Αντικειμενοστραφής Προγραμματισμός

Β' εξάμηνο – Τμήμα Πληροφορικής – Ιόνιο Πανεπιστήμιο

Ενότητα 2

Μεταβλητές · Πρωτογενείς τύποι · Μετατροπές · Τελεστές · Σχόλια · Έλεγχος ροής

Δημήτρης Ρίγγας

Μηχανικός Η/Υ & Πληροφορικής, MSc, PhD

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

riggas@ionio.gr

Βασικές Έννοιες

Βασικές έννοιες

Μεταβλητές, σταθερές, παραστάσεις, εντολές

- **Μεταβλητή (Variable)**: θέση μνήμης που μπορεί να αλλάζει τιμές στη ροή του προγράμματος.
- **Σταθερά (Constant)**: οντότητα της οποίας η τιμή δεν αλλάζει ποτέ – ορίζεται μία φορά.
- **Παράσταση (Expression)**: συνδυασμός σταθερών, μεταβλητών, τελεστών ή/και συναρτήσεων.
- **Εντολή (Statement)**: οδηγία που τελειώνει με ερωτηματικό `;`.

```
int a = 10;
int b = 5;
int c = (a + 5) * b;
```

Προσδιοριστές (Identifiers)

Ονόματα μεταβλητών, μεθόδων, κλάσεων, πακέτων και διεπαφών.

- Αποτελούνται από: γράμματα, αριθμούς, `_` ή `$`
- Είναι **case-sensitive** · οποιοδήποτε μήκος

Συμβάσεις για μεταβλητές:

- Ξεκινούν με πεζό γράμμα
- Πεζά για μεταβλητές, ΚΕΦΑΛΑΙΑ για σταθερές (`final`)
- π.χ. `currentSpeed` – `TOP_GEAR`

✓ Αποδεκτό	✗ Λάθος
`space1`	`1space`
`roomSpace`	`room space`
`doubleSize`	`double` *(δεσμευμένη λέξη)*
`price_m3`	`price/m3`

Τύποι δεδομένων

“ Όλες οι εκφράσεις, τιμές και αντικείμενα **επιβάλλεται** να έχουν τύπο δεδομένων (**strongly-typed** γλώσσα). ”

- **Πρωτογενείς τύποι δεδομένων** *(primitive types)*
- **Σύνθετοι τύποι δεδομένων** – με `class`, `interface`, `array`

Java	JavaScript
`int j = 5;`	`var j = 5; j = j + "ος παίχτης";`
strongly-typed	loosely-typed

Πρωτογενείς τύποι δεδομένων

Τύπος	Μέγεθος	Εύρος τιμών
`byte`	8 bit	-128 … 127
`short`	16 bit	-32,768 … 32,767
`int`	32 bit	-2^31 … 2^31-1
`long`	64 bit	-2^63 … 2^63-1
`float`	32 bit	IEEE 754 single
`double`	64 bit	IEEE 754 double
`char`	16 bit	Unicode (UTF-16)
`boolean`	1 bit	`true` / `false`

“ ⚠️ **Δεν υπάρχουν** `unsigned` τύποι! **Δεν υπάρχουν** δείκτες! ”

Σταθερές

Επώνυμες θέσεις μνήμης (**named / symbolic constants**), δηλώνονται με `final`.

- Μπορεί να λάβει τιμή **στο σημείο δήλωσης** ή **αργότερα, αλλά μόνο μία φορά**
- Σύμβαση: **ΚΕΦΑΛΑΙΑ** γράμματα

```
public class Constants {
    final static double VAT = 0.23; // ποσοστό ΦΠΑ

    public static void main(String[] args) {
        final double REDUCTION;
        REDUCTION = 0.15; // 15% έκπτωση

        double price = 60.0;
        double ppv   = price + price * VAT;
        double pay   = ppv - ppv * REDUCTION;
        System.out.println("Καθαρή αξία: " + price); // 60.0
        System.out.println("Αξία + ΦΠΑ: "  + ppv);   // 73.8
        System.out.println("Πληρωτέο:    "  + pay);   // 62.73
    }
}
```

Αριθμητικές τιμές & Literals

- **Ακέραια →** `int` by default; `l` / `L` suffix για `long`
- **Κινητής υποδιαστολής →** `double` by default; `f` / `F` suffix για `float`
- Υποστήριξη επιστημονικής αναπαράστασης (`1.234e2`)
- `_` **για αναγνωσιμότητα** *(Java SE 7+)*

```
boolean result   = true;
char    capitalC = 'C';
int     i        = 100_000;
long    l        = 317_000_000_000L;

float   f  = 123.4F;
double  d  = 123.4;
double  d2 = 1.234e2;          // = 123.4

int     hex  = 0xFF;           // = 255 (δεκαδικό)
int     hex2 = 0xFF_EC_D1_8C; // _ για αναγνωσιμότητα (Java 7+)
```

✓ Τα hex literals ξεκινούν με `0x` ή `0X`. Χρησιμοποιούνται συχνά σε χρώματα (RGB), bitmasks και πρωτόκολλα δικτύου.

Χαρακτήρες & String Literals

Ο τύπος `char` χρησιμοποιεί **UTF-16** encoding (16-bit).

Ακολουθία διαφυγής	Σημασία
`\b`	backspace
`\t`	tab
`\n`	line feed
`\r`	carriage return
`\"`	διπλά εισαγωγικά
`\'`	απλό εισαγωγικό
`\\`	backslash

```java
char capitalC  = 'C';
char sQuotes   = '\'';
// char lowerC = "c";  // compile error – απαιτούνται μονά εισαγωγικά
```

Αλφαριθμητικά (Strings)

Δεν είναι πρωτογενής τύπος – αντικείμενα της `java.lang.String`.

- Είναι **immutable**: από τη δημιουργία τους δεν τροποποιούνται.
- Χρήση `StringBuilder` / `StringBuffer` για mutable αλφαριθμητικά.

```
String str  = "abc";
String str2 = str + "def";  // συνένωση
```

```
// StringBuilder – γρήγορος, ΔΕΝ είναι thread-safe
StringBuilder sb = new StringBuilder("abc");
sb.append("def");
sb.append("ghi");
sb.insert(3, "-");   // "abc-defghi"
String result = sb.toString();  // → "abc-defghi"

// StringBuffer – πιο αργός, thread-safe
StringBuffer sbuf = new StringBuffer("abc");
sbuf.append("def");
sbuf.append("ghi");
String result2 = sbuf.toString();  // → "abcdefghi"
```

✓ Προτιμάτε `StringBuilder` στη συντριπτική πλειονότητα των περιπτώσεων.
✓ Χρησιμοποιείτε `StringBuffer` μόνο όταν πολλαπλά threads τροποποιούν το ίδιο αλφαριθμητικό ταυτόχρονα.

Text Blocks Java 13 preview → Java 15 stable

Πολυγραμμικά αλφαριθμητικά χωρίς escape sequences:

```
String json = """
        {
            "name": "Alice",
            "age": 30
        }
        """;
```

√ Ιδανικά για SQL, HTML, JSON, XML μέσα στον κώδικα.

Τελεστές

Αριθμητικοί τελεστές

Τελεστής	Πράξη
`+`	Πρόσθεση
`-`	Αφαίρεση
`*`	Πολλαπλασιασμός
`/`	Διαίρεση
`%`	Υπόλοιπο (modulo)

```
int result = 1 + 2;   // 3
result = result - 1;  // 2
result = result * 2;  // 4
result = result / 2;  // 2
result = result + 8;  // 10
result = result % 7;  // 3
```

Μοναδιαίοι τελεστές

Τελεστής	Σημασία
`+`	θετικό πρόσημο
`-`	αρνητικό πρόσημο
`++`	αύξηση κατά 1
`--`	μείωση κατά 1
`!`	λογική άρνηση

```
int result = +1;      // 1
result--;             // 0
result++;             // 1
result = -result;     // -1

boolean success = false;
success = !success;  // true
```

Τελεστές σύγκρισης

Τελεστής	Σημασία
`==`	ίσο με
`!=`	διάφορο από
`>`	μεγαλύτερο
`<`	μικρότερο
`>=`	μεγαλύτερο ή ίσο
`<=`	μικρότερο ή ίσο

```
int value1 = 1, value2 = 2;
if (value1 == value2) { ... }
if (value1 != value2) { ... }
if (value1 >  value2) { ... }
if (value1 <= value2) { ... }
```

√ Για **αντικείμενα** (String κ.λπ.) χρησιμοποιείτε `.equals()` για ισοδυναμία περιεχομένου!

Τελεστής σύγκρισης `? :`

Ο **μοναδικός τριαδικός τελεστής** της Java:

```
CONDITION ? STATEMENT_on_true : STATEMENT_on_false
```

Μπορεί να αποτελεί μέρος έκφρασης ανάθεσης:

```
Scanner sc = new Scanner(System.in);
int grade = sc.nextInt();

System.out.println(grade < 5
    ? "Μόνο!;.. \nθα πρέπει να ξαναδοκιμάστε."
    : "Πολύ καλά.");
```

Λογικοί τελεστές

Τελεστής	Σημασία	Short-circuit
`&&`	λογικό AND	✓
`||`	λογικό OR	✓
`&`	bitwise AND	✘
`|`	bitwise OR	✘
`!`	λογική άρνηση	—

Short-circuit αποτίμηση: ο δεύτερος τελεστέος αποτιμάται **μόνο αν χρειάζεται**:

- `A && B` → αν το `A` είναι `false`, το `B` **δεν ελέγχεται** (το αποτέλεσμα είναι ήδη `false`)
- `A || B` → αν το `A` είναι `true`, το `B` **δεν ελέγχεται** (το αποτέλεσμα είναι ήδη `true`)

```java
int value1 = 1, value2 = 2;
if ((value1 == 1) && (value2 == 2)) { ... }
if ((value1 == 1) || (value2 == 1)) { ... }

// Πρακτικό παράδειγμα: αποφυγή NullPointerException
String s = null;
if (s != null && s.length() > 0) { ... } // ✓ το s.length() δεν καλείται αν s == null

// Με & (χωρίς short-circuit) θα έσπαγε:
if (s != null & s.length() > 0) { ... }  // ✗ NullPointerException!
```

✓ Ο short-circuit δεν είναι μόνο βελτιστοποίηση — μπορεί να αποτρέψει σφάλματα εκτέλεσης.

Σύνθετοι τελεστές εκχώρησης

Τελεστής	Ισοδύναμο	Σημασία
`x += y`	`x = x + y`	πρόσθεση
`x -= y`	`x = x - y`	αφαίρεση
`x *= y`	`x = x * y`	πολλαπλασιασμός
`x /= y`	`x = x / y`	διαίρεση
`x %= y`	`x = x % y`	υπόλοιπο
`x &= y`	`x = x & y`	bitwise AND
`x |= y`	`x = x | y`	bitwise OR
`x <<= y`	`x = x << y`	αριστερή ολίσθηση
`x >>= y`	`x = x >> y`	δεξιά ολίσθηση

```java
int x = 10;
x += 5;   // x = 15
x *= 2;   // x = 30
x %= 7;   // x = 2
```

✓ **Implicit cast:** οι σύνθετοι τελεστές κάνουν αυτόματα cast στον τύπο της αριστερής μεταβλητής.
✓ `byte b = 10; b += 1;` ✓ — ισοδύναμο με `b = (byte)(b + 1)`, ενώ `b = b + 1;` ✗ (compile error!)

Προτεραιότητα τελεστών

Προτεραιότητα	Τελεστής	Κατηγορία
Υψηλότερη	`++` `--` (postfix)	μοναδιαίοι postfix
	`++` `--` `+` `-` `!` `~` (prefix)	μοναδιαίοι prefix
	`*` `/` `%`	πολλαπλασιαστικοί
	`+` `-`	προσθετικοί
	`<<` `>>` `>>>`	ολίσθηση bits
	`<` `>` `<=` `>=` `instanceof`	σχεσιακοί
	`==` `!=`	ισότητας
	`&`	bitwise AND
	`^`	bitwise XOR
	`|`	bitwise OR
	`&&`	λογικό AND
	`||`	λογικό OR
	`? :`	τριαδικός
Χαμηλότερη	`=` `+=` `-=` ...	εκχώρησης

```java
// Χωρίς γνώση προτεραιότητας → ασάφεια!
int result = 2 + 3 * 4;        // 14, όχι 20
boolean b  = 5 > 3 && 2 < 4;   // true  (σχεσιακοί πριν &&)
boolean c  = true || false && false; // true  (&& πριν ||)

// Χρησιμοποιείτε παρενθέσεις για σαφήνεια:
int result2 = (2 + 3) * 4;     // 20 — προφανές
```

✓ Μην βασίζεστε αποκλειστικά στην ιεραρχία
✓ Χρησιμοποιείτε **παρενθέσεις** για να κάνετε τον κώδικά σας σαφή

Μετατροπές (Casting)

Μετατροπές (Casting)

Ιεραρχία τύπων (υψηλότερα = μεγαλύτερη ακρίβεια):

```
double > float > long > int > short > byte
```

- **Widening (Προβιβασμός)** → αυτόματο, χωρίς απώλεια
- **Narrowing (Υποβιβασμός)** → ρητό cast, πιθανή απώλεια

```
// Widening (implicit)
int   i = 100;
long  l = i;      // αυτόματο
float f = l;      // αυτόματο

// Narrowing (explicit)
double d = 100.04;
long   l = (long)  d;  // 100 – χάνεται το δεκαδικό
int    i = (int)   l;  // 100
```

Προβιβασμοί σε αριθμητικές εκφράσεις

```
int    a = 4 / 2;     // 2     (ακέραια διαίρεση)
int    b = 5 / 2;     // 2     (ακέραια διαίρεση!!!)
double c = 4.0 / 2.0; // 2.5
double d = 5 / 2;     // 2.0   (int/int πρώτα → 2, μετά widening)
```

```
// Προσοχή στις παρενθέσεις:
double d1 = (double)(5 / 2); // → (double)2   = 2.0
double d2 = (double)5  / 2;  // → 5.0 / 2     = 2.5
```

✓ Για αναγνωσιμότητα κάνετε cast **πριν** την πράξη.

Σχόλια & Μπλοκ κώδικα

Σχόλια κώδικα

End-of-Line:

```
int x = 5; // αυτό είναι σχόλιο
```

Block:

```
/*
 * Εδώ περιγράφεται κάτι
 * σε πολλές γραμμές.
 */
```

Javadoc *(για αυτόματη τεκμηρίωση)*:

```
/**
 * Περιγραφή κλάσης/μεθόδου.
 * @version  1.0
 * @author   Firstname Lastname
 */
public class Example { ... }
```

🔗 Javadoc guide

Μπλοκ κώδικα

- Εντολές μέσα σε `{ }` · μπορούν να **εμφωλευτούν** (nested)
- Τοπικές μεταβλητές ζουν μόνο εντός του block τους
- **Δεν μπορεί** να επαναοριστεί μεταβλητή με ίδιο όνομα σε εσωτερικό block

```
int i = 100000;
int i2 = i + 1;
{
    int i3 = i2 + 1;
    System.out.println(i3); // 100002
}
// i3 δεν υπάρχει εδώ
System.out.println(i2); // 100001
System.out.println(i);  // 100000
```

- Π.χ. Η μεταβλητή ελέγχου ενός `for` **ζει μόνο μέσα στο block του**:
- αν χρειάζεστε την τιμή **μετά** τον βρόχο, δηλώστε τη μεταβλητή **έξω**:

```
for (int count = 0; count < 3; count++) {
    System.out.println("count = " + count); // ✓
}
System.out.println(count); // ✗ COMPILE ERROR
```

✓ Καλή πρακτική: δηλώνετε μεταβλητές στο **μικρότερο δυνατό scope** — ο κώδικας γίνεται πιο ασφαλής και ευανάγνωστος.

```
int count;                                  // δηλώνεται έξω
for (count = 0; count < 3; count++) {
    System.out.println("count = " + count);
}
System.out.println("Τελικό count: " + count); // ✓  → 3
```

Είσοδος / Έξοδος

Scanner – Αποδοχή εισόδου από πληκτρολόγιο

Η κλάση `Scanner` (`java.util.Scanner`) είναι ένας **αναλυτής κειμένου** που διαβάζει δεδομένα από μια πηγή εισόδου — συνήθως το πληκτρολόγιο (`System.in`), αλλά και από αρχεία ή Strings. Αναγνωρίζει αυτόματα τύπους δεδομένων (int, double κ.λπ.) και τα επιστρέφει έτοιμα για χρήση.

Μέθοδος	Επιστρέφει
`nextLine()`	Ολόκληρη γραμμή ως `String`
`next()`	Επόμενη λέξη ως `String`
`nextInt()`	`int`
`nextDouble()`	`double`
`nextFloat()`	`float`
`nextLong()`	`long`
`nextByte()`	`byte`

√ Δεν υπάρχει `nextChar()` !

√ **Καλή πρακτική**: κλείστε τον Scanner όταν τελειώσετε με `sc.close()`, ιδιαίτερα όταν διαβάζετε από αρχεία.

```java
import java.util.Scanner;  // απαραίτητο import

public class GetUserInfo {
    public static void main(String[] args) {
        // Δημιουργία Scanner συνδεδεμένου με το πληκτρολόγιο
        Scanner sc = new Scanner(System.in);

        System.out.print("Όνομα: ");
        String name = sc.nextLine();  // διαβάζει ολόκληρη γραμμή

        System.out.print("Ηλικία: ");
        int age = sc.nextInt();       // διαβάζει μόνο τον αριθμό

        System.out.println("Καλωσόρισες " + name + ", " + age + " ετών.");

        sc.close(); // αποδέσμευση πόρων
    }
}
```

Παγίδα Scanner: το Enter στο ρεύμα εισόδου

Οι μέθοδοι `nextInt()`, `nextDouble()` κ.λπ. διαβάζουν **μόνο την τιμή** — το `\n` (Enter) που πάτησε ο χρήστης **παραμένει στο ρεύμα** και θα το "δει" η επόμενη κλήση.

Χωρίς τη διόρθωση — λανθασμένη συμπεριφορά:

```
Χρήστης πληκτρολογεί:   2 5 [Enter]
                              ↑
Το nextInt() διαβάζει: 25
Στο ρεύμα απομένει:    \n  ← αυτό θα πάρει το επόμενο nextLine()!
```

```java
System.out.print("Ηλικία: ");
int age = sc.nextInt();       // διαβάζει 25, το \n μένει στο ρεύμα

System.out.print("Όνομα: ");
String name = sc.nextLine();  // παίρνει αμέσως το \n → name = ""  ✗

System.out.println("Καλωσόρισες " + name); // "Καλωσόρισες "  ← κενό!
```

Με τη διόρθωση — σωστή συμπεριφορά:

```java
System.out.print("Ηλικία: ");
int age = sc.nextInt();
sc.nextLine();                // ← "απορροφά" το \n που έμεινε  ✓

System.out.print("Όνομα: ");
String name = sc.nextLine();  // τώρα περιμένει κανονικά την είσοδο  ✓

System.out.println("Καλωσόρισες " + name); // "Καλωσόρισες Γιώργης"
```

✓ **Εναλλακτική λύση:** χρησιμοποιείτε αποκλειστικά `nextLine()` και μετατρέπετε τον τύπο χειροκίνητα:

```java
int age = Integer.parseInt(sc.nextLine()); // αποφεύγει εντελώς την παγίδα
```

Μορφοποιημένη έξοδος – printf

```
System.out.printf("Q:%10d%nP:%10.2f%n", 2, 17.29);
```

Προσδιοριστής	Παράδειγμα	Σημασία
`%d`	`24`	ακέραιος
`%5d`	`24`	ακέραιος, πλάτος 5
`%f`	`1.21997`	κινητής υποδιαστολής
`%.2f`	`1.22`	2 δεκαδικά
`%s`	`Hello`	αλφαριθμητικό
`%n`	*(νέα γραμμή)*	platform-independent newline

Προχωρημένοι μαθηματικοί υπολογισμοί – `Math`

```java
import java.lang.Math; // προαιρετικό – java.lang.* φορτώνεται αυτόματα

double a = 30, b = 2, c = 12.75;

System.out.println(Math.pow(a, b));   // 900.0
System.out.println(Math.sqrt(a));     // 5.477...
System.out.println(Math.floor(c));    // 12.0
System.out.println(Math.cos(Math.PI)); // -1.0
System.out.println(Math.abs(-42));    // 42
```

🔗 Java 21 Math API

Έλεγχος Ροής

Έλεγχος ροής – Επισκόπηση

Κατηγορία	Εντολές
Ακολουθία	εντολές με τη σειρά
Επιλογή	`if`, `if-else`, `switch`
Επανάληψη	`for`, `while`, `do-while`, `for-each`
Διακλάδωση	`break`, `continue`, `return`

Επιλογή

`if` – Απλή επιλογή

```
if (CONDITION) {
    STATEMENTS
}

// Για μία εντολή – οι αγκύλες προαιρετικές (αλλά προτιμήστε να τις βάζετε!)
if (CONDITION)
    STATEMENT;
```

```
Scanner sc = new Scanner(System.in);
int grade = sc.nextInt();

if (grade > 8)
    System.out.println("Εξαιρετικός!");

if (grade < 5) {
    System.out.println("Μόνο!;..");
    System.out.println("Θα πρέπει να ξαναδοκιμάστε.");
}
```

`if … else` – Σύνθετη επιλογή

```
if (CONDITION) {
    STATEMENTS
} else {
    STATEMENTS
}
```

```
if (grade < 5) {
    System.out.println("Θα πρέπει να ξαναδοκιμάστε.");
} else {
    System.out.println("Πολύ καλά.");
}
```

`if … else if` – Πολλαπλή επιλογή

```
if (grade < 5) {
    System.out.println("Αποτυχία.");
} else if (grade < 7) {
    System.out.println("Καλά.");
} else if (grade < 9) {
    System.out.println("Πολύ καλά.");
} else {
    System.out.println("Άριστα!");
}
```

✓ Μόλις ικανοποιηθεί μια συνθήκη, **οι υπόλοιπες δεν ελέγχονται**.

Ένθετες `if` & Αιωρούμενα (Dangling) `else`

```
// Ποιο if "ανήκει" το else;
if (grade >= 5)
    if (grade >= 9)
        System.out.println("Άριστα!");
else
    System.out.println("Αποτυχία.");
```

Ένθετες `if` & Αιωρούμενα (Dangling) `else`

```
// Ποιο if "ανήκει" το else;
if (grade >= 5)
    if (grade >= 9)
        System.out.println("Άριστα!");
else   // ← ανήκει στο ΕΣΩΤΕΡΙΚΟ if!
    System.out.println("Αποτυχία.");
```

√ Η Java (όπως C/C++) αντιστοιχεί το `else` στο **πλησιέστερο** `if`.
√ **Λύση:** χρησιμοποιείτε πάντα `{ }`.

Κενά μπλοκ & `=` αντί `==`

```
// Λάθος: το ; μετά το if δημιουργεί κενό block
if (grade > 8);
    System.out.println("Εξαιρετικός!"); // πάντα εκτελείται!
```

```
// Compile error: ανάθεση σε boolean context
if (grade = 10)  // ✗ – δεν επιτρέπεται
    System.out.println("Άριστος!");

// Σωστό:
if (grade == 10) // ✓
    System.out.println("Άριστος!");
```

`switch` – Κλασική μορφή

```
switch (expression) {       // byte, short, int, char, String, enum
    case VALUE1:
        STATEMENTS;
        break;
    case VALUE2:
    case VALUE3:            // fall-through: VALUE2 και VALUE3 μαζί
        STATEMENTS;
        break;
    default:
        STATEMENTS;
}
```

- Από **Java 7+**: η έκφραση μπορεί να είναι `String`.
- Από **Java 5+**: `enum` τύποι.

Switch Expressions Java 14+

Η νέα σύνταξη εξαλείφει το **fall-through** και το `break`:

```
String result = switch (grade) {
    case 9, 10 -> "Άριστα!";
    case 7, 8  -> "Πολύ καλά!";
    case 5, 6  -> "Καλά.";
    default    -> "Αποτυχία.";
};
System.out.println(result);
```

Με `yield` για πολύπλοκα blocks:

```
int bonus = switch (category) {
    case "A" -> 500;
    case "B" -> {
        int base = 200;
        yield base + 50;
    }
    default -> 0;
};
```

Επανάληψη

Δομές επανάληψης – Επισκόπηση

Δομή	Πότε χρησιμοποιείται
`for`	Γνωστό πλήθος επαναλήψεων
`while`	Άγνωστο πλήθος, έλεγχος **πριν**
`do-while`	Άγνωστο πλήθος, έλεγχος **μετά** (≥1 εκτέλεση)
`for-each`	Διάσχιση συλλογών / πινάκων

while

```
while (CONDITION) {
    STATEMENTS
}
```

```
int i = 0;
while (i < 11) {
    System.out.println(i);
    i++;
}
System.out.println("After loop i: " + i); // 11
```

✓ Βεβαιωθείτε ότι ο βρόχος **τερματίζει** (αποφύγετε ατέρμονα βρόχο).

do … while

Εκτελείται **τουλάχιστον μία φορά**.

```
do {
    STATEMENTS
} while (CONDITION);
```

```
int i = 1;
do {
    System.out.println(i);
    i++;
} while (i < 1);
// Εκτυπώνει: 1  (παρότι η συνθήκη ήταν false)
```

for

```
for (INITIALIZATION; CONDITION; INCREMENT) {
    STATEMENTS
}
```

```
int sum = 0;
Scanner sc = new Scanner(System.in);

for (int count = 0; count < 10; count++) {
    System.out.print("Βαθμός " + (count + 1) + ": ");
    sum += sc.nextInt();
}
System.out.println("M.O.: " + (sum / 10.0));
```

√ Οι εκφράσεις της επικεφαλίδας είναι προαιρετικές: `for (;;)` = ατέρμονας βρόχος.

`for-each` (Enhanced for) Java 5+

Απλοποιεί τη διάσχιση πινάκων και συλλογών:

```
int[] grades = {8, 6, 9, 5, 7};

// Κλασική for
for (int i = 0; i < grades.length; i++) {
    System.out.println(grades[i]);
}

// Enhanced for-each
for (int g : grades) {
    System.out.println(g);
}
```

`while` ελεγχόμενο από σήμα (Sentinel)

```java
int sum = 0, count = 0;
Scanner sc = new Scanner(System.in);

System.out.print("Βαθμός (ή -1 για τέλος): ");
int grade = sc.nextInt();

while (grade != -1) {       // -1 είναι ο "φρουρός"
    sum += grade;
    count++;
    System.out.print("Βαθμός (ή -1 για τέλος): ");
    grade = sc.nextInt();
}

System.out.println("M.O.: " + (count > 0 ? (double) sum / count : "N/A"));
```

Διακλάδωση

`break`

Εξόδος από `while`, `do-while`, `for` ή `switch`.

```
int[] arr = {32, 87, 3, 589, 12, 1076};
int searchFor = 12;
boolean found = false;
int idx = -1;

for (int i = 0; i < arr.length; i++) {
    if (arr[i] == searchFor) {
        found = true;
        idx = i;
        break;  // ← έξοδος από το for
    }
}

System.out.println(found ? "Βρέθηκε στη θέση " + idx : "Δεν βρέθηκε.");
```

`continue`

- Σε `while` / `do-while`: πηγαίνει κατευθείαν στον **έλεγχο συνθήκης**.
- Σε `for`: εκτελεί πρώτα την **έκφραση αύξησης**.

```
String text = "peter piper picked a peck of pickled peppers";
int countP = 0;

for (int i = 0; i < text.length(); i++) {
    if (text.charAt(i) != 'p')
        continue;   // παράλειψη – μας ενδιαφέρουν μόνο τα 'p'
    countP++;
}
System.out.println("Βρέθηκαν " + countP + " 'p'.");
```

Labeled `break` & `continue`

Για εμφωλευμένους βρόχους – έξοδος/συνέχιση **εξωτερικού** βρόχου:

```
outer:
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 5; j++) {
        if (j == 3) break outer;   // έξοδος και από τον εξωτερικό βρόχο
        System.out.println(i + "," + j);
    }
}
```

✓ Κατάχρηση οδηγεί σε **μη δομημένο** κώδικα – χρησιμοποιείτε με μέτρο.

`return`

Έξοδος από την τρέχουσα μέθοδο (με ή χωρίς τιμή επιστροφής).

```
public static int findInArray(int[] arr, int target) {
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == target)
            return i;     // βρέθηκε – επιστροφή δείκτη
    }
    return -1;            // δεν βρέθηκε
}
```

Σημαντικές Παρατηρήσεις

Αρχικοποίηση μεταβλητών

- **Μέλη κλάσης** (attributes): αρχικοποιούνται αυτόματα (0, `false`, `null`)
- **Τοπικές μεταβλητές**: **ΔΕΝ** έχουν αρχική τιμή – ευθύνη του προγραμματιστή

```
public class Defaults {
    static int member;  // = 0 αυτόματα

    public static void main(String[] args) {
        int local;  // χωρίς αρχική τιμή!
        System.out.println(member); // OK → 0
        System.out.println(local);  // COMPILE ERROR!
    }
}
```

✓ **Καλή πρακτική**: αρχικοποιείτε πάντα εσείς, μην εξαρτάστε από τον compiler.

Σύγκριση αντικειμένων

```
String a = new String("hello");
String b = new String("hello");

System.out.println(a == b);       // false – διαφορετικές αναφορές!
System.out.println(a.equals(b));  // true  – ίδιο περιεχόμενο ✓
```

Τελεστής	Χρήση
`==` / `!=`	Σύγκριση **αναφορών** (ή primitive τιμών)
`.equals()`	Σύγκριση **περιεχομένου** αντικειμένων

Σύγκριση `double` / `float` τιμών

Λόγω floating-point αναπαράστασης (IEEE 754), η ισότητα `==` μπορεί να αποτύχει!

```
double d1 = 1.0000000001 - 1.0;
double d2 = 0.0000000001;
System.out.println(d1 == d2); // false (!)
System.out.println(d1);       // 1.0000000002E-10
System.out.println(d2);       // 1.0E-10
```

Σωστός τρόπος:

```
double EPSILON = 1e-9;
if (Math.abs(d1 - d2) < EPSILON) {
    System.out.println("Θεωρούνται ίσοι");
}
```

🔗 Πραγματική ζωή: Patriot missile bug

Σύγκριση με μηδέν – Performance

Η σύγκριση με **0** είναι ταχύτερη από σύγκριση με αυθαίρετη τιμή:

```
// Λίγο πιο αργό
for (int x = 0; x < 100_000_000; x++) { ... }

// Λίγο πιο γρήγορο σε χαμηλό επίπεδο
for (int x = 100_000_000; x != 0; x--) { ... }
```

✓ Στη σύγχρονη JVM (JIT compiler) η διαφορά είναι ελάχιστη σε πρακτικές εφαρμογές – προτεραιότητα πάντα η **αναγνωσιμότητα**!

`var` – Local Variable Type Inference Java 10+

Ο compiler **συμπεραίνει** τον τύπο από την τιμή ανάθεσης:

```
var list    = new ArrayList<String>();  // ArrayList<String>
var message = "Hello, world!";          // String
var count   = 42;                       // int
var scanner = new Scanner(System.in);   // Scanner
```

⚠️ Ισχύει **μόνο για τοπικές μεταβλητές** – όχι για πεδία κλάσεων ή παραμέτρους.

- Ο τύπος παραμένει **στατικός** – δεν είναι dynamic typing!

Records – Αμετάβλητα data classes Java 16+

```java
// Παραδοσιακά:
public class Point {
    private final int x, y;
    public Point(int x, int y) { this.x = x; this.y = y; }
    public int x() { return x; }
    public int y() { return y; }
    // + equals(), hashCode(), toString()...
}

// Με record (ισοδύναμο!):
public record Point(int x, int y) {}

// Χρήση:
var p = new Point(3, 5);
System.out.println(p.x()); // 3
System.out.println(p);     // Point[x=3, y=5]
```

Εργαστήριο

Είσοδος παραμέτρων από τερματικό

```java
/*************************************************************************
 * ArgsReader.java
 *
 * Πρόγραμμα που διαβάζει μια παράμετρο από το τερματικό και την εμφανίζει.
 *
 * Compilation: javac ArgsReader.java
 * Execution:   java ArgsReader <text_without_spaces>
 *************************************************************************/
public class ArgsReader {
    public static void main(String[] args) {
        String s1 = args[0];
        System.out.println("Δώσατε παράμετρο: " + s1);
    }
}
```

Διαλογική είσοδος δεδομένων

```java
/*************************************************************************
 * IntScanner.java
 *
 * Πρόγραμμα που ζητά ακέραιο και τον εμφανίζει.
 *
 * Compilation: javac IntScanner.java
 * Execution:   java IntScanner
 *************************************************************************/
import java.util.Scanner;

public class IntScanner {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Πληκτρολογήστε ένα ακέραιο: ");
        int i1 = sc.nextInt();
        System.out.println("Πληκτρολογήσατε: " + i1);
    }
}
```

Πηγές

- Horstmann C., *Η γλώσσα προγραμματισμού JAVA – Αναλυτική Προσέγγιση*, Broken Hill Publishers
- Farrell J., *JAVA Εκμάθηση με πρακτικά παραδείγματα*, Εκδόσεις ΚΡΙΤΙΚΗ
- Deitel P. & H., *Java How to Program, 10/e*
- Θραμπουλίδης Κ., *Αντικειμενοστραφής Προγραμματισμός Java*, 3η εκδ.
- Λιόλιος Ν., *Αντικειμενοστραφής Προγραμματισμός Ι*
- Java Tutorials – Oracle
- Java 21 API Docs
- JLS – Java Language Specification