Αντικειμενοστραφής Προγραμματισμός

Β' εξάμηνο – Τμήμα Πληροφορικής – Ιόνιο Πανεπιστήμιο

Ενότητα 5

Πίνακες & Συλλογές

Δημήτρης Ρίγγας

Μηχανικός Η/Υ & Πληροφορικής, MSc, PhD

Ε.Δι.Π. Τμήματος Πληροφορικής – Ιόνιο Παν/μιο

riggas@ionio.gr

Ανάγκη για ομαδοποίηση ομοειδών αντικειμένων

Πίνακες (Arrays)

Δομή η οποία περιέχει ένα **σταθερό** πλήθος όμοιων αντικειμένων.

Συλλογές (Collections)

Δομές **μεταβλητού** πλήθους: `List`, `Set`, `Queue`, `Map` …

Πίνακες

Arrays

Πίνακες — Εισαγωγή

- Ένας πίνακας είναι ένα **ευρετήριο τιμών**.
- Όλα τα στοιχεία πρέπει να είναι **ίδιου τύπου**.
- Έχουν **σταθερό μέγεθος**.
- Μπορείτε να φτιάξετε πίνακα για κάθε είδους τιμή: `int`, `double`, `String`, κλπ.
- Οι Πίνακες είναι **αντικείμενα** (αναφερόμενοι τύποι, όχι primitives).
  - Μια μεταβλητή πίνακα είναι στην πραγματικότητα **αναφορά** στο αντικείμενο πίνακα.
- Τα στοιχεία μπορούν να είναι primitives **ή** αναφερόμενοι τύποι (συμπεριλαμβανομένων άλλων πινάκων).

Δήλωση & Δημιουργία πίνακα

Δήλωση — δεν δεσμεύεται μνήμη, η αναφορά είναι `null`:

```
int[]    nums;
String[] names;
```

Δημιουργία με `new` και προσδιορισμό μεγέθους:

```
nums  = new int[10];
names = new String[10];

int st_count = 100;
String[] students = new String[st_count]; // μεταβλητή ως μέγεθος
```

Αρχικοποίηση με `{ }` — μόνο κατά τη δήλωση:

```
String[] class_names = {"Sem-A", "Sem-B"}; // ✓ OK

int[] nums;
nums = {5, 6, 7}; // ✘ ΛΑΘΟΣ – δεν επιτρέπεται εκ των υστέρων
```

Ταυτόχρονη δημιουργία πολλών πινάκων

Όπως ακριβώς δημιουργούμε πολλές μεταβλητές σε μια δήλωση:

```
int x = 0, y = 1;
```

Αντίστοιχα και για πίνακες:

```
String[] names = new String[10], surnames = new String[10];
```

✓ Συνιστάται η χωριστή δήλωση για ευανάγνωστο κώδικα.

Αναφορά σε στοιχεία πίνακα

```java
int[] anArray;
anArray = new int[10];

anArray[0] = 100;   // πρώτο στοιχείο
anArray[1] = 200;   // δεύτερο στοιχείο

System.out.println("Element at index 0: " + anArray[0]);
System.out.println("Element at index 1: " + anArray[1]);
```

- **Στοιχείο (element):** κάθε αντικείμενο που περιέχεται σε πίνακα.
- **Δείκτης (index):** μη αρνητικός ακέραιος μεταξύ `0` και `μήκος - 1`.
- Το μήκος: `anArray.length` — **χωρίς παρενθέσεις!**
- Αναφορά εκτός ορίων → `ArrayIndexOutOfBoundsException` (σφάλμα χρόνου εκτέλεσης).

Αρχικοποίηση στοιχείων

Πίνακες primitives:

```
int[] nums = new int[10];
System.out.println(nums[0]); // εκτυπώνει: 0
```

Ο compiler αποδίδει προεπιλεγμένη τιμή τύπου (`0`, `false`, κλπ.).

Χωρίς δημιουργία → σφάλμα:

```
int[] nums;
System.out.println(nums[0]); // error: variable nums might not have been initialized
```

Πίνακες αντικειμένων:

```
String[] names = new String[10];
int[]    ages  = new int[10];
System.out.println(names[0] + " : " + ages[0]); // null : 0
```

Δημιουργούνται `null` αναφορές — **ο constructor ΔΕΝ καλείται αυτόματα**.

Αναφορά σε στοιχεία πίνακα με δομή επανάληψης

```java
int[] nums = new int[10];

for (int i = 0; i < 10; i++) {
    nums[i] = i * 100;
}

int i = 9;
while (i >= 0) {
    System.out.println(nums[i--]);
}
```

- Ως δείκτης επιτρέπεται κάθε έκφραση που ανάγεται σε `byte`, `short` ή `int`.
- **Δεν επιτρέπεται** `long`.

Αναφορά σε στοιχεία πίνακα αντικειμένων

```java
public class ArrayObjectsLoop {
    public static void main(String[] args) {
        Point[] points = new Point[5];
        Random rand = new Random();

        for (int i = 0; i < points.length; i++) {
            points[i] = new Point();
            points[i].x = rand.nextInt(10);
            points[i].y = rand.nextInt(10);
        }

        for (Point p : points) {
            System.out.println("Point " + p);
        }
    }
}

class Point {
    int x, y;
    public String toString() { return "(" + x + "," + y + ")"; }
}
```

Για κάθε στοιχείο πρέπει να κληθεί ο constructor (`new`) ξεχωριστά.

Πίνακες ως παράμετροι ή επιστρεφόμενες τιμές

```
public static Point[] fill(int size) { ... return spots; }

public static void shift(Point[] array) {
    Point p = array[0];
    for (int i = 0; i < array.length - 1; i++)
        array[i] = array[i + 1];
    array[array.length - 1] = p;
}
```

- Οι πίνακες είναι **αναφερόμενοι τύποι**.
- Περνώντας πίνακα σε μέθοδο, μεταφέρεται **αντίγραφο της αναφοράς** (pass-by-value of the reference).
- Η μέθοδος **μπορεί να αλλάξει** τα στοιχεία του αρχικού πίνακα.
- Ο πίνακας δεν περνιέται "με αναφορά" — μεταφέρεται η αναφορά ως τιμή.

Εκχώρηση πίνακα σε πίνακα

```
int[] arrayA = fill(5);
int[] arrayB = fill(3);

arrayA = arrayB; // αντιγραφή αναφοράς – συνώνυμο!

arrayA[0] = -1;  // αλλάζει και το arrayB[0]!
```

- `arrayA = arrayB` κάνει **αντιγραφή αναφοράς**, ΟΧΙ αντιγραφή περιεχομένων.
- Ο αρχικός πίνακας του `arrayA` χάνεται (garbage collected), αν δεν υπάρχει άλλη αναφορά.
- Για πραγματική αντιγραφή: χρησιμοποιείστε `Arrays.copyOf()` ή `Arrays.copyOfRange()`.

Έλεγχος ισότητας πινάκων

```
int[] arrayA = new int[] {1, 2, 3, 4, 5};
int[] arrayB = new int[] {1, 2, 3, 4, 5};
int[] arrayD = arrayA;

System.out.println(arrayA == arrayB); // false – διαφορετικά αντικείμενα
System.out.println(arrayA == arrayD); // true  – συνώνυμα
```

- `==` συγκρίνει **αναφορές**, όχι περιεχόμενα.
- Για σύγκριση περιεχομένων: `Arrays.equals(arrayA, arrayB)` (βλ. παρακάτω).

Μήκος πίνακα – `length`

Χρήση της προκαθορισμένης μεταβλητής `length` αντί literal σταθεράς:

```
int[] nums = new int[10];
for (int i = 0; i < nums.length; i++)
    nums[i] = i * 10;

int i = nums.length - 1;
while (i >= 0)
    System.out.println(nums[i--]);
```

Χρήσιμο και για τα ορίσματα γραμμής εντολών:

```
public static void main(String[] args) {
    System.out.println(args.length);
    for (int i = 0; i < args.length; i++)
        System.out.println(args[i]);
}
```

Εναλλακτική δομή `for-each`

✓ Εισήχθη στη **Java 5**.

Αντί της κλασικής:

```
for (int i = 0; i < nums.length; i++) {
    System.out.println("Element: " + nums[i]);
}
```

Χρήση της συντομότερης μορφής:

```
for (int element : nums) {
    System.out.println("Element: " + element);
}
```

Σύνταξη: `for (type ref_to_element : array) { STATEMENTS }`

✓ Δεν παρέχει πρόσβαση στον δείκτη `i` — χρησιμοποιείστε τη κλασική `for` όταν χρειάζεστε τον δείκτη.

Πολυδιάστατοι πίνακες

Δηλώνονται με περισσότερα `[]`:

```
int[][] a = new int[3][4];                  // 3×4

int[][] b = {{1, 2}, {3, 4}};               // 2×2 literal

int[][] c = new int[2][];                   // 2 γραμμές
c[0] = new int[5];                          // 5 στήλες στη γραμμή 0
c[1] = new int[3];                          // 3 στήλες στη γραμμή 1  (jagged array)
```

Η Java υποστηρίζει **jagged arrays** (ανισόμηκες γραμμές) — κάθε γραμμή είναι ανεξάρτητος πίνακας.

Αναφορά στοιχείων πολυδιάστατου πίνακα με επανάληψη

```java
int[][] array; // δημιουργία με κάποιο τρόπο

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

// For-each (enhanced for):
for (int[] row : array) {
    for (int elem : row) {
        System.out.print(" " + elem);
    }
    System.out.println();
}
```

Βοηθητική κλάση `Arrays`

Παρέχει **static** μεθόδους — δεν χρειάζεται δημιουργία αντικειμένου.

Μέθοδος	Περιγραφή
`Arrays.sort(arr)`	Ταξινόμηση
`Arrays.parallelSort(arr)`	Παράλληλη ταξινόμηση *(Java 8+)*
`Arrays.fill(arr, val)`	Αρχικοποίηση
`Arrays.copyOfRange(arr, f, t)`	Αντιγραφή
`Arrays.equals(arr1, arr2)`	Ισότητα περιεχομένων
`Arrays.binarySearch(arr, key)`	Αναζήτηση (απαιτεί ταξινόμηση)

```java
double[] d = {8.4, 9.3, 0.2, 7.9, 3.4};
Arrays.sort(d);

int[] filled = new int[10];
Arrays.fill(filled, 7);

int[] copy = Arrays.copyOfRange(d, 0, d.length);
System.out.println(Arrays.equals(d, copy)); // true
```

Κλάση `Arrays` — Νεότερες προσθήκες

```java
int[] nums = {5, 3, 8, 1, 9, 2};

// Άθροισμα με Stream
int sum = Arrays.stream(nums).sum(); // 28

// Φιλτράρισμα και συλλογή
int[] evens = Arrays.stream(nums)
                    .filter(n -> n % 2 == 0)
                    .toArray(); // [8, 2]

// Εκτύπωση
Arrays.stream(nums).forEach(System.out::println);
```

NEW Java 8+

`Arrays.stream()`: μετατροπή πίνακα σε `Stream` για λειτουργική επεξεργασία.

```java
int[] a = {1, 2, 3, 4};
int[] b = {1, 2, 5, 4};
System.out.println(Arrays.compare(a, b));    // αρνητικό (a < b)
System.out.println(Arrays.mismatch(a, b));   // 2  (πρώτη θέση διαφοράς)
```

NEW Java 9+

`Arrays.compare()` & `Arrays.mismatch()`: λεξικογραφική σύγκριση και εύρεση πρώτης διαφοράς.

Συλλογές

Collections

Συλλογές (Collections) — Εισαγωγή

Προκαθορισμένες δομές δεδομένων που:

- Αποθηκεύουν ομάδες σχετικών αντικειμένων στη μνήμη.
- Παρέχουν αποτελεσματικές μεθόδους οργάνωσης, αποθήκευσης και ανάκτησης.
- Δεν απαιτούν γνώση της εσωτερικής υλοποίησης.

Βασικές συλλογές:

Διεπαφή	Περιγραφή
`List`	Ταξινομημένη λίστα, επιτρέπει διπλότυπα
`Set`	Σύνολο μοναδικών στοιχείων
`Queue` / `Deque`	Ουρά FIFO / διπλής κατεύθυνσης
`Map`	Αντιστοίχιση κλειδιών → τιμών

📖 https://docs.oracle.com/javase/tutorial/collections/interfaces/

Συλλογές ως Γενικές κλάσεις (Generics)

Κατά τη δήλωση συλλογής δηλώνεται και ο **τύπος** των δεδομένων:

```
ArrayList<String> list = new ArrayList<String>();
```

Diamond syntax — ο τύπος στο δεξί μέλος μπορεί να παραληφθεί *(Java 7+)*:

```
ArrayList<String> list = new ArrayList<>();
```

🆕 **Java 10+** — τοπική συμπέρανση τύπου με `var`:

```
var list = new ArrayList<String>(); // ο compiler συμπεραίνει ArrayList<String>
var map  = new HashMap<String, Integer>();
```

✓ Η `var` επιτρέπεται μόνο για **τοπικές μεταβλητές** (local variables), όχι για πεδία κλάσης ή παραμέτρους.

Αμετάβλητες Συλλογές — Factory Methods

🆕 **Java 9+** — `List.of()`, `Set.of()`, `Map.of()`: δημιουργία **αμετάβλητης** (immutable) συλλογής σε μία γραμμή.

```
// Αμετάβλητη λίστα
List<String> immutableList = List.of("A", "B", "Γ");

// Αμετάβλητο σύνολο
Set<Integer> immutableSet = Set.of(1, 2, 3, 4, 5);

// Αμετάβλητος χάρτης
Map<String, Integer> immutableMap = Map.of("one", 1, "two", 2, "three", 3);
```

✓ Κλήση `add()`, `remove()` κλπ. σε immutable συλλογή πετά `UnsupportedOperationException`.

🆕 **Java 10+** — `List.copyOf()`, `Set.copyOf()`, `Map.copyOf()`: αντιγραφή υπάρχουσας συλλογής σε αμετάβλητη.

```
List<String> mutable = new ArrayList<>(Arrays.asList("A", "B", "Γ"));
List<String> copy    = List.copyOf(mutable); // αμετάβλητο αντίγραφο
```

List

Λίστα στοιχείων **μεταβλητού** μεγέθους (κατά την εκτέλεση).

- Εσωτερική υλοποίηση με δυναμικό πίνακα (`ArrayList`) ή συνδεδεμένη λίστα (`LinkedList`).
- **Διατηρεί τη σειρά εισαγωγής**.
- Επιτρέπει **διπλότυπα**.

Μέθοδος	Περιγραφή
`get(i)` / `set(i, v)`	Ανάγνωση / Τροποποίηση κατά θέση
`add(v)` / `add(i, v)`	Προσθήκη στο τέλος / στη θέση i
`remove(i)` / `remove(obj)`	Αφαίρεση κατά θέση / τιμή
`indexOf(obj)`	Θέση πρώτης εμφάνισης
`contains(obj)`	Έλεγχος ύπαρξης
`size()`	Πλήθος στοιχείων
`clear()`	Άδειασμα

Παράδειγμα χρήσης `ArrayList`

```
ArrayList<String> list = new ArrayList<String>();
for (String a : args) list.add(a);

Collections.shuffle(list);
System.out.println("Shuffled: " + list);

Collections.sort(list);
System.out.println("Sorted: "   + list);

System.out.println("Contains '5'? " + list.contains("5"));
System.out.println("indexOf  '5'? " + list.indexOf("5"));

list.remove("5");
System.out.println("Contains '5'? " + list.contains("5"));

System.out.println("Before add size: " + list.size());
list.add("last");
System.out.println("After  add size: " + list.size());
System.out.println("List: "           + list);
```

`ArrayList` — Νεότερες δυνατότητες

🆕 **Java 8+** — `forEach()` με λάμδα (lambda expression):

```
List<String> list = new ArrayList<>(List.of("A", "B", "Γ"));

list.forEach(s -> System.out.println(s));   // lambda
list.forEach(System.out::println);           // method reference
```

🆕 **Java 8+** — `removeIf()`: αφαίρεση στοιχείων με βάση συνθήκη:

```
list.removeIf(s -> s.startsWith("A")); // αφαιρεί όσα αρχίζουν με "A"
```

🆕 **Java 8+** — `replaceAll()`: μετασχηματισμός κάθε στοιχείου επί τόπου:

```
List<String> names = new ArrayList<>(List.of("alice", "bob", "carol"));
names.replaceAll(String::toUpperCase); // [ALICE, BOB, CAROL]
```

`LinkedList` — Συνδεδεμένη Λίστα

`LinkedList<E>` υλοποιεί και `List` και `Deque`:

```
LinkedList<String> linked = new LinkedList<>();
linked.add("B");
linked.addFirst("A");  // προσθήκη στην αρχή
linked.addLast("Γ");   // προσθήκη στο τέλος
System.out.println(linked); // [A, B, Γ]

linked.removeFirst();
linked.removeLast();
System.out.println(linked); // [B]
```

	`ArrayList`	`LinkedList`
Τυχαία πρόσβαση `get(i)`	O(1) ✓	O(n) ✘
Εισαγωγή / Διαγραφή στη μέση	O(n) ✘	O(1) ✓
Κατανάλωση μνήμης	μικρότερη	μεγαλύτερη

√ Προτιμήστε `ArrayList` ως default — χρησιμοποιείστε `LinkedList` μόνο αν έχετε συχνές εισαγωγές/διαγραφές.

Sets

Μη ταξινομημένες (ως επί το πλείστον) συλλογές **μοναδικών** αντικειμένων.

- Δεν επιτρέπουν διπλότυπα.
- Δεν παρέχουν πρόσβαση μέσω δείκτη.

Μέθοδος	Περιγραφή
`add(v)`	Προσθήκη
`remove(obj)`	Αφαίρεση
`contains(obj)`	Έλεγχος ύπαρξης
`size()`	Πλήθος
`clear()`	Άδειασμα

Εναλλακτικές υλοποιήσεις:

Κλάση	Χαρακτηριστικό
`HashSet`	Βέλτιστη απόδοση O(1), μη ταξινομημένο
`TreeSet`	Ταξινομημένη διάταξη O(log n)
`LinkedHashSet`	Διατηρεί σειρά εισαγωγής

Παράδειγμα χρήσης Set

```
Set<String>    hashSet = new HashSet<String>();
TreeSet<String> treeSet = new TreeSet<String>();

for (String a : args) {
    hashSet.add(a);
    treeSet.add(a);
}

System.out.println("HashSet: " + hashSet); // τυχαία σειρά
System.out.println("TreeSet: " + treeSet); // αλφαβητική σειρά

System.out.println("HashSet Contains '5'? " + hashSet.contains("5"));
hashSet.remove("5");
System.out.println("HashSet Contains '5'? " + hashSet.contains("5"));
System.out.println("HashSet: " + hashSet);

System.out.println("First and last in TreeSet: "
        + treeSet.first() + " " + treeSet.last());
```

Stack, Queue, Priority Queue

Stack (Στοίβα)

- LIFO — εισαγωγή και εξαγωγή μόνο από την **κορυφή**.

Queue (Ουρά)

- FIFO — εισαγωγή στο **tail**, εξαγωγή από το **head**.

PriorityQueue

- Τα στοιχεία εξάγονται βάσει **προτεραιότητας** (φυσική διάταξη ή `Comparator`).

Παράδειγμα χρήσης Stack / Queue / PriorityQueue

```
Stack<String> s = new Stack<>();
s.push("A"); s.push("B"); s.push("C");
while (s.size() > 0)
    System.out.print(s.pop() + " "); // C B A

Queue<String> q = new LinkedList<>();
q.add("A"); q.add("B"); q.add("C");
while (q.size() > 0)
    System.out.print(q.remove() + " "); // A B C

PriorityQueue<String> pq = new PriorityQueue<>();
pq.add("C"); pq.add("A"); pq.add("B");
while (pq.size() > 0)
    System.out.print(pq.remove() + " "); // A B C
```

`Deque` και `ArrayDeque` — Προτεινόμενη εναλλακτική

🆕 **Προτείνεται από Java 6+** — Η κλάση `java.util.Stack` θεωρείται **legacy** (παρωχημένη).

Η διεπαφή `Deque` (Double-Ended Queue) και η υλοποίηση `ArrayDeque` είναι η σύγχρονη επιλογή για **Stack** και **Queue**.

```
// Ως Stack (LIFO):
Deque<String> stack = new ArrayDeque<>();
stack.push("A"); stack.push("B"); stack.push("C");
while (!stack.isEmpty())
    System.out.print(stack.pop() + " "); // C B A

// Ως Queue (FIFO):
Deque<String> queue = new ArrayDeque<>();
queue.offer("A"); queue.offer("B"); queue.offer("C");
while (!queue.isEmpty())
    System.out.print(queue.poll() + " "); // A B C
```

✓ `ArrayDeque` είναι ταχύτερη από `Stack` και `LinkedList` για τις περισσότερες χρήσεις.

Maps

Αντιστοιχούν **keys** σε **values**:

- Τα keys είναι **μοναδικά**.
- Τα values δεν χρειάζεται να είναι μοναδικά.

Μέθοδος	Περιγραφή
`get(key)` / `put(key, val)`	Ανάγνωση / Εγγραφή
`remove(key)`	Αφαίρεση
`containsKey(key)` / `containsValue(val)`	Έλεγχος ύπαρξης
`size()` / `clear()`	Μέγεθος / Άδειασμα
`keySet()` / `values()` / `entrySet()`	Πλήθη / Τιμές / Ζεύγη

Εναλλακτικές υλοποιήσεις:

Κλάση	Χαρακτηριστικό
`HashMap`	Βέλτιστη απόδοση O(1)
`TreeMap`	Ταξινομημένα κλειδιά O(log n)
`LinkedHashMap`	Διατηρεί σειρά εισαγωγής

Παράδειγμα χρήσης `Map`

Υπολογισμός συχνότητας εμφάνισης παραμέτρων:

```
Map<String, Integer> m = new HashMap<String, Integer>();

for (String a : args) {
    Integer freq = m.get(a);
    m.put(a, (freq == null) ? 1 : freq + 1);
}

System.out.println("Count parameters:    " + args.length);
System.out.println("Distinct parameters: " + m.size());
System.out.println(m);
```

`Map` — Νεότερες δυνατότητες

🆕 **Java 8+** — Νέες μέθοδοι για πιο εκφραστικό κώδικα:

```java
Map<String, Integer> freq = new HashMap<>();

// getOrDefault: επιστρέφει default αν το key δεν υπάρχει
for (String a : args)
    freq.put(a, freq.getOrDefault(a, 0) + 1);

// putIfAbsent: εισάγει μόνο αν το key δεν υπάρχει
freq.putIfAbsent("νέο", 0);

// computeIfAbsent: υπολογισμός τιμής μόνο αν δεν υπάρχει
Map<String, List<String>> groups = new HashMap<>();
groups.computeIfAbsent("A", k -> new ArrayList<>()).add("Alice");

// forEach: επανάληψη πάνω σε ζεύγη
freq.forEach((key, val) -> System.out.println(key + " -> " + val));
```

Πρωτογενείς τύποι σε συλλογές — Wrapper Classes

Οι συλλογές δέχονται μόνο **αντικείμενα**, όχι primitives.
Λύση: **κλάσεις περιτυλίγματος (wrapper classes)** με αυτόματη μετατροπή (autoboxing/unboxing).

primitive	wrapper class
`boolean`	`Boolean`
`char`	`Character`
`byte`	`Byte`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`

```
ArrayList<Integer> integers = new ArrayList<Integer>();
integers.add(42);          // autoboxing: int → Integer
int x = integers.get(0);   // auto-unboxing: Integer → int
```

Πρόσβαση σε στοιχεία μέσω Iterators

```
ArrayList<Integer> list = new ArrayList<>();
Random rand = new Random();
for (int i = 0; i < 10; i++)
    list.add(rand.nextInt(100));
System.out.println(list);

Iterator<Integer> iter = list.iterator();
while (iter.hasNext()) {
    int i = iter.next();
    System.out.print(i + " ");
    if (i % 2 != 0)
        iter.remove(); // ασφαλής αφαίρεση κατά τη διάσχιση
}
System.out.println("\n" + list);
```

- `next()` επιστρέφει το επόμενο στοιχείο και προχωράει μία θέση.
- `hasNext()` ελέγχει αν υπάρχει επόμενο (αποφυγή `NoSuchElementException`).
- `iter.remove()` αφαιρεί ασφαλώς — **μην αφαιρείτε απευθείας από τη λίστα** μέσα σε for-each (πετά `ConcurrentModificationException`).

SequencedCollection — Νέα Ιεραρχία

🆕 **Java 21** — Εισαγωγή της διεπαφής `SequencedCollection` (και `SequencedSet`, `SequencedMap`).

Χωρίς `SequencedMap`

Collection	Accessing the first element	Accessing the last element
List	`list.get(0)`	`list.get(list.size() - 1)`
Deque	`deque.getFirst()`	`deque.getLast()`
SortedSet	`sortedSet.first()`	`sortedSet.last()`
LinkedHashSet	`linkedHashSet.iterator().next()`	`// missing`

Με `SequencedMap` παρέχει ενιαία πρόσβαση στο **πρώτο** και **τελευταίο** στοιχείο, και **αντίστροφη** διάσχιση:

```
SequencedCollection<String> seq = new ArrayList<>(List.of("A", "B", "Γ"));

seq.getFirst();  // "A"
seq.getLast();   // "Γ"

seq.addFirst("0"); // ["0", "A", "B", "Γ"]
seq.addLast("Δ");  // ["0", "A", "B", "Γ", "Δ"]

seq.removeFirst();
seq.removeLast();

// Αντίστροφη προβολή (view):
SequencedCollection<String> reversed = seq.reversed();
```

Υλοποιείται από: `ArrayList`, `LinkedList`, `ArrayDeque`, `TreeSet`, `LinkedHashSet`, κλπ.

Συνοπτική Σύγκριση Συλλογών

	`ArrayList`	`LinkedList`	`HashSet`	`TreeSet`	`HashMap`	`ArrayDeque`
Δομή	Δυν. πίνακας	Συνδ. λίστα	Πίνακας κατακερμ.	Κόκκινο-μαύρο δέντρο	Πίνακας κατακερμ.	Δυν. πίνακας
Σειρά	Εισαγωγής	Εισαγωγής	Καμία	Ταξινομημένη	Καμία	LIFO/FIFO
Διπλότυπα	✓	✓	✘	✘	✘ keys	✓
`get(i)`	O(1)	O(n)	—	—	O(1)	—
`add` / `remove`	O(1)*	O(1)	O(1)	O(log n)	O(1)	O(1)

*amortised για `ArrayList`

Πηγές

- Cay Horstmann, *Η γλώσσα προγραμματισμού JAVA – Αναλυτική Προσέγγιση*, Broken Hill
- Joyce Farrell, *JAVA Εκμάθηση με πρακτικά παραδείγματα*, Εκδόσεις ΚΡΙΤΙΚΗ
- Paul Deitel & Harvey Deitel, *Java How to Program*, 10/e
- Evan Jones, Adam Marcus, Eugene Wu, *Introduction to Programming in Java*, MIT OCW
- https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html
- https://docs.oracle.com/javase/tutorial/collections/interfaces/index.html
- https://openjdk.org/jeps/431 *(JEP 431 – Sequenced Collections, Java 21)*
- https://openjdk.org/jeps/286 *(JEP 286 – Local-Variable Type Inference* `var`*, Java 10)*