντιγράφονται σε τοπικές μεταβλητές της μεθόδου
- Στην περίπτωση ωστόσο των σύνθετων δομών δεδομένων, όπως οι πίνακες, αυτό που περνάει κατά τιμή είναι ο δείκτης (το reference) του πίνακα. Τα περιεχόμενα περνάνε by reference. Επομένως το μόνο που αντιγράφεται στο scope της μεθόδου είναι **μία διεύθυνση μνήμης** (το όνομα του πίνακα) που δείχνει στα περιεχόμενα του πίνακα.
- Περνώντας by reference τα περιεχόμενα του πίνακα μπορούμε να κάνουμε αλλαγές σε αυτά γιατί τα ίδια τα περιεχόμενα δεν αντιγράφονται στο scope της μεθόδου, μόνο ο δείκτης προς αυτά αντιγράφεται

Κλήση μεθόδου


```
public static void main(String[] args) {
    int[] ages = {20, 25, 30, 33, 45, 19, 90, 55};

    printArray(ages);
```

- Παρατηρούμε ότι κατά την κλήση της μεθόδου printArray(ages) ως πραγματική παράμετρο περνάμε το όνομα του πίνακα, που είναι απλά ένας δείκτης προς τα περιεχόμενα του πίνακα

Υπερφορτωμένη έκδοση (1)


```
public static void printArray(int[] arr, int low, int high) {
    if ((low < 0)  || (high > arr.length - 1)) return;

    for (int i = low; i <= high; i++) {
        System.out.print(arr[i] + " ");
    }
}
```

- Μπορούμε να υπερφορτώσουμε (overload) την printArray() με διαφορετικές τυπικές παραμέτρους, ώστε να μπορεί να εκτυπώνει όχι μόνο ολόκληρο τον πίνακα όπως η προηγούμενη printArray(int[] arr) αλλά και μέρος του πίνακα, από low μέχρι high

Υπερφορτωμένη έκδοση (2)


```
1   package testbed.ch7;
2
3   public class StringVsCharArrayApp {
4
5       public static void main(String[] args) {
6           int[] ages = {20, 25, 30, 33, 45, 19, 90, 55};
7
8           printArray(ages);
9           System.out.println();
10          printArray(ages, 1, ages.length - 2);
11      }
```

- Κάνουμε invoke τις υπερφορτωμένες εκδόσεις μέσα στην main

Αλλαγή στοιχείων του πίνακα με μέθοδο


```java
package testbed.ch6;

public class ReplaceWithMethod {

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

        upscaleByOne(grades);
        printArray(grades);
    }

    public static void upscaleByOne(int[] arr) {
        for (int i = 0; i < arr.length; i++) {
            arr[i] += 1;
        }
    }

    public static void printArray(int[] arr) {
        for (int item : arr) {
            System.out.print(item + " ");
        }
    }
}
```

- Όπως αναφέραμε οι πίνακες (δηλαδή τα περιεχόμενα των πινάκων) και γενικά οι σύνθετοι τύποι δεδομένων περνάνε ως παράμετροι μεθόδων **by reference** (οι ίδιες οι αναφορές περνάνε by value)
- Επομένως, η ίδια η αναφορά δεν μπορεί να αλλάξει μετά την έξοδο από τον πίνακα, αλλά οι τιμές του πίνακα μπορούν να αλλάξουν

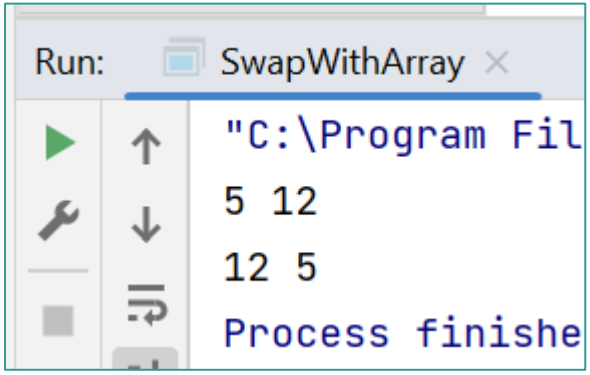
Swap με πίνακα (1)


```java
public static void swap(int[] arr) {
    if (arr.length != 2) return;

    int tmp = arr[0];
    arr[0] = arr[1];
    arr[1] = tmp;
}
```

- Η swap αντιστρέφει τις τιμές των θέσεων arr[0] και arr[1]
- Οι τιμές ανταλλάσσονται πραγματικά μιας και τα περιεχόμενα του πίνακα περνάνε by reference

Swap με πίνακα (2)


```java
package testbed.ch6;

public class SwapWithArray {

    public static void main(String[] args) {
        int[] arr = {5, 12};
        printArray(arr);
        System.out.println();
        swap(arr);
        printArray(arr);
    }

    public static void swap(int[] arr) {
        if (arr.length != 2) return;

        int tmp = arr[0];
        arr[0] = arr[1];
        arr[1] = tmp;
    }

    public static void printArray(int[] arr) {
        for (int item : arr) {
            System.out.print(item + " ");
        }
    }
```



- Όπως βλέπουμε οι τιμές του πίνακα έχουν αλλάξει αμοιβαία

By Value vs by reference

- Επομένως παρότι η Java και όλες οι γλώσσες προγραμματισμού υποστηρίζουν πέρασμα παραμέτρων κατά τιμή (**by value**), μπορούμε να περνάμε και αναφορές (διευθύνσεις μνήμης) σε σύνθετους τύπους δεδομένων κάτι το οποίο ονομάζεται και ως πέρασμα παραμέτρων κατ' αναφορά (**by reference**) κάτι το οποίο επιτρέπει οι όποιες αλλαγές στα στοιχεία που δείχνουν οι αναφορές να διατηρούνται και μετά την έξοδο από τη μέθοδο
- Επαναλαμβάνουμε όμως ότι οι ίδιες οι αναφορές περνάνε κατά τιμή και επομένως οι ίδιες οι αναφορές δεν αλλάζουν μετά την έξοδο από τη μέθοδο

Αναζητήσεις / Αντικαταστάσεις

- Αναζητήσεις. Μία πολύ βασική πράξη στους πίνακες είναι η αναζήτηση ενός στοιχείου
  - Αναζήτηση απλού στοιχείου
  - Αναζήτηση μικρότερου στοιχείου
  - Αναζήτηση μεγαλύτερου στοιχείου

Αναζήτηση στοιχείου σε Πίνακα (1)

- Ο βασικός τρόπος αναζήτησης είναι μέσα σε μία FOR ψάχνοντας για ένα συγκεκριμένο στοιχείο μέσα στον πίνακα με συγκρίσεις
- Μπορεί να υπάρχουν και παραλλαγές, όπως αναζήτηση του μικρότερου ή μεγαλύτερου στοιχείου

Αναζήτηση στοιχείου σε Πίνακα (2)


```java
/**
 * Searches for a key number in an array of ints.
 */
public class ArraySearchApp {

    public static void main(String[] args) {
        final int KEY = 10;
        boolean keyIsFound = false;
        int[] arr = {1, 5, 8, 9, 10, 15};

        for (int item : arr) {
            if (item == KEY) {
                keyIsFound = true;
                break;
            }
        }

        if (keyIsFound) {
            System.out.println("Key was found");
        } else {
            System.out.println("key was not found");
        }
    }
}
```

- Κάνουμε συγκρίσεις κάθε στοιχείου του πίνακα με το στοιχείο που αναζητούμε
- Η for εκτελείται μέχρι να έχουμε **item == KEY**, κάτι το οποίο συμβαίνει όταν item == 10

Αναζήτηση στοιχείου με μέθοδο


```java
public class SearchArrayWithMethod {

    public static void main(String[] args) {
        int[] quantities = {100, 200, 300};
        int position = 0;
        int value = 100;

        position = getElementPosition(quantities, value);

        System.out.printf("Position: %d, Value: %d", position, quantities[position]);
    }

    public static int getElementPosition(int[] arr, int value) {
        if (arr == null) return -1;

        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == value) {
                return i;
            }
        }

        // if value was not found
        return -1;
    }
}
```

- Η βασική ιδέα είναι η μέθοδος να επιστρέφει **τη θέση του στοιχείου στον πίνακα ή -1** αν το στοιχείου δεν βρεθεί

Update στοιχείου πίνακα

- Αντικαταστάσεις. Η αντικατάσταση (update / replace) ενός συγκεκριμένου στοιχείου του πίνακα μπορεί να γίνει **αφού πρώτα αναζητήσουμε και βρούμε το στοιχείο**

Αναζήτηση και αντικατάσταση


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

/**
 * Replaces all occurrences of KEY in an array of ints.
 */
public class ArrayUpdateApp {

    public static void main(String[] args) {
        final int KEY = 10;
        boolean keyIsFound = false;
        int[] arr = {1, 5, 8, 9, 10, 15};

        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == KEY) {
                arr[i] = KEY * 2;
            }
        }

        for (int item : arr) {
            System.out.print(item + " ");
        }
    }
}
```

- Με την κλασσική for αναζητούμε και αντικαθιστούμε
- Με enhanced for δεν μπορούμε να αντικαταστήσουμε γιατί το item δεν είναι τα πραγματικά στοιχεία του πίνακα, αλλά ένας δείκτης σε αυτά

Αντικατάσταση στοιχείου με μέθοδο (1)


```java
package testbed.ch6;

public class FindAndReplaceWithMethod {

    public static void main(String[] args) {
        int[] arr = new int[] {1, 2, 3, 4, 5, 6, 7, 8, 9};

        replace(arr, 9, 10);
        printArray(arr);
    }

    public static void replace(int[] arr, int oldValue, int newValue) {
        int positionToUpdate = -1;

        if (arr == null) return;
        positionToUpdate = getElementPosition(arr, oldValue);
        if (positionToUpdate != -1) arr[positionToUpdate] = newValue;
    }
```

- Η replace χρησιμοποιεί εσωτερικά την getElementPosition() (βλ. επόμενη διαφάνεια)

Αντικατάσταση στοιχείου με μέθοδο (2)


```
20      public static int getElementPosition(int[] arr, int value) {
21          if (arr == null) return -1;
22
23          for (int i = 0; i < arr.length; i++) {
24              if (arr[i] == value) {
25                  return i;
26              }
27          }
28
29          // if value was not found
30          return -1;
31      }
32
33  @   public static void printArray(int[] arr) {
34          for (int item : arr) {
35              System.out.print(item + " ");
36          }
37      }
38  }
```

- Η getElementPosition() επιστρέφει τη θέση στον πίνακα του στοιχείου με τιμή value
- Αν το στοιχείο δεν υπάρχει επιστρέφει -1
- Η printArray(int[] arr) διασχίζει και εκτυπώνει τα στοιχεία του πίνακα arr

Αντικατάσταση με το επόμενο


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

/**
 * Moves elements one position to the left.
 * Last element becomes zero. Essentially,
 * replaces each array element with the next one.
 */
public class ArrayReplaceNextApp {

    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};

        for (int i = 0; i <= arr.length - 2; i++) {
            arr[i] = arr[i+1];
        }
        arr[arr.length - 1] = 0;

        for (int element : arr) {
            System.out.print(element + " ");
        }
    }
}
```

- Αντικαθιστά κάθε στοιχείο i με το i + 1
- Άρα η for πρέπει να πάει μέχρι το προτελευταίο στοιχείο (length - 2), το οποίο πρέπει να αντικατασταθεί με το τελευταίο (length – 1)
- Το τελευταίο στοιχείο (length – 1) γίνεται 0

Replace with next – Method (1)


```java
/**
 * Replaces each array element with the next one
 * starting at a particular position.
 *
 * @param arr       the source array
 * @param low       the position to start
 */
public static void shiftLeftByOne(int[] arr, int low) {
    if (arr == null) return;
    if ((low < 0) || (low > arr.length - 1)) return;

    for (int i = low; i < arr.length - 1; i++) {
        arr[i] = arr[i+1];
    }

    arr[arr.length-1] = 0;
}
```

- Η μέθοδος παίρνει ως παράμετρο τον πίνακα και τη θέση από την οποία ξεκινάει να αντικαθιστά

Replace with next – Method (2)


```
1   package testbed.ch6;
2
3   public class ReplaceWithNextWithMethod {
4
5       public static void main(String[] args) {
6           int[] arr = {1, 3, 5, 7, 9};
7
8           shiftLeftByOne(arr, 0);
9           printArray(arr);
10      }
```

```
Run:  ReplaceWithNextWithMethod
"C:\Program Files\Amazon Corretto\jdk1
3 5 7 9 0
Process finished with exit code 0
```

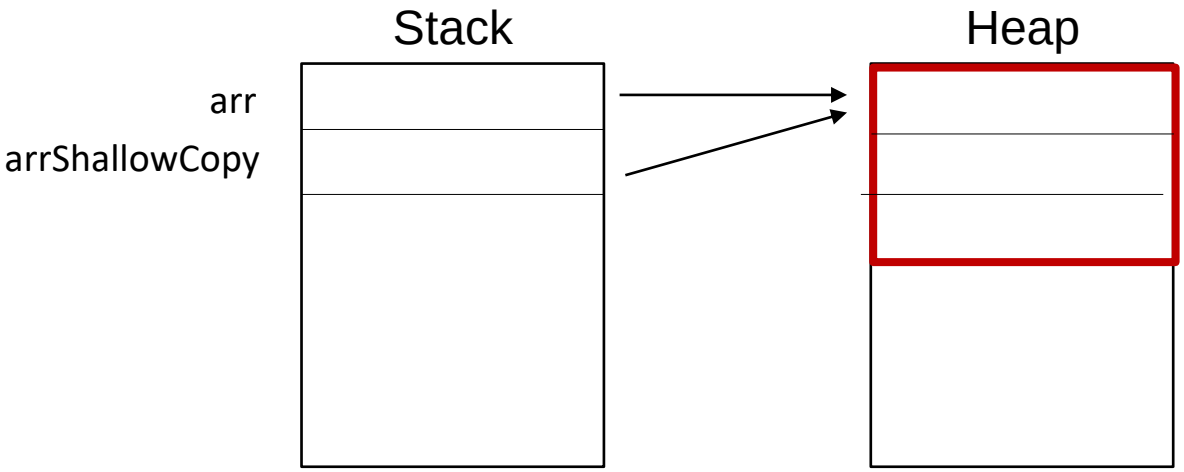
- Στην main ως πραγματική παράμετρο περνάμε τον πίνακα *arr* και τη θέση 0 ώστε να ξεκινήσει η αντικατάσταση από την 1η θέση

Shallow Copy (1)


```
1   package testbed.ch6;
2
3   public class ArrayShallowCopy {
4
5       public static void main(String[] args) {
6           int[] arr = {1, 2, 3, 4};
7           int[] arrShallowCopy;
8
9           arrShallowCopy = arr;
10          arrShallowCopy[0] = 100;
11
12          /*
13           * Side effect. arrShallowCopy changes
14           * its elements, but arr elements are
15           * changed as well. This happens because
16           * we copied the reference not the contents
17           * of the source array. So, we end up with two
18           * references point to the same array.
19           * So, arrShallowCopy is not a 'real' copy,
20           * it is a shallow copy
21           */
22          printArray(arr);
23      }
```

- Το *arrShallowCopy* είναι ένας πίνακας (μία αναφορά για την ακρίβεια)
- Στην γραμμή 9 αντιγράφουμε τον arr στον *arrShallowCopy*, αλλά δεν αντιγράφουμε τα περιεχόμενα, αλλά μόνο την αναφορά
- Αυτό έχει το side-effect ότι ό,τι αλλαγές κάνουμε στον *arrShallowCopy* γίνονται και στον *arr* και το αντίθετο μιας και οι δύο αναφορές (*arr* και *arrShallowCopy*) δείχνουν στα ίδια περιεχόμενα πίνακα (βλ. επόμενη διαφάνεια)

Shallow Copy (2)



- Οι δύο δείκτες (references), δηλαδή τα ονόματα των πινάκων δείχνουν μετά την αντιγραφή, (arrShallowCopy = arr;) στην ίδια θέση στο Heap που βρίσκονται τα περιεχόμενα του αρχικού πίνακα *arr*
- Επομένως οποιεσδήποτε αλλαγές γίνονται από το **arr** ή το **arrShallowCopy** επηρεάζουν και τους ‘δύο’ πίνακες που στην πραγματικότητα είναι ένας

Array Copy


```
1   package testbed.ch6;
2
3   public class ArrayCopy {
4
5       public static void main(String[] args) {
6           int[] arr = {1, 2, 3, 4};
7           int[] arrayCopy = new int[4];
8
9           for (int i = 0; i < arr.length; i++) {
10              arrayCopy[i] = arr[i];
11          }
12
13          printArray(arrayCopy);
14      }
15
16      public static void printArray(int[] arr) {
17          for (int item : arr) {
18              System.out.print(item + " ");
19          }
20          System.out.println();
21      }
22  }
```

- Δημιουργούμε πρώτα ένα νέο πίνακα ***arrayCopy με new*** και στη συνέχεια κάνουμε copy ένα-ένα (με μία for) τα στοιχεία του πίνακα *arr* στον *arrayCopy*
- Επομένως δεν έχουμε το προηγούμενο πρόβλημα που περιέχει στοιχεία int και γενικά θα μπορούσε να περιέχει στοιχεία οποιουδήποτε πρωταρχικού τύπου

Array copy με έτοιμες κλάσεις

- Η Java παρέχει δύο κλάσεις που κάνουν array copy
  - Την κλάση **Arrays** και τις μεθόδους *.copyOf* και *.copyOfRange*
  - Την κλάση **System** και την μέθοδο *.arraycopy*

Κλάσεις: Arrays και System


```java
import java.util.Arrays;

public class ArrayCopyWithMethods {

    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        int[] arrayCopy = new int[5];

        System.arraycopy(arr, 0, arrayCopy, 0, arr.length);
        printArray(arrayCopy);

        arrayCopy = Arrays.copyOf(arr, arr.length);
        printArray(arrayCopy);

        arrayCopy = Arrays.copyOfRange(arr, 0, arr.length);
        printArray(arrayCopy);
    }

    public static void printArray(int[] arr) {
        for (int item : arr) {
            System.out.print(item + " ");
        }
        System.out.println();
    }
}
```

- Η *System.arraycopy()* παίρνει πέντε παραμέτρους, τον source array, τη θέση που ξεκινάμε από τον source array, τον destination array, την θέση από την οποία ξεκινάμε στον destination array και το πλήθος των στοιχείων που αντιγράφουμε. Αν το πλήθος των στοιχείων είναι μεγαλύτερο από το length του src ή dest array δημιουργείται *IndexOutOfBoundsException*
- Η *Arrays.copyOf()* παίρνει δύο παραμέτρους, τον source array και το πλήθος των στοιχείων που θα αντιγραφούν. Επιστρέφει ένα νέο πίνακα. Αν το πλήθος των στοιχείων είναι μεγαλύτερο από το length του array γίνεται right padding με zeroes, αν είναι μικρότερο, γίνεται truncate (αποκοπή)
- Η *Arrays.copyOfRange()*, είναι σαν την *Arrays,copyOf()* αλλά δίνουμε start position (inclusive) και last position (exclusive)

Διαγραφή στοιχείου με την Arrays.copyOf


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

import java.util.Arrays;

public class ArrayReplaceNextArraysClassApp {

    public static void main(String[] args) {
        int[] ages = {20, 22, 23, 24, 26};

        // Truncates one item at RHS (Right Hand Side)
        ages = Arrays.copyOf(ages, ages.length - 1);

        // Adding and Padding with zero one position at RHS
        ages = Arrays.copyOf(ages, ages.length + 1);

        for (int age : ages) {
            System.out.print(age + " ");
        }
    }
}
```

- Κάνουμε διαγραφή του τελευταίου στοιχείου δίνοντας ως πλήθος στοιχείων length – 1
- Ταυτόχρονα επιστρέφουμε το αποτέλεσμα στον ίδιο πίνακα
- Αυτό υλοποιείται με ένα ενδιάμεσο πίνακα, στον οποίο δείχνει στη συνέχεια ο ages
- Αν θέλουμε να διατηρήσουμε το αρχικό length αντιγράφουμε ξανά με πλήθος length + 1 οπότε γίνεται right padding με zero

Ελάχιστο στοιχείο αταξινόμητου πίνακα


```java
/**
 * Finds the min value and min position of an unsorted array
 */
public class ArrayMinApp {

    public static void main(String[] args) {
        int[] arr = {7, 6, 2, 9, 10, 4, 6, 6, 7};

        // Έστω ότι το ελάχιστο στοιχείο είναι στη θέση 0
        int minPosition = 0;
        int minValue = arr[minPosition];

        // Ξεκινάμε να ελέγχουμε από τη θέση 1. Κάθε φορά που
        // βρίσκουμε μικρότερο στοιχείο από το minValue,
        // ανανεώνουμε minPosition και minValue
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] < minValue) {
                minPosition = i;
                minValue = arr[minPosition];
            }
        }

        System.out.printf("Min Value: %d, Min Position: %d", minValue, minPosition + 1);
    }
}
```

- Η βασική ιδέα είναι ότι διασχίζουμε ολόκληρο τον πίνακα και ελέγχουμε κάθε στοιχείο αν είναι μικρότερο από το τρέχον μικρότερο (minValue)
- Κάθε φορά που βρίσκουμε ένα μικρότερο, ανανεώνουμε τα minPosition και minValue

Ελάχιστο στοιχείο (χωρίς comments)


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

/**
 * Finds the min value and min position of an unsorted array
 */
public class ArrayMinApp {

    public static void main(String[] args) {
        int[] arr = {7, 6, 2, 9, 10, 4, 6, 6, 7};
        int minPosition = 0;
        int minValue = arr[minPosition];

        for (int i = 1; i < arr.length; i++) {
            if (arr[i] < minValue) {
                minPosition = i;
                minValue = arr[minPosition];
            }
        }
        System.out.printf("Min Value: %d, Min Position: %d", minValue, minPosition + 1);
    }
}
```

- Η εύρεση του ελάχιστου γίνεται με συγκρίσεις κάθε στοιχείου του πίνακα με το minValue

Ελάχιστο στοιχείο – Αλγόριθμος με maxint


```java
/**
 * Finds the min element of an array of ints.
 * The initial min value is set to Integer.MAX_VALUE
 * and the initial position to 0.
 */
public class ArrayMinApp2 {

    public static void main(String[] args) {
        int[] arr = {4, 6, 3, 8, 9, 8, 2, 11};

        // Ορίζουμε ως min value το max-int, οπότε κάποιο στοιχείο θα είναι
        // μικρότερο από max-int εκτός εάν όλα τα στοιχεία του πίνακα είναι max-int,
        // οπότε τότε το position παραμένει 0, που είναι σωστό.
        int minValue = Integer.MAX_VALUE;
        int minPosition = 0;

        for (int i = 0; i < arr.length; i++) {
            if (arr[i] < minValue) {
                minPosition = i;
                minValue = arr[i];
            }
        }
        System.out.printf("Min Value: %d, Min Position: %d", minValue, minPosition + 1);
    }
}
```

- Ο αλγόριθμος θεωρεί αρχική ελάχιστη τιμή ένα πολύ μεγάλο ακέραιο (Integrer.MAX_VALUE)
- Ελέγχει κάθε στοιχείο του πίνακα
- Ελέγχει και το 1ο στοιχείο του πίνακα (i = 0), οπότε κάνει μία επιπλέον σύγκριση σε σχέση με τον προηγούμενο αλγόριθμο που θεωρεί αρχική τιμή του minValue το arr[0] και ξεκινάει με i = 1

Ελάχιστο πίνακα - Μέθοδος


```java
/**
 * Returns the position of the min value in a source array.
 *
 * @param arr       the source array
 * @param low       starting position in the array
 * @param high      ending position in the array
 * @return          the position of the array containing the least element value
 */
public static int getMinPosition(int[] arr, int low, int high) {
    int minPosition = low;
    int minValue;

    if ((low < 0) || (high > arr.length - 1)) {
        System.out.println("Error in array dimensions");
        return -1;      // Returns an invalid position
    }

    minValue = arr[low];
    for (int i = low; i <= high; i++) {
        if (arr[i] < minValue) {
            minPosition = i;
            minValue = arr[i];
        }
    }

    return minPosition;
}
}
```

- Πρόκειται για γενική μέθοδο που βρίσκει το ελάχιστο στοιχείο ενός πίνακα
- Παίρνει ως παραμέτρους τον πίνακα *arr* και τα *low* και *high (*start και end positions), τον subarray δηλαδή μέσα στον οποίο ψάχνει

Main Method


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

/**
 * Finds the min pos and val of an int-array
 * based on a generic method.
 *
 * @author a8ana
 */
public class ArrayMinGenericMethodApp {

    public static void main(String[] args) {
        int[] grades = {4, 8, 10, 3, 6, 2, 1, 4};
        int minPosition = 0;

        minPosition = getMinPosition(grades, 0, grades.length - 1);

        // Κάνουμε μία διόρθωση στο minPosition (+1, οπότε ) για να είναι user-friendly
        System.out.printf("Min value: %d, Min position: %d", grades[minPosition], minPosition + 1);
    }
```

- Καλούμε (invoke) την getMinPosition με πραγματικές παραμέτρους grades, 0, grades.length - 1

Min & Max μαζί


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

/**
 * Finds thw min/max element position and the
 * corresponding element value in an array of ints.
 */
public class MinMaxApp {

    public static void main(String[] args) {
        int[] grades = new int[] {7, 6, 3, 9, 10, 4, 6, 6, 7};
        int minPosition = 0;
        int maxPosition = 0;
        int minValue = grades[0];
        int maxValue = grades[0];

        for (int i = 1; i < grades.length; i++) {
            if (grades[i] < minValue) {
                minPosition = i;
                minValue = grades[i];
            }

            if (grades[i] > maxValue) {
                maxPosition = i;
                maxValue = grades[i];
            }

        System.out.printf("MinValue: %d, MinPosition: %d\n",
                minValue, minPosition + 1);
        System.out.printf("MaxValue: %d, MaxPosition: %d",
                maxValue, maxPosition + 1);
    }
}
```

- Η εύρεση του στοιχείου με τη μεγαλύτερη τιμή είναι παρόμοια με την min value
- Στη max value απλά αλλάζει ο έλεγχος σε grades[i] > maxValue

Κατανομή βαθμών και εμφάνιση με αστεράκια (1)


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

/**
 * Visualizes grades' distribution with stars.
 */
public class GradesDistributionApp {

    public static void main(String[] args) {
        int[] grades = {30, 9, 8, 14, 17, 22, 40, 55, 57, 59, 60, 61, 67, 68, 77, 72, 75, 80, 85, 93, 91, 100};
        int[] distribution = new int[10];

        for (int grade : grades) {
            if (grade == 100) distribution[9] += 1;
            else distribution[grade / 10] += 1;
        }

        for (int i = 0; i < distribution.length; i++) {
            System.out.printf("%02d-%02d", i * 10, (i != 9) ? (i * 10 + 9) : (i * 10 + 10));
            for (int j = 1; j <= distribution[i]; j++) {
                System.out.print("*");
            }
            System.out.println();
        }
    }
}
```

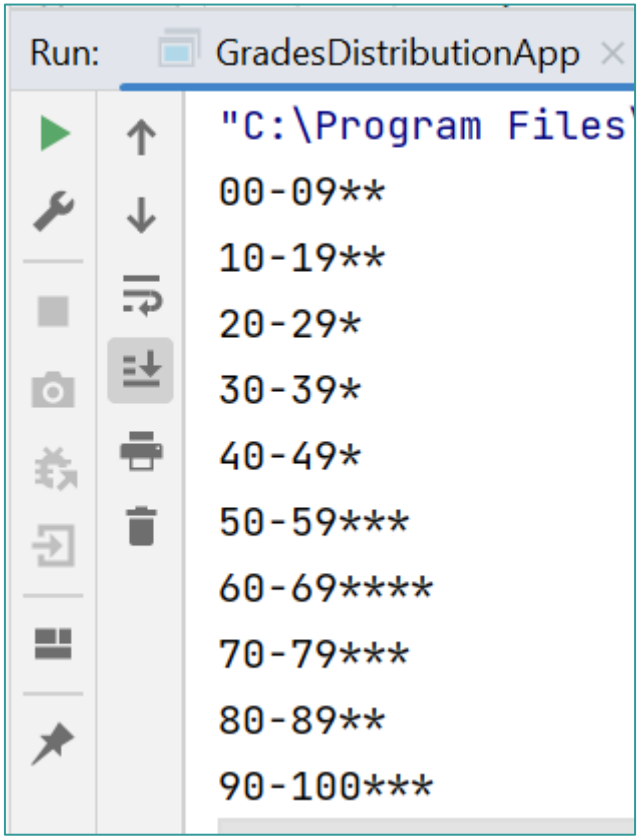
- Κάθε βαθμός ανήκει σε μία δεκάδα στη βαθμολογική περιοχή από 0 – 100 (10 δεκάδες). Ο πίνακας distribution έχει αυτές τις δεκάδες στις θέσεις 0 – 9
- Η 1η for εκχωρεί στον πίνακα distribution το πλήθος των βαθμών που ανήκουν στην ίδια δεκάδα (grade / 10). Ελέγχουμε για το 100 μιας και διαφέρει τεχνικά ο χειρισμός του (ανήκει στην δεκάδα που ξεκινάει με 9)

Κατανομή βαθμών και εμφάνιση με αστεράκια (2)


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

/**
 * Visualizes grades' distribution with stars.
 */
public class GradesDistributionApp {

    public static void main(String[] args) {
        int[] grades = {30, 9, 8, 14, 17, 22, 40, 55, 57, 59, 60, 61, 67, 68, 77, 72, 75, 80, 85, 93, 91, 100};
        int[] distribution = new int[10];

        for (int grade : grades) {
            if (grade == 100) distribution[9] += 1;
            else distribution[grade / 10] += 1;
        }

        for (int i = 0; i < distribution.length; i++) {
            System.out.printf("%02d-%02d", i * 10, (i != 9) ? (i * 10 + 9) : (i * 10 + 10));
            for (int j = 1; j <= distribution[i]; j++) {
                System.out.print("*");
            }
            System.out.println();
        }
    }
}
```

- Στη συνέχεια η 2η for κάνει traverse (διασχίζει) τον πίνακα distribution και σε κάθε γραμμή από τις 10 γραμμές του πίνακα, πρώτα τυπώνει την βαθμολογική περιοχή (που σχετίζεται με το i) και στη συνέχεια έχει μία εσωτερική for που τυπώνει τα αστεράκια (τόσα αστεράκια όσα και η αντίστοιχη τιμή του πίνακα – distribution[i]

Αποτελέσματα



- Τα αποτελέσματα (actual) είναι συμβατά με τα **αναμενόμενα αποτελέσματα (expected)** σύμφωνα με τα δοκιμαστικά δεδομένα που έχουμε δώσει

Πλήθος βαθμών στην κλίμακα 0 – 5 (1)


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

/**
 * Prints grades percentage. Grades exist in
 * the range 0 - 5.
 */
public class GradesPercentageApp {

    public static void main(String[] args) {
        int[] grades = new int[] {1, 2, 2, 4, 5, 5, 0, 2, 0, 4, 5, 2, 1, 1};
        int[] counts = new int[6];

        for (int grade : grades) {
            counts[grade]++;
        }

        for (int i = 0; i < counts.length; i++) {
            System.out.printf("Grade %d: %.2f%%\n", i, (double) counts[i] / grades.length );
        }
    }
}
```

- Για να εκτυπώσουμε τον χαρακτήρα % στην printf, τον γράφουμε ως %%

- Ο πίνακας counts έχει 6 θέσεις ώστε να είμαστε συμβατοί με τη βαθμολογία που είναι από 0 – 5
- Κάθε θέση του counts είναι το πλήθος εμφανίσεων στον πίνακα του αντίστοιχου βαθμού
- Στην 2η for διατρέχουμε τον πίνακα counts και υπολογίζουμε τα ποσοστά διαιρώντας με το grades.length

Διδιάστατοι Πίνακες (1)


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

/**
 * Defines a 2D array and populates it.
 * 2D arrays are defined as [rowCount][colCount]
 */
public class TwoDimArrayApp {

    public static void main(String[] args) {
        int[][] grid = new int[2][2];

        grid[0][0] = 0;     // 1η γραμμή, 1η στήλη
        grid[0][1] = 1;     // 1η γραμμή, 2η στήλη
        grid[1][0] = 2;     // 2η γραμμή, 1η στήλη
        grid[1][1] = 3;     // 2η γραμμή, 2η στήλη

        for (int[] row : grid) {
            for (int col : row) {
                System.out.print(col + " ");
            }
            System.out.println();
        }
    }
}
```

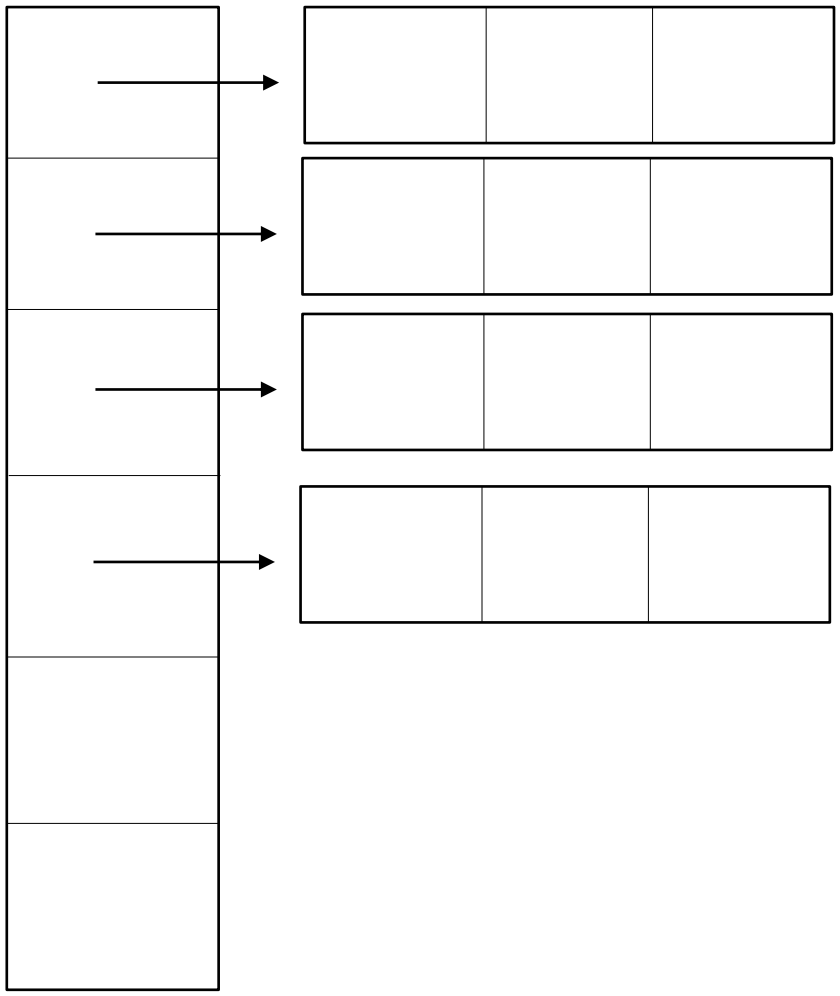
- Οι δισδιάστατοι πίνακες ορίζουν το πλήθος των γραμμών και των στηλών
- Στο παράδειγμα ορίζουμε ένα πίνακα δύο γραμμών και δύο στηλών
- Στη συνέχεια εκχωρούμε τιμές 0, 1, 2, 3 στις θέσεις του πίνακα
- Στις θέσεις του πίνακα αναφερόμαστε ως grid[row][column]

Unsized Init (2)


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

/**
 * Unsized 2D Array Initialization.
 */
public class ArrayUnsizedInitApp {

    public static void main(String[] args) {
        int[][] grid = {{1, 2},{3, 4},{5, 6}};  // 3x2

        for (int[] row : grid) {
            for (int col : row) {
                System.out.print(col + " ");
            }
            System.out.println();
        }
    }
}
```

- Μέσα σε { } είναι ο κάθε μονοδιάστατος πίνακας, δηλαδή η κάθε γραμμή του δισδιάστατου πίνακα
- Στο παράδειγμα, έχουμε ορίσει ένα δισδιάστατο πίνακα με τρεις γραμμές: όλες έχουν διάσταση 2.
  - Η 1η γραμμή έχει τιμές {1, 2}
  - Η 2η γραμμή {3, 4}
  - Η 3η γραμμή {5, 6}

Διδιάστατοι Πίνακες (2)



- Για ευκολία στην νοητική μας αναπαράσταση μπορούμε να θεωρούμε την τεχνική υλοποίηση που είναι ότι η **κάθε θέση του βασικού πίνακα είναι ένας πίνακας**

Jagged Arrays


```java
/**
 * Μπορούμε να δώσουμε διαφορετικές
 * διαστάσεις σε κάθε γραμμή του
 * δισδιάστατου πίνακα.
 */
public class JaggedArrayApp {

    public static void main(String[] args) {
        int[][] arr = new int[3][];

        // Σε κάθε γραμμή εκχωρούμε ένα
        // πίνακα με διαφορετική διάσταση
        arr[0] = new int[4];
        arr[1] = new int[3];
        arr[2] = new int[2];

        for (int[] row : arr) {
            for (int col : row) {
                System.out.print(col + " ");
            }
            System.out.println();
        }
    }
}
```

- Μπορούμε να δώσουμε σε κάθε γραμμή του δισδιάστατου πίνακα ως τιμή ένα μονοδιάστατο πίνακα με διαφορετική διάσταση
- Στο παράδειγμα η κάθε μία γραμμή από τις 3 γραμμές έχει διαφορετικό μήκος
  - Η $1^{η}$ γραμμή έχει μήκος 4 στοιχεία
  - Η $2^{η}$ γραμμή έχει μήκος 3 στοιχεία
  - Η $3^{η}$ γραμμή έχει μήκος 2 στοιχεία

Διδιάστατοι πίνακες - Συνοπτικά

- `int a[][] = new a[3][4];`

Δήλωση με [ ][ ] και new. Διδιάστατος πίνακας 3 γραμμές x 4 στήλες

- ```
  int a[][] = {{1, 55, 6},
               {7, 15, 26},
               {11, 15, 56},
               {61, 56, 64}};
  ```

Unsized δήλωση και αρχικοποίηση. Το πλήθος γραμμών/στηλών παρέχεται έμμεσα (4 γραμμές x 3 στήλες)

- ```
  int a[][] = new int [3][];
  a[0] = new int[10];
  a[1] = new int[5];
  a[2] = new int[7];
  ```

Γραμμές και στήλες
Αποθήκευση κατά γραμμές

a[0,0]	a[0,1]	a[0,2]
a[1,0]	a[1,1]	a[1,2]
a[2,0]	a[2,1]	a[2,2]
a[3,0]	a[3,1]	a[3,2]

Κάθε μία από τις 3 γραμμές έχει διαφορετικό μήκος (μέγεθος)

- Η 1η γραμμή έχει μήκος 10 στοιχεία
- Η 2η γραμμή έχει μήκος 5 στοιχεία
- Η 3η γραμμή έχει μήκος 7 στοιχεία

Εκτύπωση στοιχείων 2D πίνακα


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

/**
 * Different types of Array traverse.
 */
public class TraverseArrayApp {

    public static void main(String[] args) {
        int[][] arr = new int[][] {{1, 2}, {3, 4, 5}};

        for (int i = 0; i < arr.length; i ++) {
            for (int j = 0; j < arr[i].length; j++) {
                System.out.print(arr[i][j] + " ");
            }
            System.out.println();
        }

        for (int[] row : arr) {
            for (int j = 0; j < row.length; j++) {
                System.out.print(row[j] + " ");
            }
            System.out.println();
        }

        for (int row[] : arr) {
            for (int column : row) {
                System.out.print(column + " ");
            }
            System.out.println();
        }
    }
}
```

- Έστω ένα jagged array
- Για να εμφανίσουμε τα στοιχεία δισδιάστατων πινάκων χρειαζόμαστε ***δύο for μία μέσα στην άλλη***
- Παρουσιάζονται τρεις μορφές που κάνουν το ίδιο πράγμα:
    1. Με κλασσική for,
    2. Με enhanced-for και for και
    3. Με δύο enhanced-for

Ταξινόμηση (Selection Sort)

- Βρίσκουμε το ελάχιστο κάθε υποπίνακα και το ανεβάζουμε στη θέση του 1ου στοιχείου του υποπίνακα. Το 1ο στοιχείο του υποπίνακα το 'κατεβάζουμε' στη θέση του ελάχιστου. Άρα κάνουμε swap μεταξύ των δύο στοιχείων
- Αυτό το κάνουμε για κάθε υποπίνακα από τους n-1 υποπίνακες

Selection Sort


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

/**
 * Sorts an array with selection sort
 */
public class SelectionSort {

    public static void main(String[] args) {
        int[] arr = {5, 8, 3, 9, 4, 1, 2};
        int min;
        int minPosition;
        int tmp;

        for (int i = 0; i < arr.length - 1; i++) {

            // Find sub-array min
            min = arr[i];
            minPosition = i;
            for (int j = i + 1; j < arr.length; j++) {
                if (arr[j] < min) {
                    min = arr[j];
                    minPosition = j;
                }
            }

            // Swap sub-array min <-> top element of
            // sub-array (arr[i])
            tmp = arr[i];
            arr[i] = min;
            arr[minPosition] = tmp;
        }

        for (int el : arr) {
            System.out.print(el + " ");
        }
    }
}
```

- Η 1η for δείχνει στο 1ο στοιχείο κάθε υποπίνακα
- Η 2η for διατρέχει κάθε υποπίνακα από αυτούς που δείχνει το i της 1ης for
- Για κάθε υποπίνακα βρίσκει το μικρότερο και το κάνει swap με το 1ο στοιχείο του υποπίνακα

Ταξινόμηση - Bubble Sort

- Πρόκειται για παραλλαγή του Selection Sort
- Ανταλλάσσει αμοιβαία τα διαδοχικά στοιχεία των υποπινάκων, κατεβάζοντας το μεγαλύτερο στοιχείο προς τα κάτω σε κάθε υποπίνακα

Bubble Sort


```java
/**
 * Ταξινομεί ένα πίνακα με τη μέθοδο Bubble Sort
 */
public class BubbleSort {

    public static int[] arr = {2, 6, 9, 3, 1, 4, 3, 12};

    public static void main(String[] args) {

        for (int i = arr.length - 1; i > 0; i--) {
            for (int j = 0; j < i; j++) {
                if (arr[j] > arr[j+1]) swap(j, j+1);
            }
        }

        for (int element : arr) {
            System.out.println(element);
        }
    }

    public static void swap(int i, int j) {
        int tmp = arr[i];
        arr[i] = arr[j];
        arr[j] = tmp;
    }
```

- Το i δείχνει στο τελευταίο στοιχείο κάθε υποπίνακα
- Το j ξεκινά από το 1ο στοιχείο (index 0) και φτάνει μέχρι το i. Ανταλλάσσει αμοιβαία τα διαδοχικά στοιχεία, κατεβάζοντας το μεγαλύτερο στοιχείο προς τα κάτω

Πολυπλοκότητα Χρόνου (1)

- Από όλους τους αλγόριθμους που επιλύουν ένα πρόβλημα, ενδιαφέρουν οι "καλοί" αλγόριθμοι, δηλαδή οι πιο αποδοτικοί σε όρους χρόνου και χώρου.
- Ιδιαίτερα η πολυπλοκότητα χρόνου (time complexity) αποτελεί το βασικότερο κριτήριο αξιολόγησης των αλγορίθμων.
- Ένας αλγόριθμος είναι καλύτερος από κάποιον άλλο αν η μέση συμπεριφορά του σε όρους χρόνου είναι πιο αποδοτική, δηλαδή έχει μικρότερο χρόνο εκτέλεσης.
- Λέγοντας πολυπλοκότητα χρόνου (time complexity) δεν εννοείται ο απόλυτος χρόνος σε λεπτά, δευτερόλεπτα κλπ., γιατί τότε η συμπεριφορά του αλγόριθμου θα εξαρτιόταν από τις επιδόσεις του υπολογιστή που τον εκτελεί. Άλλωστε, στην αξιολόγηση των αλγορίθμων δεν ενδιαφέρει ο απόλυτος χρόνος εκτέλεσης, γιατί η έννοια του "καλύτερου" είναι σχετική.
- Πολυπλοκότητα χρόνου ενός αλγορίθμου είναι το πλήθος των βασικών πράξεων σε σχέση με τα δεδομένα εισόδου.

Πολυπλοκότητα Χρόνου (2)

- Η εύρεση του μικρότερου ή μεγαλύτερου στοιχείου στον Selection Sort απαιτεί n συγκρίσεις και αφού αυτό γίνεται n φορές (δύο nested for), η τελική πολυπλοκότητα χρόνου (Time Complexity) είναι $O(n^2)$. Το Big O notation σημαίνει στην χειρότερη περίπτωση
- Το ίδιο ισχύει και στον Bubble Sort, όπου πάλι η πολυπλοκότητα χρόνου είναι $O(n^2)$

Merge Sort (1)


```java
public static int[] mergeSort(int[] arr, int s, int n) {
    // if arr.length <= 1 return arr
    if (n - s <= 0) {
        return new int[] { arr[n] };
    }

    // Choose a pivot -> middle - easy split
    int middle = (n + s)  / 2;

    int[] left = mergeSort(arr, s, middle);
    int[] right = mergeSort(arr, middle + 1, n);

    return merge (left, right);
}
```

- Η mergeSort επιστρέφει ταξινομημένο πίνακα, χωρίζοντας τον αρχικό πίνακα αναδρομικά σε δύο μέρη, τα οποία ταξινομεί επίσης αναδρομικά
- Η mergeSort ταξινομεί με την χρήση της merge, την οποία θα δούμε στην επόμενη διαφάνεια

Merge Sort (2)


```
    public static int[] merge(int[] left, int[] right) {
        int[] merged = new int[left.length + right.length];
        int i = 0;
        int j = 0;
        int k = 0;

        while ((i < left.length) && (j < right.length)) {
            if (left[i] > right[j]) {
                merged[k++] = right[j++];
            } else {
                merged[k++] = left[i++];
            }
        }

        while (i < left.length) {
            merged[k++] = left[i++];
        }

        while (j < right.length) {
                merged[k++] = right[j++];
        }

        return merged;
    }
}
```

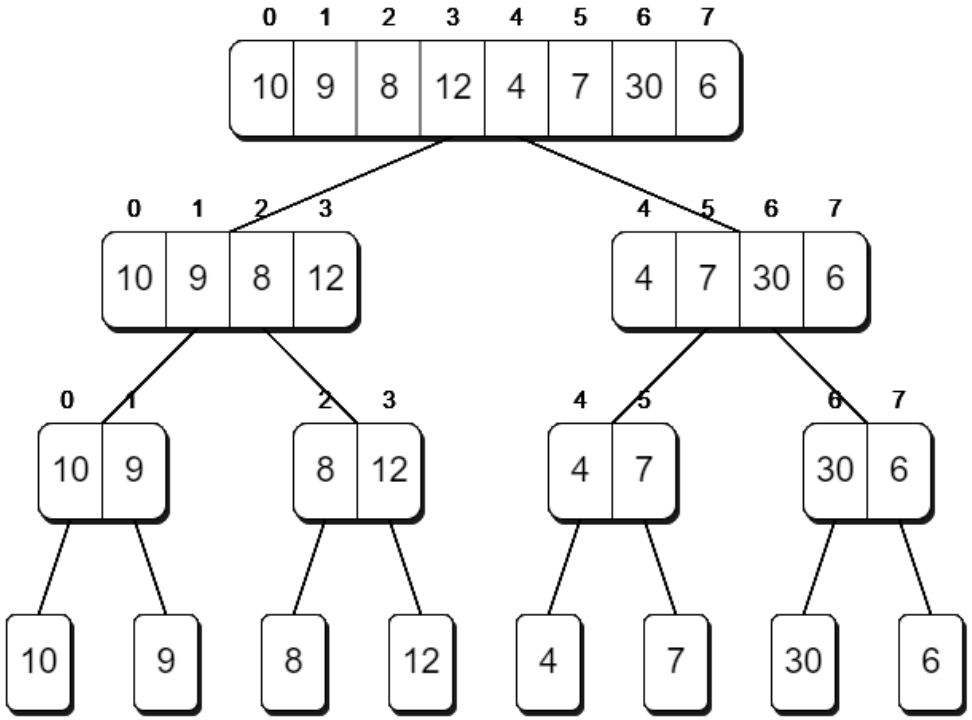
- Η merge κάνει απλά συγχώνευση δύο ταξινομημένων πινάκων
- Διατρέχει ταυτόχρονα τους δύο πίνακες και συγκρίνει τα στοιχεία τους
- Το εκάστοτε μικρότερο στοιχείο το εισάγει στην merged
- Αν εξαντληθούν τα στοιχεία κάποιου από τους δύο πίνακες, τότε απλά αντιγράφει τα εναπομείναντα στοιχεία στον merged

Merge Sort (3)


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

public class MergeSort {

    public static void main(String[] args) {
        int[] arr = new int[] {1, 2, 6, 3, 4, 12, 56, 34, 23, 11};

        int[] merged = mergeSort(arr, 0, arr.length - 1);

        for (int el : merged) {
            System.out.print(el + " ");
        }
    }
```

- Καλούμε την mergeSort στον αρχικό πίνακα
- Η πολυπλοκότητα χρόνου της mergeSort είναι O(nlogn) μιας και για κάθε ένα από τα logn segments (που προκύπτουν από τις διαδοχικές υποδιαιρέσεις του πίνακα) χρειάζεται γραμμικό χρόνο ταξινόμησης

Merge Sort (4)



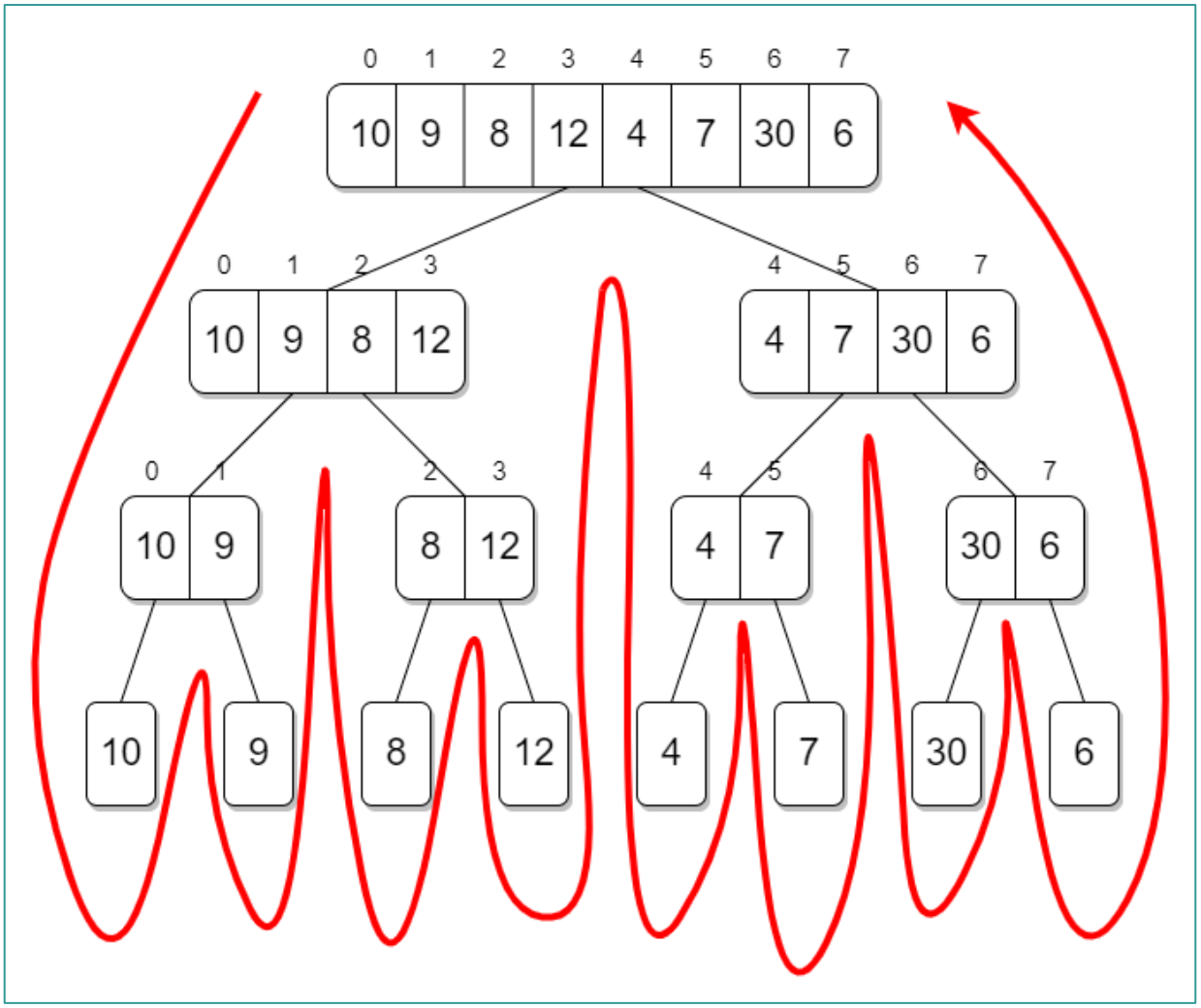
$$\text{middle} = \frac{0+7}{2} = 3$$

3 / 2 = 1 , 11 / 2 = 5

1 / 2 = 0 , 5 / 2 = 2 , 9 / 2 = 4 , 13 / 2 = 6

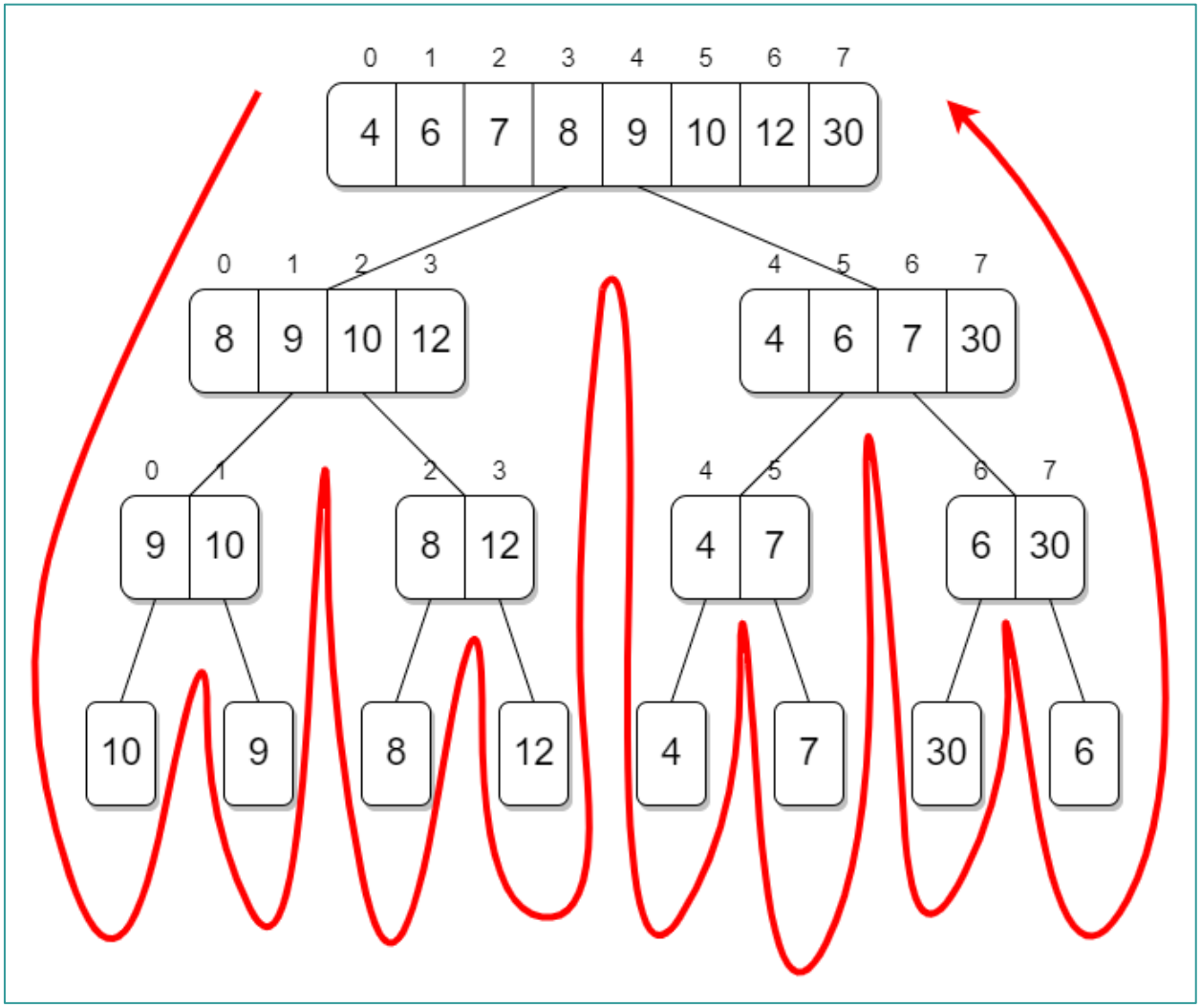
- Ο merge sort κάνει διαδοχικές κατατμήσεις και διαγραμματικά απεικονίζεται με ένα δένδρο αναδρομής

Merge Sort (5)



- Διαγραμματικά το δένδρο της αναδρομής διασχίζεται με Euler Tour, δηλαδή μεταδιατεταγμένη διάσχιση, Αριστερά-Δεξιά-Ρίζα.
- Ο merge sort δηλαδή στην κατάβαση (βλ. κόκκινο βέλος) κατατέμνει και στην άνοδο ταξινομεί αναδρομικά τα αριστερά υπόδενδρα και μετά τα δεξιά
- Για παράδειγμα αφού στο τελευταίο επίπεδο χωρίσει αριστερά κάτω σε 9 και 10, ο merge ταξινομεί και δίνει στην άνοδο τον [9,10] πίνακα που είναι ταξινομημένος

Merge Sort (6)



- Σε κάθε βήμα της ανάβασης ταξινομεί με merge τους δύο ήδη ταξινομημένους πίνακες (οι πίνακες με ένα στοιχείο θεωρούνται ήδη ταξινομημένοι)
- Ο τελικός χρόνος είναι n*logn αφού έχουμε logn κατατμήσεις (logn επίπεδα) και σε κάθε επίπεδο έχουμε γραμμικό χρόνο για την merge

Binary Search (1)

- Η δυαδική αναζήτηση, δηλαδή η αναζήτηση ενός στοιχείου σε ένα ταξινομημένο πίνακα, υλοποιείται με την επιλογή του μεσαίου στοιχείου του πίνακα και την σύγκριση με το στοιχείο που αναζητούμε
- Αν το στοιχείο βρεθεί ο αλγόριθμος σταματά διαφορετικά ξανακαλείται αναδρομικά, δηλαδή αν α αριθμός που ψάχνουμε είναι μικρότερος, τότε κάνουμε πάλι δυαδική αναζήτηση στο πάνω-μισό του πίνακα, αν είναι μεγαλύτερος κάνουμε αναζήτηση στο κάτω-μισό, κλπ., μέχρι να βρούμε το στοιχείο με διαδοχικές κατατμήσεις του πίνακα και συγκρίσεις του μεσαίου κάθε φορά στοιχείου

Binary Search (1)


```java
/**
 * Returns the position in a sorted array of
 * a certain value.
 *
 * @param value     the searched value
 * @param low       the start index
 * @param high      the end index
 * @return          the position if the value eas found, -1 otherwise
 */
public static int binarySearch(int[] arr, int value, int low, int high) {
    int mid = 0;

    if (arr == null) return -1;
    if ((low < 0) || (high > arr.length - 1)) return -1;
    if (high < low) return -1;

    mid = (low + high) / 2;

    if (value == arr[mid]) {
        return mid;
    }
    if (value < arr[mid]) {
        return binarySearch(arr, value, low, mid - 1);
    }
     else {
         return binarySearch(arr, value, mid + 1, high);
     }
}
}
```

- Αναδρομικά βρίσκει αν το arr[mid] == value, με διαδοχικές κατατμήσεις του πίνακα
- Κάνει logn κατατμήσεις
- Ο συνολικός χρόνος αναζήτησης είναι logn

Binary Search (2)


```java
package testbed.ch6;

public class BinarySearch {

    public static void main(String[] args) {
        int[] arr = {2, 4, 5, 6, 7, 9};
        int low = 0;
        int high = arr.length - 1;
        int position = -1;

        position = binarySearch(arr, 7, low, high);

        if (position == -1) {
            System.out.println("Value was not found");
        } else {
            System.out.println("Value was found in position " + (position + 1));
        }
    }
```

```
Run:  BinarySearch
"C:\Program Files\Amazon Corretto\
Value was found in position 5
```

- Καλεί την binarySearch αρχικά με low = 0 και high = arr.length – 1 για την τιμή 7

Symmetric Array (1)


```java
/**
 * Decides if an array is symmetric. The algorithm uses
 * a special form of for iteration with two indexes i, j.
 *
 * @param arr   the source array
 * @return      true if array is symmetric, false otherwise
 */
public static boolean isArraySymmetric(int[] arr) {
    boolean isSymmetric = true;

    for (int i = 0, j = arr.length - 1; i < j; i++, j--) {
        if (arr[i] != arr[j]) {
            isSymmetric = false;
            break;
        }
    }

    return isSymmetric;
}
```

- Ελέγχει αν ένας πίνακας ακεραίων είναι συμμετρικός δηλαδή αν διαβάζεται το ίδιο από την αρχή και το τέλος
- Χρησιμοποιεί μία for με δύο δείκτες i, j όπου ο i ξεκινάει από το μηδέν και αυξάνει, ενώ ο j ξεκινάει από το length–1 και μειώνεται
- Ο έλεγχος γίνεται μεταξύ των τιμών των θέσεων i, j
- Η for τρέχει όσο i < j

Symmetric Array (2)


```java
/**
 * Decides if an array is symmetric. The algorithm
 * considers an iterative procedure, that is, a for
 * loop that runs n / 2 times, where n = arr.length-1.
 *
 * @param arr   the source array
 * @return      true if array is symmetric, false otherwise
 */
public static boolean isArraySymmetric2(int[] arr) {
    boolean isSymmetric = true;
    int n = arr.length - 1;

    for (int i = 0; i < n / 2; i++) {
        if (arr[i] != arr[n - i]) {
            isSymmetric = false;
            break;
        }
    }

    return isSymmetric;
}
```

- Ένας άλλος αλγόριθμος που αποφασίζει αν ένας πίνακας ακεραίων είναι συμμετρικός είναι να τρέχει η for έως n/2, όπου n = arr.length-1
- Ο έλεγχος γίνεται μεταξύ των τιμών των θέσεων i και n-i

Symmetric Array - main


```java
package testbed.ch6;

/**
 * Ελέγχει αν ένα πίνακας ακεραίων είναι συμμετρικός
 * δηλαδή διαβάζεται το ίδιο από την αρχή και το τέλος.
 */
public class SymmetricArray {

    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 2, 1, 3};

        System.out.println("Array is symmetric: " + isArraySymmetric(arr));
        System.out.println("Array is symmetric: " + isArraySymmetric2(arr));
    }
```

Μικρές Εργασίες (1)

1. Αναπτύξτε μία γενική μέθοδο εύρεσης του μέγιστου ενός πίνακα getMaxPosition (int[] arr, int low, int high) που επιστρέφει τη θέση στον πίνακα arr του μέγιστου στοιχείου του πίνακα
2. Αναπτύξτε ένα πρόγραμμα που βρίσκει το 2ο μικρότερο στοιχείο ενός πίνακα, το στοιχείο δηλαδή που είναι αμέσως μεγαλύτερο από το ελάχιστο στοιχείο

Μικρές Εργασίες (2)

- Έστω ένας πίνακας με στοιχεία της επιλογής σας. Γράψτε μεθόδους που κάνουν τα παρακάτω
- 1. Αναζήτηση ενός στοιχείο στον πίνακα (Επιστρέφει την θέση του στοιχείου)
- 2. Φιλτράρισμα των ζυγών (επιστρέφει void, μόνο εκτύπωση των ζυγών)
- 3. Mapping κάθε στοιχείου του πίνακα με το διπλάσιο (η μέθοδος επιστρέφει πίνακα)
- 4. Έλεγχος αν υπάρχει έστω ένας θετικός αριθμός (η μέθοδος πρέπει να επιστρέφει boolean)
- 5. Έλεγχος αν όλα τα στοιχεία του πίνακα είναι θετικοί (η μέθοδος πρέπει να επιστρέφει boolean)