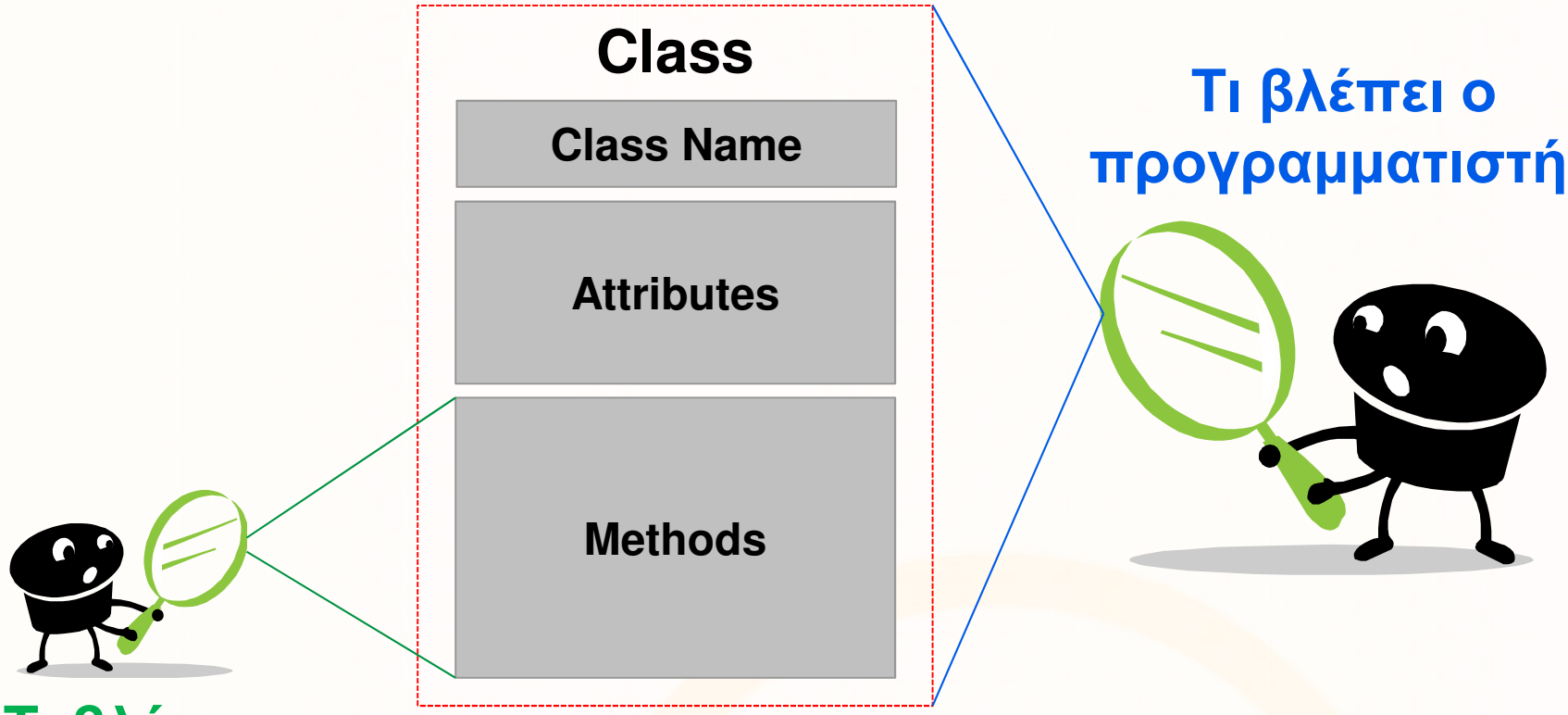
Διάλεξη 7: Ενθυλάκωση (encapsulation), Τροποποιητές (modifiers)

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

- Ενθυλάκωση
- Τροποποιητές Πρόσβασης (Access Modifiers), public, protected, private, friendly
- Στατικότητα
- final, άλλοι τροποποιητές

Διδάσκων: Παναγιώτης Ανδρέου

Ενθυλάκωση (encapsulation)



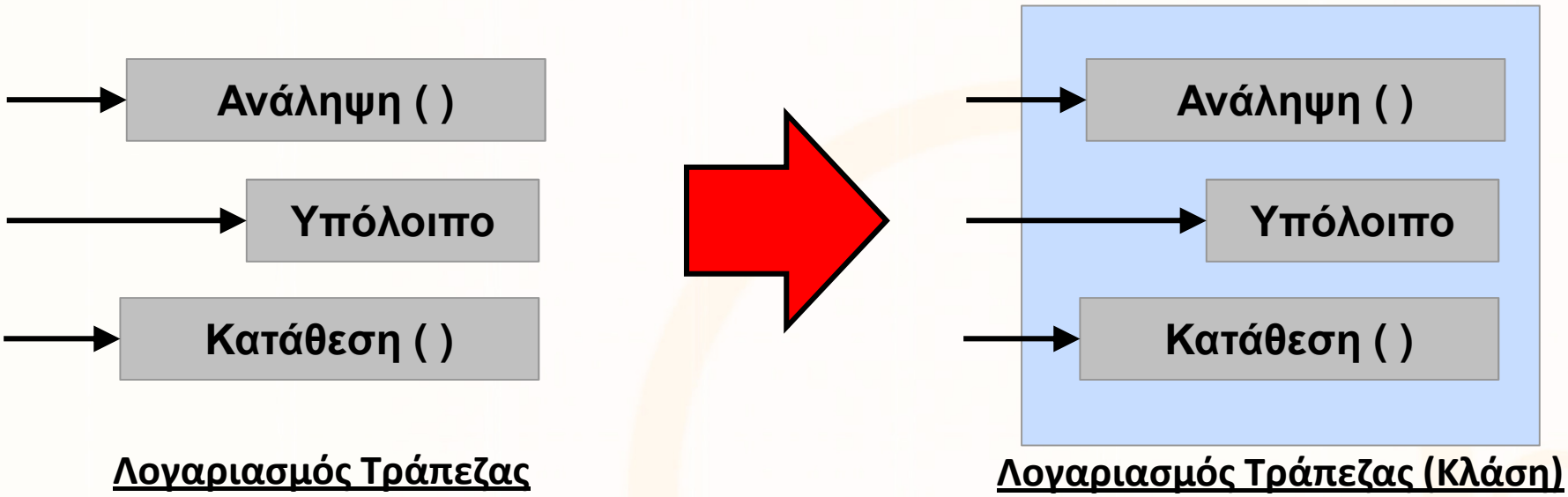
Ενθυλάκωση (encapsulation) (συν.)

Η Ενθυλάκωση (encapsulation) έχει δύο έννοιες:

1. **Συνδυασμός δεδομένων και μεθόδων σε ένα θύλακα (capsule)**

   Η κλάση σαν ένας τύπος δεδομένων περιλαμβάνει:

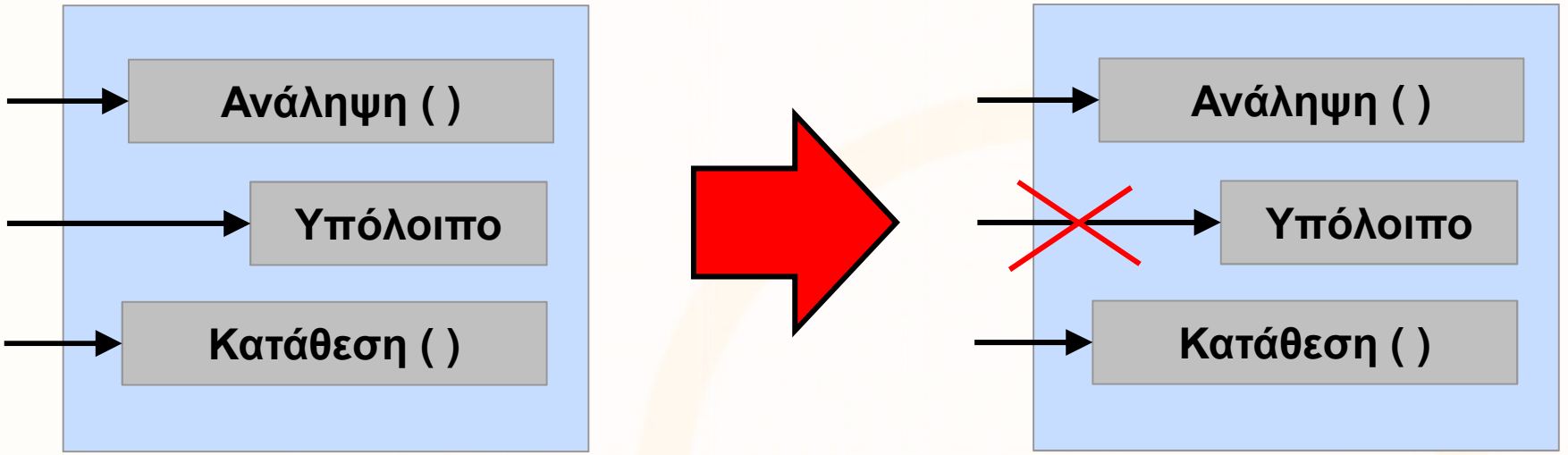
   - Μεταβλητές ή πεδία τα οποία αντιπροσωπεύουν διάφορες ιδιότητες
   - Μεθόδους οι οποίες χρησιμοποιούνται για να κατασκευάσουν στιγμιότυπα της κλάσης (constructors) και Μεθόδους οι οποίες χρησιμοποιούνται για να αναπαραστήσουν τις λειτουργίες της κλάσης (methods)



Ενθυλάκωση (encapsulation) (συν.)

2. Έλεγχος στην πρόσβαση δεδομένων (information hiding)

- Η κλάση εμφανίζει στους χρήστες και άλλες κλάσεις μία διαπροσωπεία (interface) η οποία παρουσιάζει τις λειτουργίες της κλάσης
- Με αυτό τον τρόπο, κρύβονται πληροφορίες χαμηλού επιπέδου για τον τρόπο υλοποίησης των λειτουργιών
- Παράδειγμα, για τον αφηρημένο τύπο δεδομένων Stack, παρουσιάζουμε Push(), Pop(), Top(), IsEmpty(), MakeEmpty() αλλά όχι την δομή δεδομένων

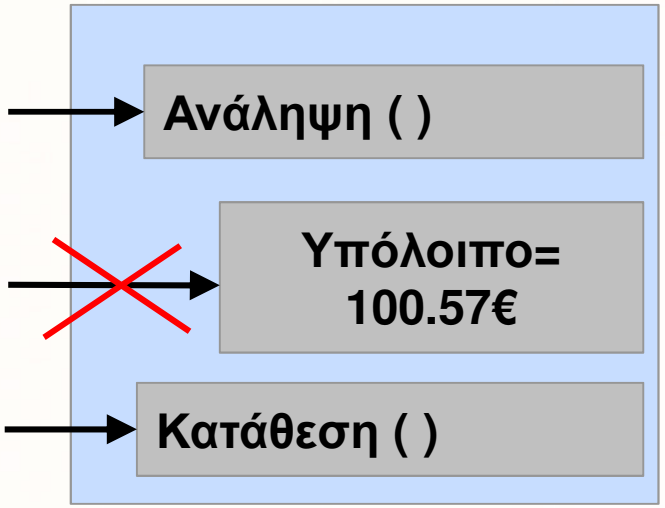


Ενθυλάκωση (encapsulation) (συν.)

Γιατί να χρησιμοποιήσουμε Ενθυλάκωση (encapsulation):

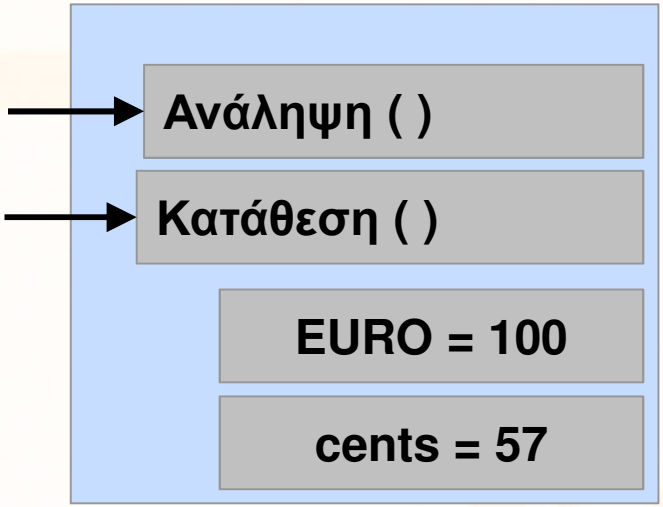
1. Επιτρέπει Έλεγχο

- Η χρήση του αντικειμένου γίνεται μόνο μέσα από της μεθόδους του
- Δεν επιτρέπεται η άμεση αλλαγή των δεδομένων του αντικειμένου



2. Επιτρέπει αλλαγές

- Η χρήση του αντικειμένου δεν αλλάζει αν αλλάξουν τα δεδομένα (που δεν έχουν πρόσβαση, δηλ. private)
- Πρέπει όμως να παραμείνουν τα ίδια πρότυπα μεθόδων



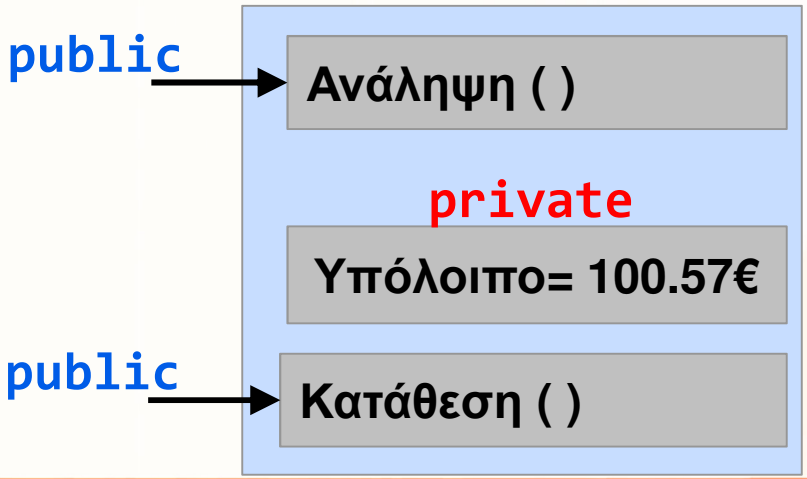
Ενθυλάκωση (encapsulation) (συν.)

Ερώτηση: Πως επιτυγχάνεται η Ενθυλάκωση;

Απάντηση: Με τους τροποποιητές πρόσβασης

Τυπικοί κανόνες:

- Τα μέλη μίας κλάσης όπως οι υψηλού επιπέδου μέθοδοι ορίζονται με **public** για να εκθέσουν τις λειτουργίες τους σε χρήστες/κλάσεις.
- Τα μέλη μίας κλάσης που παρέχουν λεπτομέρειες υλοποίησης ορίζονται με **private** για να κρύψουν πληροφορίες από τους χρήστες/κλάσεις.




```
public class BankAccount {
  private double balance;

  public void withdraw(
              double amount) { … }

  public void deposit(
              double amount) { … }
}
```

Τροποποιητές Πρόσβασης (Access Modifiers)

Μία κλάση ελέγχει την πρόσβαση στα διάφορα μέλη της (δεδομένα, μεθόδους) με τους **τροποποιητές πρόσβασης**

Υπάρχουν 4 τροποποιητές πρόσβασης:

- **public**: Η πρόσβαση στην κλάση, δεδομένα, μεθόδους είναι **ανοικτή σε όλους!**
- **private**: Η πρόσβαση στα δεδομένα και τις μεθόδους είναι **περιορισμένη μόνο στην κλάση**
- **default (friendly)**: Η πρόσβαση στην κλάση, δεδομένα και μεθόδους είναι **περιορισμένη στο σε κλάσεις που βρίσκονται στο ίδιο πακέτο με την κλάση**
- **protected**: **Όπως το default** αλλά **και σε κλάσεις που κληρονομούν από την ίδια κλάση** (εντός πακέτου ή όχι)

Τροποποιητές Πρόσβασης (Access Modifiers) (συν.)

- Σε περίπτωση που δεν χρησιμοποιείται προσδιοριστής πρόσβασης, έχουμε τον προκαθορισμένο **(default)** προσδιοριστής πρόσβασης, ο οποίος αναφέρεται σαν «φιλικός» (friendly):
- Ο φιλικός προσδιοριστής υποδηλώνει ότι όλες οι κλάσεις σε ένα πακέτο (βιβλιοθήκη) έχουν πρόσβαση στα «φιλικά» μέλη, τα οποία εμφανίζονται ως «ιδιωτικά» σε όλες τις κλάσεις έξω από την βιβλιοθήκη.
- Αφού μια μονάδα μετάφρασης (αρχείο) μπορεί να ανήκει σε μια μόνο βιβλιοθήκη, όλες οι κλάσεις μέσα σε αυτή θεωρούνται αυτομάτως φιλικές η μια προς την άλλη.

Τροποποιητές Πρόσβασης (Access Modifiers) (συν.)

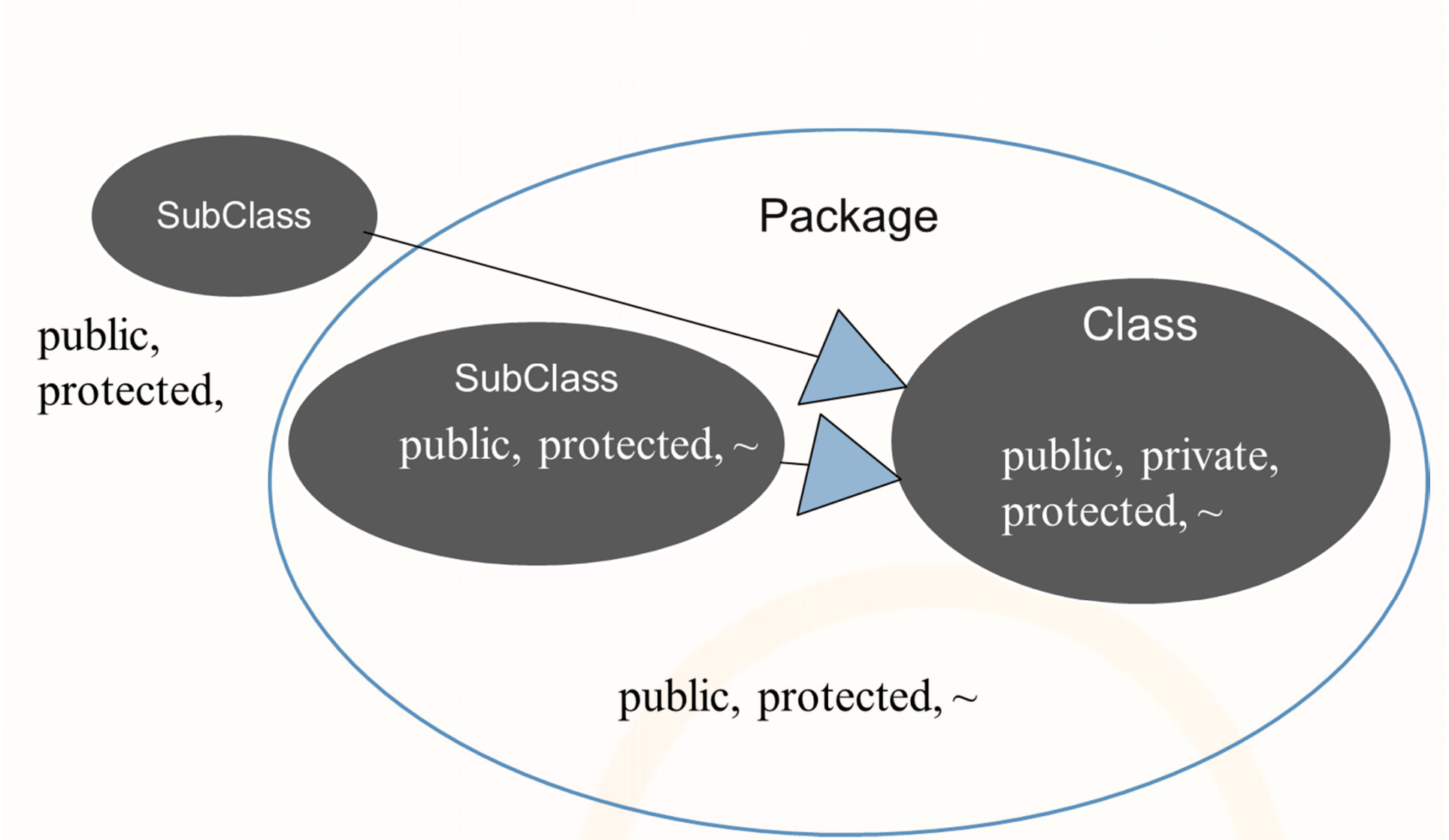
Για να δοθεί πρόσβαση προς τα μέλη μιας κλάσης, από άλλες κλάσεις, υπάρχουν οι εξής εναλλακτικές δυνατότητες:

- Να καταστεί το μέλος της κλάσης που μας ενδιαφέρει **δημόσιο (public)**. Όλοι μπορούν να έχουν.
- Να γίνει το μέλος της κλάσης που μας ενδιαφέρει **φιλικό**, οριζόμενο χωρίς κανέναν προσδιοριστή – και οι υπόλοιπες κλάσεις να τοποθετηθούν στην ίδια βιβλιοθήκη.
- Στην περίπτωση κληρονομικότητας, με χρήση του προσδιοριστή **protected.**
- Με δήλωση **μεθόδων πρόσβασης/τροποποίησης (accessor/mutator)**, οι οποίες διαβάζουν και γράφουν κάποιες τιμές-μέλη της κλάσης που μας αφορά.

Τροποποιητές Πρόσβασης: Σύγκριση

Τροποποιητής Πρόσβασης	Ίδια Κλάση	Ίδιο Πακέτο	Υποκλάση	Άλλα Πακέτα
public	√	√	√	√
protected	√	√	√	
(default) friendly	√	√		
private	√			

Τροποποιητές Πρόσβασης: Σύγκριση (συν.)



Τροποποιητές Πρόσβασης: private

```
public class BankAccount {
  private double balance;
  BankAccount(){ balance = 0.0; }
  BankAccount(double balance){ this.balance = balance; }
  public void withdraw( double amount ) { balance-=amount; }
  public void deposit(double amount) { balance+=amount;  }
  public double getBalance() { return balance; }
}
```

```
public class TestBankAccount {
  public static void main(String args[]) {
    BankAccount ba1 = new BankAccount();
    ba1.deposit(100.0);
    ba1.withdraw(40.0);

    System.out.println( ba1.getBalance() );
    //compile error BankAccount.balance is private
    System.out.println( ba1.balance );
  }
}
```

Τροποποιητές Πρόσβασης: default

<root>/bank

```
package bank;
public class BankAccount {
  private double balance;
  BankAccount(){ balance = 0.0; } //no modifier →default
  public BankAccount(double balance){ this.balance = balance; }
  …
}
```

<root>

```
import bank.BankAccount;
public class TestBankAccount {
  public static void main(String args[]) {

    BankAccount ba1 = new BankAccount();
    //compile error constructor BankAccount() does not exist

    BankAccount ba2 = new BankAccount(100.0);
    //OK. BankAccount(double) exists
  }
}
```

Τροποποιητές Πρόσβασης: protected

Θα αναλυθεί περισσότερο στην διάλεξη κληρονομικότητα

```
package bank;
public class BankAccount {
   private double balance;
   protected BankAccount(){
      balance = 0.0; }
  …
}
```

<root>/bank

```
package bank;
public class SavingsAccount
extends BankAccount{
   public SavingsAccount(){
      balance = 0.0; }
```

<root>

```
import bank.*;
public class TestBankAccount {
   public static void main(String args[]) {
      BankAccount ba1 = new BankAccount();
      //compile error constructor BankAccount() does not exist
      SavingsAccount sa1 = new SavingsAccount();
      //OK. SavingsAccount() exists
   }
}
```

Τροποποιητές Πρόσβασης: Καλύτερες Πρακτικές

- Όταν κατασκευάζετε μια κλάση είναι σκόπιμο να δηλώσετε τα πεδία δεδομένων της σαν ιδιωτικά, για να τα προστατεύσετε από ανεπιθύμητες αλλαγές από τρίτους.
- Αυτό ισχύει ακόμα κι αν η κλάση σας είναι προσβάσιμη από τη βιβλιοθήκη της μόνο (όχι δημόσια).
- Επίσης, είναι συνήθως λογικό να δώσετε στις μεθόδους της κλάσης την ίδια προσβασιμότητα με την κλάση, δηλ. δημόσια ή φιλική.
- Αν θέλετε να αποκλείσετε τη χρήση μιας κλάσης από τρίτους, δηλώστε όλους τους κατασκευαστές της **private**.
- Έτσι, κανείς δεν μπορεί να δημιουργήσει αντικείμενα της κλάσης, εκτός από εσάς (μέσω από ένα στατικό μέλος της κλάσης).

Τροποποιητές Πρόσβασης για Κλάσεις

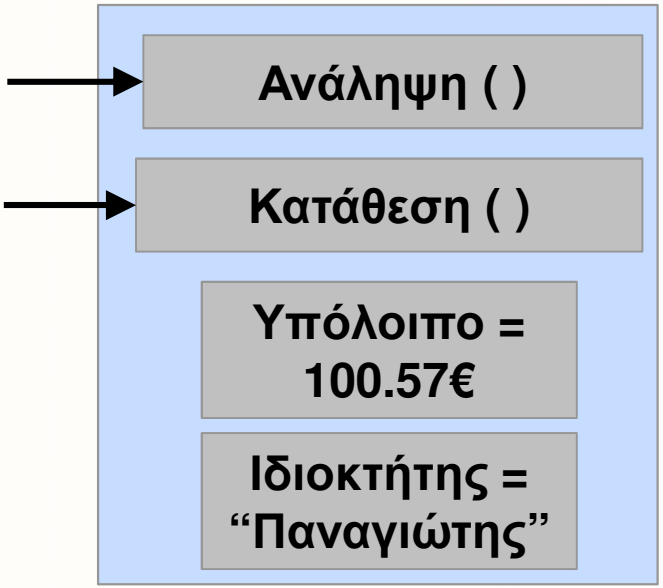
- Οι προσδιοριστές πρόσβασης μπορούν να χρησιμοποιηθούν και για τον καθορισμό των κλάσεων μιας βιβλιοθήκης που είναι προσβάσιμες είτε μέσα από την ίδια βιβλιοθήκη είτε από άλλες βιβλιοθήκες.
- Μια κλάση μπορεί να προσδιορισθεί **μόνο** σαν φιλική ή σαν δημόσια.
  - **public: ΟΛΟΙ** έχουν πρόσβαση στην κλάση
  - **<no modifier>: Μόνο κλάσεις στο ίδιο πακέτο** έχουν πρόσβαση στην κλάση
- Κανονικές (μη εσωτερικές) κλάσεις **δεν** μπορούν να συνδυαστούν με προσδιοριστές πρόσβασης **private** ή **protected.**

Τροποποιητές Πρόσβασης για Κλάσεις (συν.)

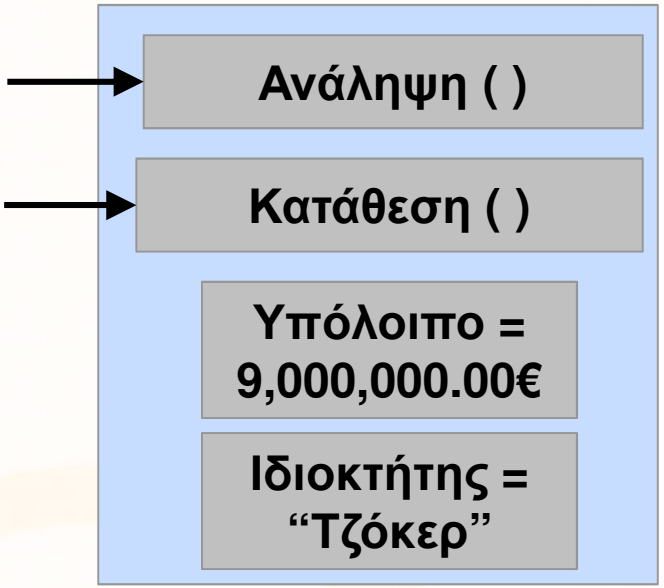
- Το πολύ μια **public** κλάση σε μια μονάδα μετάφρασης (αρχείο Java).
- Το όνομα της **public** κλάσης πρέπει να είναι το ίδιο με το όνομα του αρχείου που την περιλαμβάνει.
- Είναι αποδεκτό (αλλά όχι συνηθισμένο) να έχουμε μια μονάδα μετάφρασης χωρίς δημόσια κλάση στο εσωτερικό της. Στην περίπτωση αυτή, το όνομα του αρχείου μπορεί να είναι οποιοδήποτε (καλό είναι, όμως, να μην χρησιμοποιούμαι άσχετα ονόματα).

Στατικότητα (static)

- Τα δεδομένα ενός αντικείμενου περιγράφουν πληροφορίες για το ίδιο το αντικείμενο
  Παράδειγμα: Κάθε λογαριασμός τράπεζας έχει το δικό του υπόλοιπο



BankAccount **ba1**

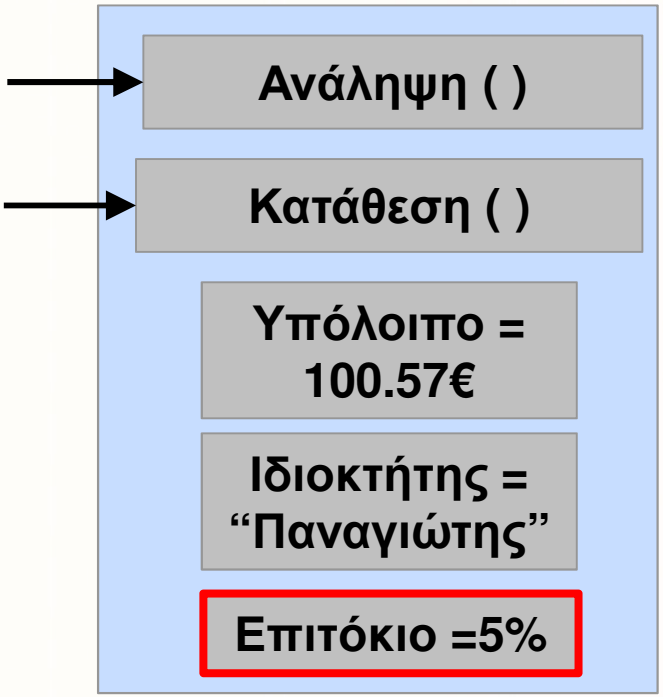


BankAccount **ba2**

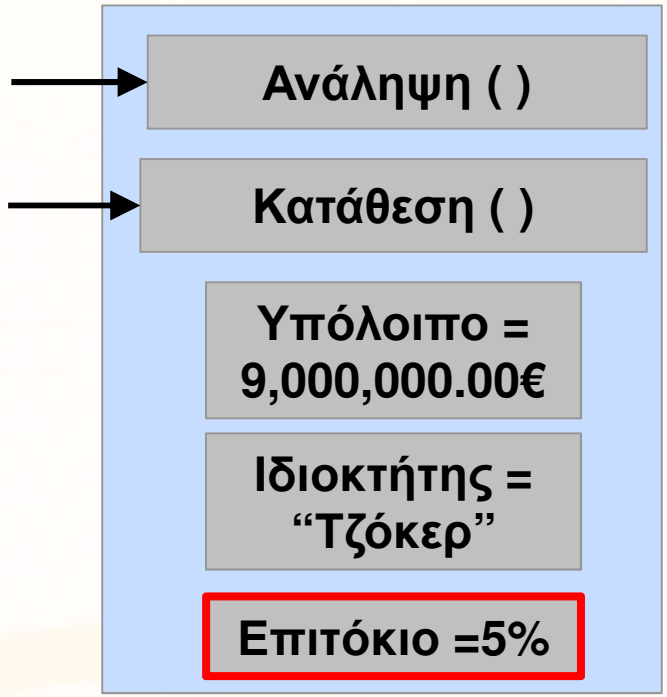
- **Ερώτηση:** Τι γίνεται αν κάποιες πληροφορίες/δεδομένα ισχύουν για όλους τους τραπεζικούς λογαριασμούς; Για παράδειγμα, το επιτόκιο όλως των λογαριασμών είναι 5%

Στατικότητα (static) (συν.)

- **Ερώτηση:** Θα ήταν σωστή η ακόλουθη υλοποίηση;



BankAccount **ba1**

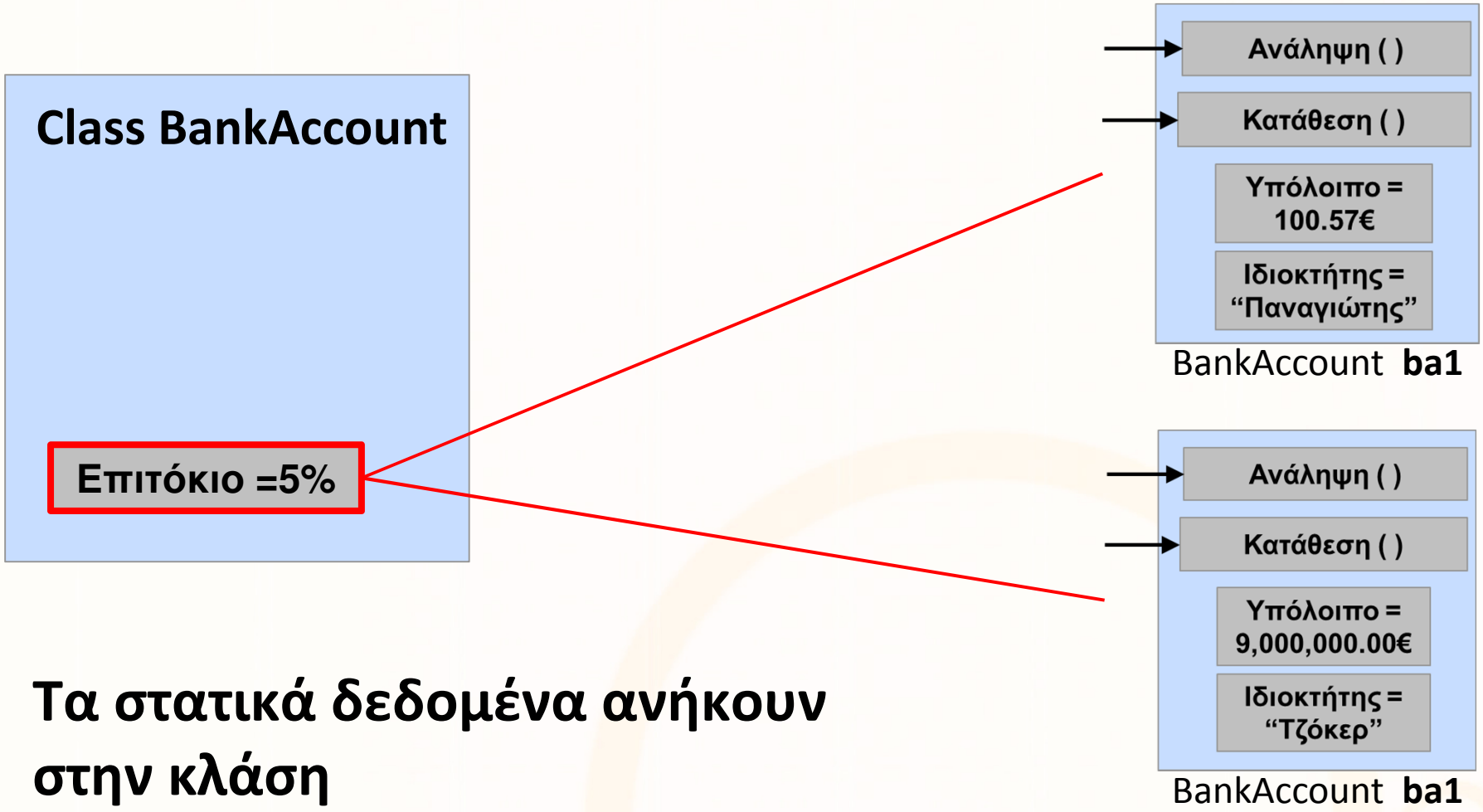


BankAccount **ba2**

- **Απάντηση:** ΟΧΙ! Με την πιο πάνω υλοποίηση επιτρέπουμε αλλαγή στο επιτόκιο κάθε λογαριασμού. Πρόβλημα αφού **ΓΙΑ ΟΛΟΥΣ ΤΟΥΣ ΛΟΓΑΡΙΑΣΜΟΥΣ ΙΣΧΥΕΙ ΕΠΙΤΟΚΙΟ ΤΟΥ 5%**

Στατικότητα (static) (συν.)

- Τα **στατικά δεδομένα (static)** περιγράφουν πληροφορίες που **ισχύουν για όλα τα αντικείμενα της κλάσης** τους



- **Τα στατικά δεδομένα ανήκουν στην κλάση**

Στατικά Δεδομένα/Μεταβλητές

- Τα **στατικά δεδομένα (static)** περιγράφουν πληροφορίες που **ισχύουν για όλα τα αντικείμενα της κλάσης** τους
- Τα στατικά δεδομένα **ανήκουν στην κλάση και όχι στο αντικείμενο**
- Ονομάζονται και **class data**
- Οι στατικές μεταβλητές **αρχικοποιούνται μόνο μία φορά**, στην αρχή της εκτέλεσης του προγράμματος και πριν την δημιουργία οποιουδήποτε αντικείμενου της κλάσης
- **Ένα αντίγραφο** μίας στατικής μεταβλητής **κατανέμεται σε όλα τα αντικείμενα (στιγμιότυπα) της κλάσης**
- Μπορούμε να έχουμε **πρόσβαση** σε μία στατική μεταβλητή **απευθείας**, δεν χρειάζεται αντικείμενο
  Σύνταξη: **<class-name>.<variable-name>**

Στατικά Δεδομένα/Μεταβλητές: Παράδειγμα 1

```
public class BankAccount {
  private double balance;
  public static double interest = 0.05;
  …
}
```

```
public class TestBankAccount {
  public static void main(String args[])
{

    BankAccount.interest = 0.06;

  }
}
```

- Απευθείας Πρόσβαση
- Δεν χρειάζεται να δηλώσουμε αντικείμενο τύπου BankAccount

Στατικά Δεδομένα/Μεταβλητές: Παράδειγμα 2

```
public class BankAccount {
  private double balance;
  public static double interest = 0.05;
  …
}
```

```
public class TestBankAccount {
  public static void main(String args[]) {
    BankAccount ba1 = new BankAccount();
    System.out.println(ba1.interest);
    BankAccount ba2 = new BankAccount();
    System.out.println(ba2.interest);
    // OR ba2.interest OR BankAccount.interest
    ba1.interest = 0.06;

    System.out.println(ba2.interest);
  }
}
```

0.05

0.05

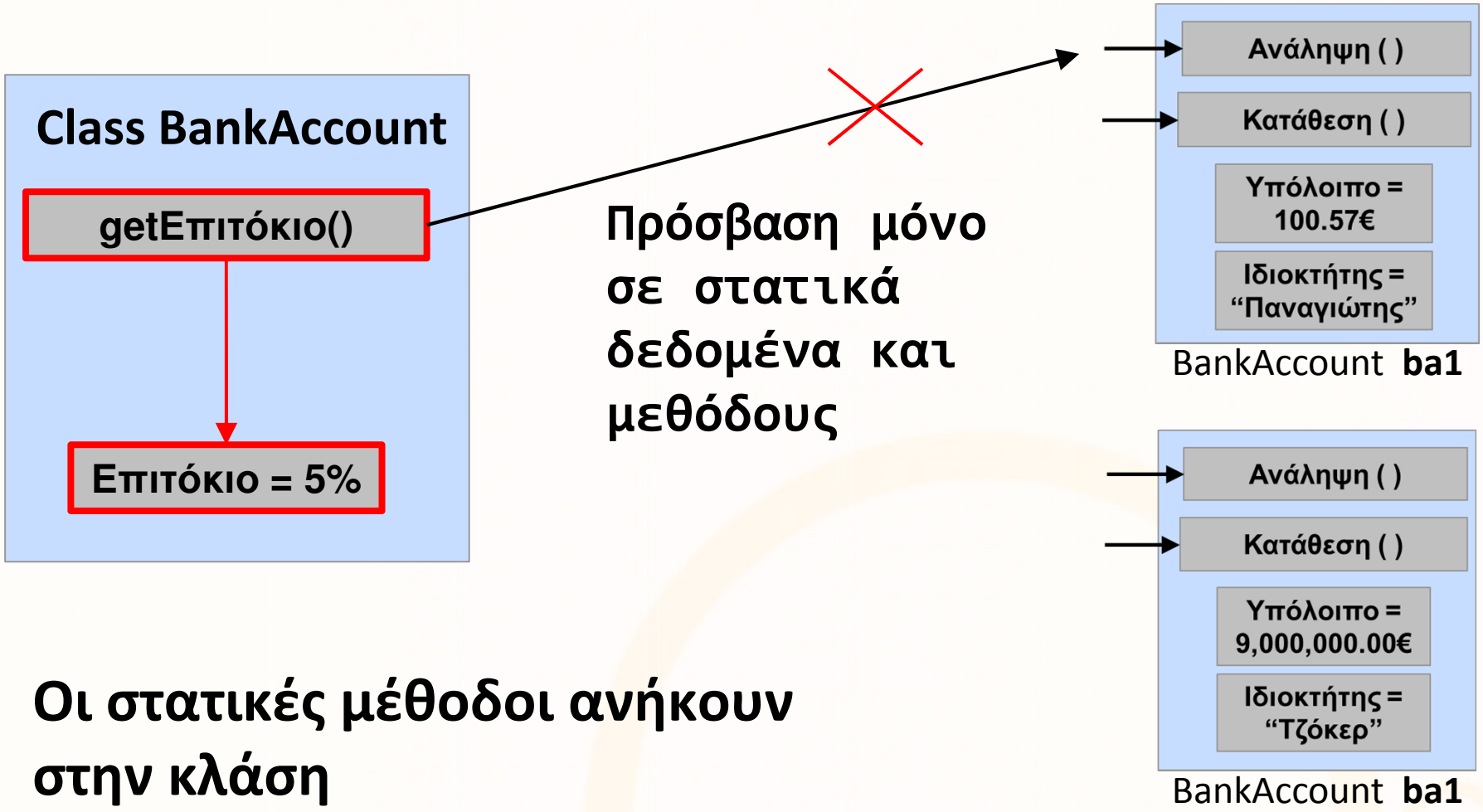
0.06

Στατικές Μέθοδοι

- Οι **στατικές μέθοδοι ισχύουν για όλα τα αντικείμενα της κλάσης**
- Ονομάζονται και **class methods**
- Οι στατικές μέθοδοι **μπορούν να έχουν πρόσβαση ΜΟΝΟ σε στατικά δεδομένα.**
- Οι στατικές μέθοδοι **μπορούν να καλέσουν ΜΟΝΟ άλλες στατικές μεθόδους**
- Μπορούμε να έχουμε **πρόσβαση** σε μία στατική μέθοδο **απευθείας**, δεν χρειάζεται αντικείμενο,
  Σύνταξη: **<class-name>.<method-name>**
- **Πιο γνωστό παράδειγμα:** `public static void main()`

Στατικές Μέθοδοι (συν.)

- Οι **στατικές μέθοδοι είναι λειτουργίες τις κλάσης** που **ισχύουν για όλα τα αντικείμενα της κλάσης**



- **Οι στατικές μέθοδοι ανήκουν στην κλάση**

Στατικές Μέθοδοι: Παράδειγμα 1

```
public class StaticMethods {
  int var1;
  static int var2;

  public static void main( String args[]) {
    // NOT OK var2 is NOT static
    var1 = 5;

    // OK var2 is static
    var2 = 5;
  }

}
```

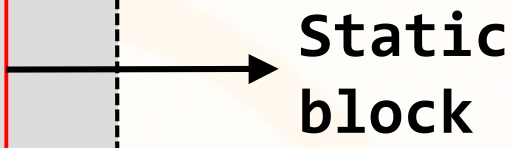
Παράδειγμα 2: Στατικές Μέθοδοι

```
public class StaticMethods {
 public void method1( ) { }
 public static void method2( ) { }

  public static void main(String args[]) {
    // NOT OK method1() is NOT static
    method1( );

    // OK method2() is static
    method2( );
  }

}
```

Στατικό Τμήμα Κώδικα (static code block)

- Το στατικό τμήμα κώδικα, είναι ένα τμήμα με δηλώσεις μέσα σε μία κλάση το οποίο εκτελείται αμέσως μόλις φορτωθεί η κλάση στο JVM
  - Ερώτηση 1: Τι θα τυπωθεί στο πιο κάτω πρόγραμμα;
  - Ερώτηση 2: Τι θα τυπωθεί αν αφαιρέσουμε όλη τη δήλωση new;

```
public class StaticBlock {
 public StaticBlock() {
   System.out.println("Constructor");
 }
 static {
   System.out.println("static block");
 }
 public static void main(String args[]) {
   StaticBlock obj = new StaticBlock();
 }
}
```



Στατική Εισαγωγή Βιβλιοθήκης (static import)

- Παράδειγμα πρόσβασης σε στατικά μέλη μιας κλάσης:

```
double r = Math.cos(Math.PI * theta);
```

- Με τη χρήση στατικής εισαγωγής μπορούμε να κάνουμε χρήση των στατικών πεδίων μιας κλάσης, χωρίς να χρησιμοποιούμε το όνομα της κλάσης στην οποία ανήκουν:
- Παράδειγμα

```
import static java.lang.Math.PI;
import static java.lang.Math.*;
double r = cos(PI * theta);
```

- Η δήλωση στατικής εισαγωγής εισάγει τα στατικά μέλη μιας κλάσης (ή κλάσεων) στο πρόγραμμά μας.

Τροποποιητής (final)

- Ο τροποποιητής **final** χρησιμοποιείται για να καθορίσει ότι μία οντότητα (κλάση, μεταβλητή, μέθοδος) δεν μπορεί να αλλάξει σε μεταγενέστερο επίπεδο
- Σύνταξη: `<other modifiers> final <name>`
- **Ανάθεση** στις final μεταβλητές **μπορεί να γίνει μόνο μία φορά**
- Όταν τελειώσει η ανάθεση, **οι μεταβλητές δεν μπορούν να αλλαχθούν**
- **Οι κλάσεις** που ορίζονται σαν **final, δεν μπορούν να κληρονομηθούν**

Τροποποιητής (final)

- **Οι μέθοδοι που ορίζονται σαν final δεν μπορούν να επικαλυφθούν (κληρονομικότητα)**
- **Αντίθετα με τις μεταβλητές τύπου constant**, δεν είναι ανάγκη να γνωρίζουμε την τιμή των final κατά την ώρα τις μεταγλώττισης
- Αν δεν γίνει ανάθεση στην μεταβλητή final τότε οι κατασκευαστές αναγκάζονται να την αρχικοποιήσουν
- Αν μία μεταβλητή δηλωθεί σαν static final τότε η compiled κλάση τρέχει πιο γρήγορα.
- Γνωστό παράδειγμα final κλάσης: **`java.lang.Math`**

Παράδειγμα 1: final μεταβλητές

```
public class FinalVariables {
  final int var1 = 10;
  final int var2 = 20;
  final int var3;

  FinalVariables(){
    // NOT OK var2 already has value
    var2 = 10;
    // OK var3 was not initialized
    var3 = 30;
  }
  public static void main(String args[]) {
    FinalVariables obj = new FinalVariables();
  }
}
```

Παράδειγμα 2: final κλάσεις

```
public final class Math { }

// NOT OK Math cannot be inherited
public class MyMath extends Math{ }
```

Παράδειγμα 3: final μεθόδοι

```
public class FinalMethods {
  public void method() {}

  public final void final_method() {}
}

public class MyClass extends FinalMethods {
  // OK method() can be overriden
  public void method() {}

  // NOT OK final_method() cannot be overriden
  // declared as final
  public void final_method() {}
}
```

Άλλοι τροποποιητές

- **abstract:** δηλώνει ότι κάτι πρέπει να υπερκαλυφθεί (override)
- **static:** δηλώνει ότι μόνο ένα στιγμιότυπο μπορεί να υπάρξει
- **final:** δηλώνει ότι η οντότητα δεν μπορεί να αλλαχθεί
- **strictfp:** επιβολή ότι πρέπει να χρησιμοποιηθεί αυστηρή συμπεριφορά floating point
- **native:** δηλώνει ότι μία οντότητα είναι υλοποιημένη σε μία άλλη γλώσσα προγραμματισμού (π.χ., C, C++)
- **synchronized/transcient/volatile:** χρησιμοποιούνται από πολυνηματικές εφαρμογές
- **<Annotations> (@):** δηλώνει σχόλια ειδικής μορφής