Αντικειμενοστραφής Προγραμματισμός

Β' εξάμηνο – Τμήμα Πληροφορικής – Ιόνιο Πανεπιστήμιο

Ενότητα 3

Κλάσεις & Αντικείμενα

Δημήτρης Ρίγγας

Μηχανικός Η/Υ & Πληροφορικής, MSc, PhD

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

riggas@ionio.gr

Γιατί χρησιμοποιούμε κλάσεις;

Αναπαράσταση εννοιών του πραγματικού κόσμου

- Ομαδοποίηση δεδομένων και μεθόδων που επενεργούν σε αυτά
- Προστασία, **ενθυλάκωση**, απόκρυψη, κληρονομικότητα

Κάθε αντικείμενο έχει:

- το δικό του σύνολο δεδομένων (**fields**)
- ένα σύνολο μεθόδων που επενεργούν σε αυτά τα δεδομένα

√ Έχουμε ήδη γνωρίσει το αντικείμενο `out` (της κλάσης `System`) και την κλάση `Scanner`

Πώς ορίζουμε μια κλάση

Ένας ορισμός κλάσης περιλαμβάνει:

Συστατικό	Περιγραφή
Όνομα	Ίδιο με το όνομα αρχείου (για `public` κλάσεις)
Πεδία (fields)	Περιγράφουν την κατάσταση του αντικειμένου
Κατασκευαστές	Αρχικοποίηση κατά τη δημιουργία
Μέθοδοι	Η συμπεριφορά — τι μπορεί να κάνει

Αρχείο: `ClassName.java`

```
public class ClassName {
    // field variables

    // constructors

    // methods
}
```

Ας ορίσουμε μια κλάση

Η κλάση `Baby` περιγράφει αντικείμενα με τα εξής πεδία:

- Όνομα, Βάρος, Φύλο, Πλήθος δοντιών

“ Το σύνολο των τιμών όλων των πεδίων ενός αντικειμένου καλείται **κατάσταση** του αντικειμένου. ”

```
public class Baby {

    String name;
    double weight = 4.0;
    boolean isABoy;
    int numTeeth = 0;

}
```

Έχοντας ορίσει αυτή την κλάση, μπορούμε να δημιουργήσουμε πολλά αντικείμενα τύπου `Baby`. Πώς;

Ας δημιουργήσουμε ένα στιγμιότυπο

- **Ορισμός κλάσης** = δημιουργία ενός νέου τύπου
- **Δημιουργία στιγμιότυπου** = δέσμευση μνήμης με τον τελεστή `new`

```
ΌνομαΚλάσης όνομαΑντικειμένου = new ΌνομαΚλάσης();
```

```
Baby george = new Baby();
```

Πώς ονομάζεται το μωρό; Ποια τιμή έχει το `name`;

Κατασκευαστές (constructors)

- Εκτελούνται **κατά τη δημιουργία** αντικειμένων
- Φροντίζουν για τη **σωστή αρχική διαμόρφωση** (αρχικοποίηση)
- Έχουν **ίδιο όνομα με την κλάση** και συνήθως δηλώνονται `public`

```
public class Baby {

    String name;
    double weight = 4.0;
    boolean isABoy;
    int numTeeth = 0;

    public Baby() { }  // Default constructor

}
```

Εάν εκτελέσουμε `Baby george = new Baby();` — ποιες default τιμές θα έχουν τα πεδία;

✓ Θυμηθείτε: `int` → `0`, `boolean` → `false`, `String` → `null`

Περισσότεροι κατασκευαστές

Αν θέλουμε **διαφορετική αρχικοποίηση**, δημιουργούμε επιπλέον κατασκευαστές.

`this`: αναφέρεται στο **τρέχον αντικείμενο** — χρησιμεύει για να διαχωρίσουμε τα πεδία από τις παραμέτρους με ίδιο όνομα.

```
public class Baby {

    String name;
    double weight = 4.0;
    boolean isABoy;
    int numTeeth = 0;

    public Baby() { }  // Default constructor

    public Baby(String name, boolean boy) {
        this.name = name;
        this.isABoy = boy;
    }
}
```

Εάν εκτελέσουμε `Baby george = new Baby("George", true);` — τι τιμές θα έχουν τα πεδία;

Πώς προσπελαύνουμε τα δεδομένα μιας κλάσης;

Για να αποκτήσουμε πρόσβαση στα μέλη ενός αντικειμένου, πρέπει πρώτα να το **δημιουργήσουμε** με `new`.

Η προεπιλεγμένη πρόσβαση είναι (περίπου σαν) `public` — οποιοσδήποτε μπορεί να αλλάξει τα πεδία:

```
Baby george = new Baby("George", true);
System.out.println("New baby is " + george.name); // George
george.name = "Nancy"; // Θέλουμε να μπορεί να γίνει αυτό;
System.out.println("New baby is " + george.name); // Nancy
```

Μπορούμε να **περιορίσουμε** αυτή την πρόσβαση:

- Δηλώνοντας `private` τα πεδία
- Παρέχοντας ελεγχόμενη πρόσβαση μέσω **μεθόδων**

Προσπέλαση πεδίων μέσω μεθόδων

Προσδιοριστές πρόσβασης:

Προσδιοριστής	Πρόσβαση
`public`	Από κάθε κλάση
`private`	Μόνο μέσα από την ίδια κλάση
`protected`	Από την ίδια κλάση και τις υποκλάσεις
(default) `package-private`	Από την ίδια κλάση και κλάσεις στο ίδιο package

```java
public class Baby {
    private String name;
    private double weight = 4.0;
    private boolean isABoy;
    private int numTeeth = 0;

    public String getName() {
        return this.name;
    }
    public void addTooth() {
        this.numTeeth++;
    }
}
```

Μέθοδοι πρόσβασης get/set

Getters — επιστρέφουν την τιμή πεδίου:

- Όνομα πεδίου: `name` → Όνομα μεθόδου: `getName()`
- Επιστρέφουν ίδιο τύπο με το πεδίο, δεν παίρνουν παραμέτρους

Setters — θέτουν την τιμή πεδίου:

- Όνομα πεδίου: `weight` → Όνομα μεθόδου: `setWeight(double weight)`
- Παίρνουν παράμετρο ίδιου τύπου, επιστρέφουν `void`

```java
public double getWeight() { return this.weight; }

public void setWeight(double weight) { this.weight = weight; }

// Χρήση:
Baby george = new Baby("George", true);
george.setWeight(3.420);
System.out.println("Baby: " + george.getName()
    + ", weight: " + george.getWeight());
// New baby is George, weight is 3.42
```

Προσέξτε στην `private` πρόσβαση

✓ `private` σημαίνει πρόσβαση από την **ίδια κλάση** — όχι το ίδιο αντικείμενο!

```
// Μέσα στην Baby:
public void removeOtherChildsTooth(Baby other) {
    other.numTeeth--;  // επιτρεπτό – ίδια κλάση
}
```

```
// Σε άλλη κλάση:
// george.numTeeth = 3;  error: numTeeth has private access in Baby
nancy.addTooth(); nancy.addTooth();
george.removeOtherChildsTooth(nancy);  //  μέσω public μεθόδου
```

Αντικείμενο της κλάσης `Baby` μπορεί να προσπελάσει `private` πεδία **άλλου** αντικειμένου της ίδιας κλάσης.

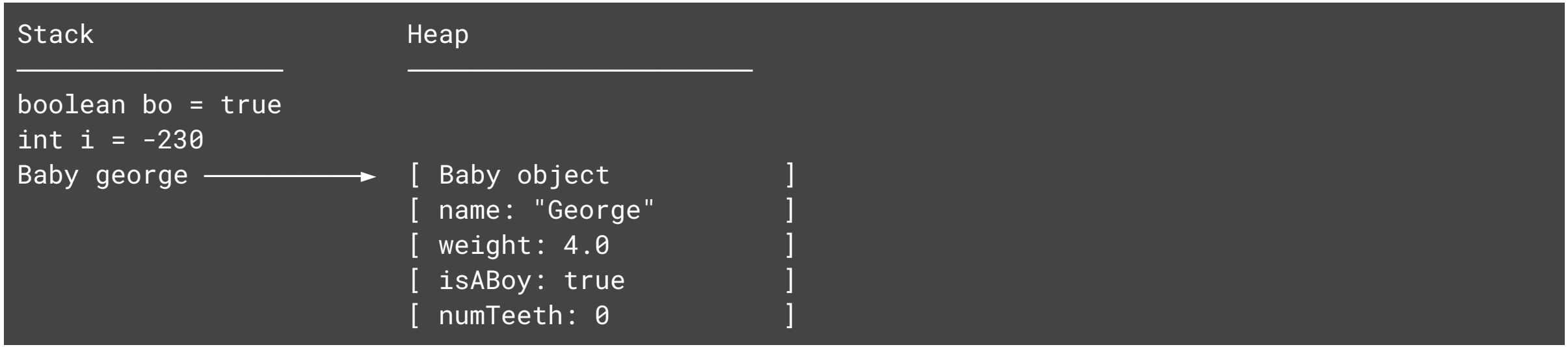
Διαχείριση μνήμης

Πρωτογενείς τύποι (`boolean`, `char`, `byte`, `short`, `int`, `long`, `float`, `double`):

- Αποθηκεύονται απ' ευθείας στη **στοίβα εκτέλεσης (stack)**

Τύποι αναφοράς (`String`, `Baby`, `int[]`, ...):

- Οι **μεταβλητές** αποθηκεύονται στο stack και περιέχουν **αναφορά (reference)**
- Τα **ίδια τα αντικείμενα** δημιουργούνται στο **σωρό (heap)**



Αναφορά — Σύγκριση με `==`

```java
Baby george1 = new Baby("George", true);
Baby george2 = new Baby("George", true);

System.out.println(george1 == george2);  // false – διαφορετικές θέσεις μνήμης
```

```
Stack            Heap
──────────       ──────────────────────────
george1 ──────►  [ Baby: name="George" ]
george2 ──────►  [ Baby: name="George" ]   ← διαφορετικό αντικείμενο!
```

Αναφορά — Ανάθεση αναφοράς

```
Baby george1 = new Baby("George", true);
Baby george2 = new Baby("George", true);
Baby george3 = george1;  // αντιγραφή αναφοράς!

System.out.println(george1 == george2);  // false
System.out.println(george1 == george3);  // true – ίδια θέση μνήμης
```

Ανάθεση

Τύπος	Τελεστής `=` κάνει...
Πρωτογενείς τύποι	Αντιγραφή τιμής
Τύποι αναφοράς	**Αντιγραφή διεύθυνσης** (alias — κοινό αντικείμενο)

```
Baby george1 = new Baby("George", true);
Baby george3 = george1;   // alias – ίδιο αντικείμενο

double w1 = 4.300;
double w2 = w1;     // αντιγραφή τιμής
w2 = 3.280;         // δεν επηρεάζει το w1

george1.setWeight(w1);
george3.setWeight(w2);  // george1 και george3 είναι ΤΟ ΙΔΙΟ αντικείμενο!

System.out.println(george1.getWeight()); // 3.28
System.out.println(george3.getWeight()); // 3.28
```

Σύγκριση

Τύπος	Τελεστής `==` κάνει...
Πρωτογενείς τύποι	Σύγκριση **τιμής**
Τύποι αναφοράς	Σύγκριση **διεύθυνσης** (όχι περιεχομένου!)

Για σύγκριση **περιεχομένου** χρησιμοποιούμε `equals()`:

```
Baby george1 = new Baby("George", true);

System.out.println(george1.getName() == new String("George"));
// false – διαφορετικές αναφορές

System.out.println(george1.getName().equals(new String("George")));
// true – ίδιο περιεχόμενο
```

√ `equals()` κληρονομείται από την κλάση `Object` και μπορεί να εξειδικευθεί στις δικές μας κλάσεις.

Σύγκριση — Ειδικά για `String` literals

Τα `String` literals υποβάλλονται σε **interning** από τον compiler:
ίδια literals → ίδια αναφορά στη μνήμη.

```
String hello = "Hello", lo = "lo";
final String cnst_lo = "lo";

System.out.println(hello == "Hello");          // true  (intern)
System.out.println(Other.hello == hello);      // true  (intern)
System.out.println(hello == ("Hel" + "lo"));   // true  (compile-time constant)
System.out.println(hello == ("Hel" + lo));     // false (runtime concatenation)
System.out.println(hello == ("Hel" + cnst_lo));// true  (final → compile-time constant)
```

“ ⚠️ Να χρησιμοποιείτε **πάντα** `equals()` για σύγκριση String περιεχομένου — ποτέ `==`.
Πηγή: JLS §3.10.5 ”

Σύγκριση πιθανά `null` τιμών

```
Baby george1 = new Baby("George", true);
Baby george3 = null;

george1.equals(george3);                  // false – ασφαλές
george3.equals(george1);                  // ❌ NullPointerException!
java.util.Objects.equals(george3, george1); // false – ασφαλές
java.util.Objects.equals(george1, george3); // false – ασφαλές
```

Κανόνας: όταν υπάρχει πιθανότητα `null`, χρησιμοποιήστε `java.util.Objects.equals()`.

Helpful NullPointerExceptions Java 14+

Από την Java 14, τα μηνύματα `NullPointerException` είναι **αναλυτικά** και δείχνουν ακριβώς ποια μεταβλητή ήταν `null`.

Πριν (Java < 14):

```
Exception in thread "main" java.lang.NullPointerException
    at EqualsNull.main(EqualsNull.java:19)
```

Μετά (Java 14+):

√ Από Java 21 ενεργοποιείται **by default**

```
Exception in thread "main" java.lang.NullPointerException:
  Cannot invoke "Baby.equals(Object)" because "george3" is null
    at EqualsNull.main(EqualsNull.java:19)
```

Εκτύπωση αντικειμένου – `toString()`

Για εκτύπωση αντικειμένου ορίζουμε `public String toString()`:

```
public class Baby {
    public String toString() {
        return "Baby is named " + this.getName()
             + " is " + (this.getIsABoy() ? "a boy" : "a girl")
             + " weights " + this.getWeight()
             + " and has " + this.getNumTeeth() + " teeth";
    }

    public static void main(String[] args) {
        Baby george1 = new Baby("George", true);
        george1.addTooth(); george1.addTooth(); george1.addTooth();

        System.out.println(george1);  // καλεί αυτόματα toString()
        // Baby is named George is a boy weights 4.0 and has 3 teeth

        Baby nancy = new Baby("Nancy", false);
        nancy.setWeight(3.200);
        nancy.addTooth(); nancy.addTooth();
        System.out.println(nancy);
    }
}
```

Αποδοτικότητα συνένωσης Strings

Κλάση	Χαρακτηριστικά
`String`	Αμετάβλητη (immutable), σταθερού μεγέθους
`StringBuffer`	Μεταβλητή, thread-safe
`StringBuilder`	Μεταβλητή, **χωρίς** thread-safety (γρηγορότερη)

Δοκιμή 100.000 επαναλήψεων `append`:

```
String:         ~4453 ms
StringBuffer:      ~9 ms
StringBuilder:     ~6 ms
```

✓ Ο compiler βελτιστοποιεί αυτόματα **απλές** συνενώσεις (`+`) με `StringBuilder`, αλλά σε βρόχους προτιμήστε ρητή χρήση `StringBuilder`.
✓ Πηγή: JLS §15.18.1

`toString()` με `StringBuilder`

```
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("Baby is named ").append(this.name)
      .append(" is ").append(this.isABoy ? "a boy" : "a girl")
      .append(" weights ").append(this.weight)
      .append(" and has ").append(this.numTeeth)
      .append(" teeth");
    return sb.toString();
}
```

“ Η `toString()` καλείται **αυτόματα** από τη JVM όταν περνάμε αντικείμενο σε `System.out.println()`. ”

`toString()` με `String.formatted()` Java 15+

Η μέθοδος `String.formatted()` (instance version του `String.format()`) κάνει τον κώδικα πιο ευανάγνωστο:

```
// Java 1.5+ – κλασικός τρόπος με String.format()
public String toString() {
    return String.format("Baby[name=%s, sex=%s, weight=%.3f, teeth=%d]",
                this.name,
                this.isABoy ? "boy" : "girl",
                this.weight,
                this.numTeeth);
}
```

```
// Java 15+ – ισοδύναμο με instance method .formatted()
public String toString() {
    return "Baby[name=%s, sex=%s, weight=%.3f, teeth=%d]"
            .formatted(
                this.name,
                this.isABoy ? "boy" : "girl",
                this.weight,
                this.numTeeth
            );
}
```

```
// Και οι δύο εκδοχές εκτυπώνουν:
// Baby[name=George, sex=boy, weight=4.000, teeth=3]
```

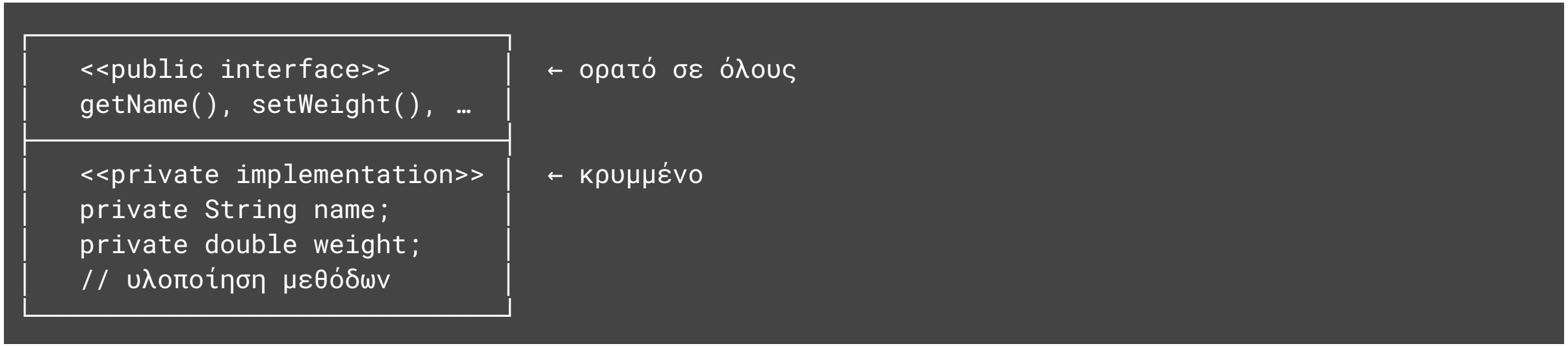
✓ Η λογική είναι ίδια — η `.formatted()` απλώς αποφεύγει την επανάληψη του format string ως πρώτο όρισμα.

Δημόσια διεπαφή μιας κλάσης (public interface)

Δημόσια διεπαφή = το σύνολο των `public` μεθόδων + η περιγραφή της συμπεριφοράς τους.

- Δεν χρειάζεται να γνωρίζει κάποιος **πώς υλοποιούνται** οι μέθοδοι
- Τα `private` μέλη και το σώμα μεθόδων = **ιδιωτική υλοποίηση**

✓ **Ενθυλάκωση** = απόκρυψη λεπτομερειών υλοποίησης
✓ → Μπορούν να γίνουν αλλαγές στην υλοποίηση χωρίς να επηρεάζονται οι χρήστες της κλάσης



Βασικές αρχές ΑΟΠ που είδαμε σε αυτή την ενότητα:

Αρχή	Τι σημαίνει	Πώς το είδαμε
Ενθυλάκωση	Δεδομένα και μέθοδοι μαζί σε μια κλάση	Κλάση `Baby` με fields + methods
Απόκρυψη δεδομένων	Εσωτερική υλοποίηση μη ορατή εξωτερικά	`private` fields, `public` interface
Αφαίρεση	Χρήση κλάσης χωρίς γνώση της υλοποίησης	getters/setters, `toString()`

Αποφυγή επαναλαμβανόμενου κώδικα

Project Lombok

Βιβλιοθήκη που συνδέεται με το σύστημα μεταγλώττισης και **ενθέτει αυτόματα κώδικα** με βάση annotations.

```
import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

@ToString(includeFieldNames = true)
public class Baby {
    @Getter              private String name;
    @Getter @Setter      private double weight = 4.0;
    @Getter              private boolean isABoy;
    @Getter              private int numTeeth = 0;

    public Baby(String name, boolean boy) {
        this.name = name;
        this.isABoy = boy;
    }
    public void addTooth() { this.numTeeth++; }
}
```

🔗 projectlombok.org

✓ Απαιτείται **κατά τη μεταγλώττιση**, όχι κατά την εκτέλεση.

Project Lombok — Βασικά Annotations

Annotation	Τι παράγει
`@Getter`	getter μέθοδο για κάθε πεδίο
`@Setter`	setter μέθοδο για κάθε πεδίο
`@ToString`	`toString()` με όλα τα πεδία
`@NoArgsConstructor`	Default constructor
`@AllArgsConstructor`	Constructor με παραμέτρους για όλα τα πεδία
`@EqualsAndHashCode`	`equals()` και `hashCode()`
`@Data`	Συνδυασμός: `@Getter` + `@Setter` + `@ToString` + `@EqualsAndHashCode`

Μεταγλώττιση & εκτέλεση:

```
javac -cp ./:./lombok.jar Baby.java
java Baby
```

✓ `-cp` (classpath): ορίζει τις θέσεις αναζήτησης κλάσεων κατά τη μεταγλώττιση. Εδώ περιλαμβάνει τον τρέχοντα φάκελο (`.`) και το `lombok.jar`, διαχωρισμένα με `:` (Linux/macOS) ή `;` (Windows).

Records — Σύγχρονη εναλλακτική Java 16+

Από Java ≥16, τα **Records** επιτρέπουν τη συνοπτική δήλωση κλάσεων δεδομένων (**immutable by default**), χωρίς boilerplate.

```
public record Baby(String name, double weight, boolean isABoy, int numTeeth) {

    // Compact constructor για validation
    public Baby {
        if (weight <= 0) throw new IllegalArgumentException("Αρνητικό βάρος!");
    }

    // Προσαρμοσμένη μέθοδος
    public void describe() {
        System.out.println(this);
    }
}
```

Αυτόματα παράγονται: `getter`, `equals()`, `hashCode()`, `toString()`.

```
Baby george = new Baby("George", 4.0, true, 0);
System.out.println(george.name());    // getter χωρίς "get"
System.out.println(george);           // Baby[name=George, weight=4.0, ...]
```

Records — Σύγκριση με κλασική κλάση

Χαρακτηριστικό	Κλασική κλάση	Lombok	Record
Boilerplate	Πολύ	Ελάχιστο	Καθόλου
Mutability	Mutable (default)	Mutable (default)	**Immutable**
`equals` / `hashCode`	Χειροκίνητα	`@EqualsAndHashCode`	Αυτόματα
`toString()`	Χειροκίνητα	`@ToString`	Αυτόματα
Κατάλληλο για	Γενική χρήση	Γενική χρήση	**DTOs, value objects**
Extra dependency	Όχι	Ναι (lombok.jar)	Όχι

✓ Τα Records είναι **immutable** — δεν υπάρχουν setters. Ιδανικά για χρήση ως data transfer objects, configuration κ.ά.

Τοπική Συναγωγή Τύπου — `var` Java 10+

Από Java ≥10, ο compiler μπορεί να **συναγάγει τον τύπο** τοπικών μεταβλητών αυτόματα:

```
// Παραδοσιακός τρόπος
Baby george = new Baby("George", true);
ArrayList<String> names = new ArrayList<String>();

// Με var — ο compiler γνωρίζει τον τύπο
var george = new Baby("George", true);   // τύπος: Baby
var names  = new ArrayList<String>();    // τύπος: ArrayList<String>
var weight = 4.3;                        // τύπος: double
```

✓ Το `var` δεν είναι δυναμικός τύπος — η Java παραμένει **static typed γλώσσα**.
✓ Το `var` επιτρέπεται **μόνο** για τοπικές μεταβλητές με αρχικοποίηση — όχι για πεδία κλάσης ή παραμέτρους.

Pattern Matching για `instanceof` Java 16+

Κλασικός τρόπος ελέγχου τύπου και casting:

```
// Παλιός τρόπος – επαναλαμβανόμενο cast
if (obj instanceof Baby) {
    Baby b = (Baby) obj;
    System.out.println(b.getName());
}
```

Με pattern matching (Java 16+):

```
// Σύγχρονος τρόπος – ο έλεγχος και το cast γίνονται μαζί
if (obj instanceof Baby b) {
    System.out.println(b.getName());  // b είναι ήδη τύπου Baby
}
```

✓ Χρήσιμο κυρίως κατά την υλοποίηση της `equals()` σε κλάσεις μας (βλ. επόμενη ενότητα).

Υλοποίηση `equals()` — Σύγχρονος τρόπος

Συνδυασμός pattern matching + `Objects.equals()` για ασφαλή σύγκριση:

```java
@Override
public boolean equals(Object obj) {
    if (this == obj) return true;
    // Java 16+ pattern matching
    if (!(obj instanceof Baby other)) return false;
    return Objects.equals(this.name, other.name)
        && this.isABoy == other.isABoy
        && Double.compare(this.weight, other.weight) == 0;
}

@Override
public int hashCode() {
    return Objects.hash(name, isABoy, weight);
}
```

✓ Τα **Records** παράγουν αυτόματα `equals()` και `hashCode()` χωρίς χειροκίνητο κώδικα! Αλλά προσοχή..

Records — `equals()` και σύγκριση τιμών Java 16+

Η αυτόματη `equals()` των Records συγκρίνει **τιμές όλων των components** — σε αντίθεση με τις κλάσεις:

	Default `equals()`	Τι συγκρίνει
Κλάση (extends Object)	Κληρονομείται από `Object`	Αναφορά (`==`)
Record	Αυτόματα παραγόμενη	Τιμές **όλων** των components

```java
// Κλάση – default equals() συγκρίνει αναφορές
Baby b1 = new Baby("George", 4.0, true, 0);
Baby b2 = new Baby("George", 4.0, true, 0);
b1.equals(b2); // false – εκτός αν ορίσουμε equals() χειροκίνητα

// Record – αυτόματη equals() συγκρίνει τιμές
record BabyRecord(String name, double weight, boolean isABoy, int numTeeth) {}
BabyRecord r1 = new BabyRecord("George", 4.0, true, 0);
BabyRecord r2 = new BabyRecord("George", 4.0, true, 0);
r1.equals(r2); // true – όλα τα components είναι ίδια
```

✓ Γι' αυτό τα Records είναι ιδανικά για **value objects** — αντικείμενα όπου η ταυτότητα ορίζεται από τα δεδομένα τους, όχι από τη θέση τους στη μνήμη.

Σύνοψη — Εξέλιξη στη δήλωση κλάσεων

```java
// Java 1.0+ – Κλασική κλάση (πολύ boilerplate)
public class Baby {
    private String name;
    private double weight;
    // + constructor, getters, setters, toString, equals, hashCode
}

// Java 10+ – var για συνοπτική δήλωση μεταβλητής (όχι κλάσης)
var george = new Baby("George", 4.0, true, 0);  // τύπος συνάγεται: Baby

// Project Lombok – Annotations αντί κώδικα
@Data public class Baby { ... }

// Java 16+ – Record (immutable, zero boilerplate)
public record Baby(String name, double weight, boolean isABoy, int numTeeth) {}
```

Πηγές

- Cay Horstmann, *Η γλώσσα προγραμματισμού JAVA — Αναλυτική Προσέγγιση*, Broken Hill
- Joyce Farrell, *JAVA — Εκμάθηση με πρακτικά παραδείγματα*, Κριτική
- Paul Deitel & Harvey Deitel, *Java How to Program, 10/e*
- Κλεάνθης Θραμπουλίδης, *Αντικειμενοστραφής Προγραμματισμός Java*, 3η εκδ.
- E. Jones, A. Marcus, E. Wu, *Introduction to Programming in Java* — MIT OCW
- JLS §3.10.5 — String Literals & Interning
- JEP 395 — Records (Java 16)
- JEP 286 — Local-Variable Type Inference / `var` (Java 10)
- JEP 394 — Pattern Matching for `instanceof` (Java 16)
- Project Lombok
- Primitives vs. References