Διάλεξη 13:
Επανάληψη για ενδιάμεση εξέταση

Διδάσκων: Παναγιώτης Ανδρέου

Προγράμματα

Τα προγράμματα (*software)*, είναι βασικά ένα σύνολο από οδηγίες στον υπολογιστή. Χωρίς τα προγράμματα ο υπολογιστής είναι μία άδεια μηχανή.

Δεδομένου ότι ο υπολογιστής δεν καταλαβαίνει την φυσική γλώσσα αλλά μόνο την γλώσσα μηχανής (machine language), χρειαζόμαστε κάποιες ειδικές γλώσσες (γλώσσες προγραμματισμού) για να επικοινωνήσουμε μαζί του.

Παραδείγματα γλωσσών προγραμματισμού: C, C++, JAVA, C#, Visual Basic, Fortran, κ.τ.λ.

ΠΑΡΑΔΕΙΓΜΑ ΠΡΟΓΡΑΜΜΑΤΟΣ σε JAVA

```
public class HelloWorld{
   public static void main(String[] args){
      System.out.println("Hello World!!!");
   }
}
```

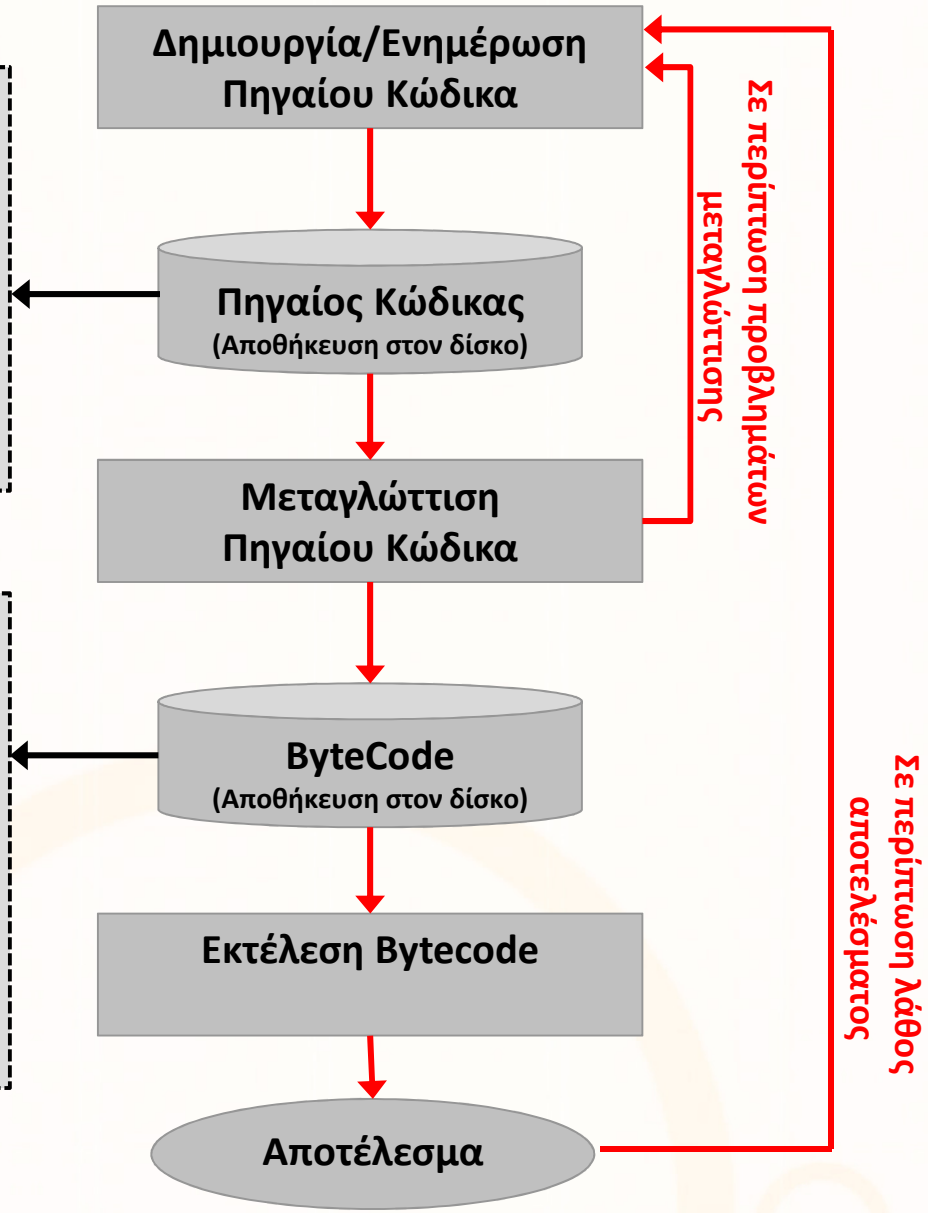
Τυπικός Κύκλος Ζωής ενός Προγράμματος JAVA

ΠΑΡΑΔΕΙΓΜΑ ΠΗΓΑΙΟΥ ΚΩΔΙΚΑ

```
public class HelloWorld{
   public static void main(String[] args){
      System.out.println("Hello World!!!");
   }
}
```

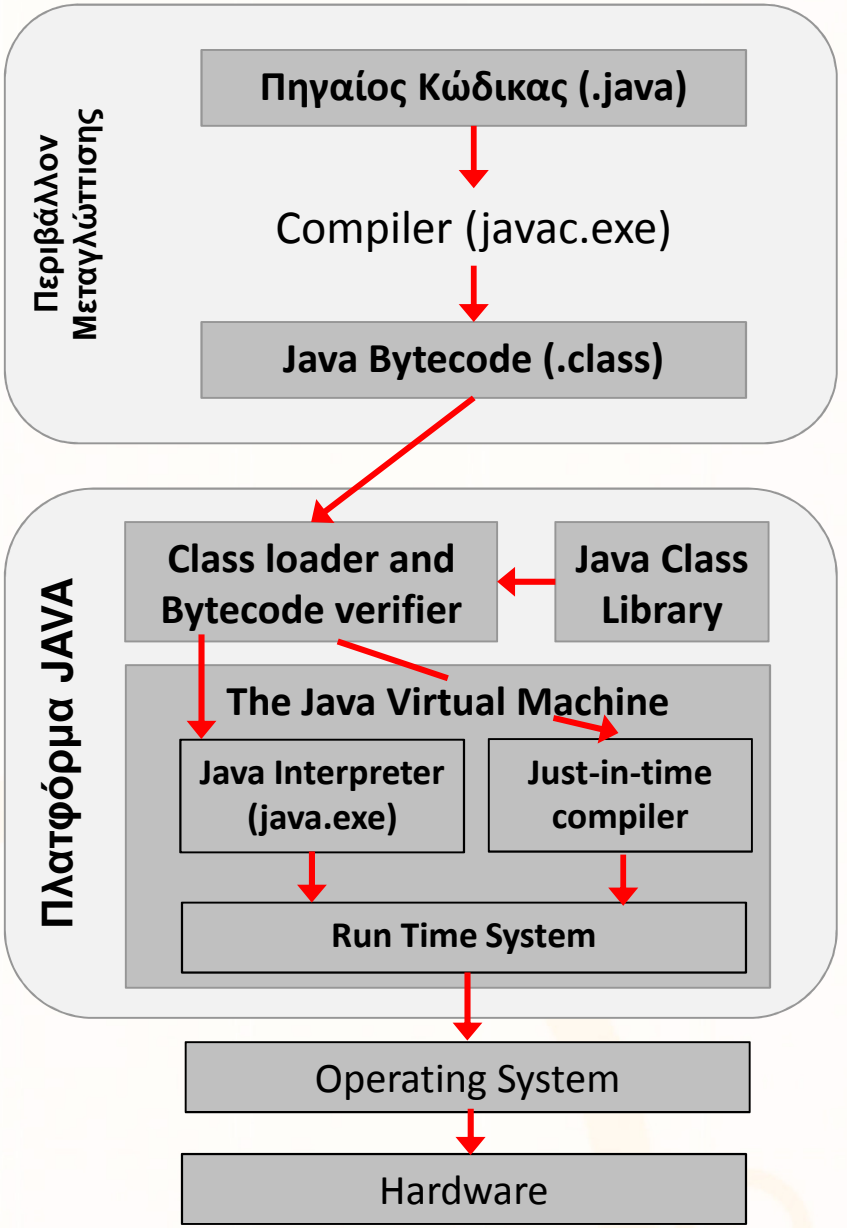
ΠΑΡΑΔΕΙΓΜΑ BYTE CODE

```
…
Method HelloWorld()
  0       aload_0
…
Method void main(java.lang.String[]){
  0  getstatic #2 …
  3  ldc #3 < String "Hello World!!!" >
  5  invokevirtual #4
…
```



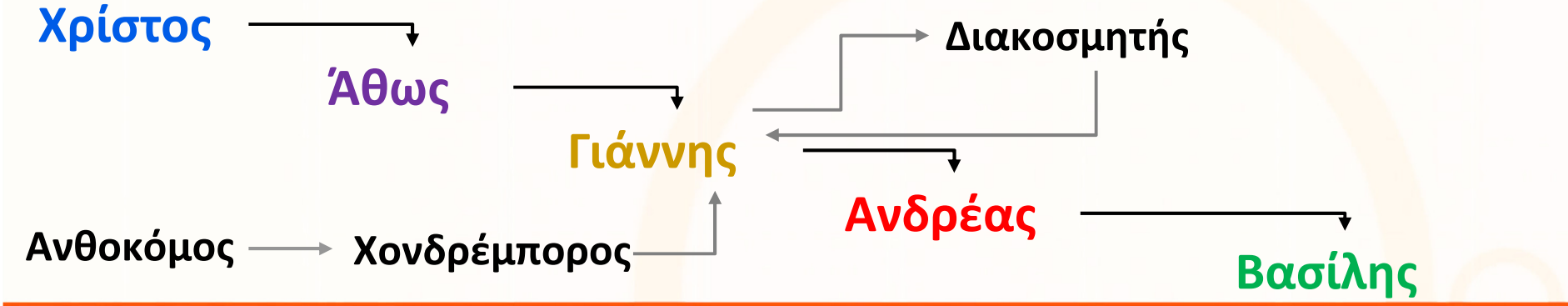
Η πλατφόρμα της JAVA

- Πλατφόρμα:
  - Περιβάλλον λογισμικού και υλικού στο οποίο εκτελείται ένα πρόγραμμα.
  - Συνήθως είναι συνδυασμός του Λειτουργικού Συστήματος και του Υλικού Υποστρώματος του ΛΣ.
  - Δημοφιλείς πλατφόρμες: Microsoft Windows, Linux, Solaris OS, Mac OS.
- Πλατφόρμα Java: Σύστημα λογισμικού που τρέχει πάνω σε διάφορες πλατφόρμες υλικού. Αποτελείται από:
  - Την Εικονική Μηχανή JAVA: Java Virtual Machine
  - Την Προγραμματιστική Διαπροσωπεία Εφαρμογών της JAVA (Java Application Programming Interface - API)



Το παράδειγμα της αποστολής λουλουδιών

- Ο **Χρίστος** θέλει να στείλει λουλούδια στον **Βασίλη**, ο οποίος βρίσκεται σε άλλη πόλη.
- Ο **Χρίστος** δεν μπορεί να παραδώσει τα λουλούδια ο ίδιος, έτσι χρησιμοποιεί τις υπηρεσίες κάποιου ανθοπώλη (**Άθως**).
- Ο **Χρίστος** λέει στον **Άθω** τη διεύθυνση του Βασίλη, το ποσό που μπορεί να διαθέσει και το είδος των λουλουδιών.
- Ο **Άθως** με τη σειρά του επικοινωνεί με άλλο ανθοπώλη (**Γιάννης**) στην πόλη του **Βασίλη**, του μεταφέρει τις πληροφορίες. Ο **Γιάννης** ετοιμάζει την παραγγελία και ενημερώνει τον αποστολέα (**Ανδρέας**).
- Ο **Ανδρέας** παραδίδει τα λουλούδια.



Ιδιότητες Αντικειμενοστρεφή Προγραμματισμού

- Κλάσεις & Στιγμιότυπα
  - Μία κλάση είναι όπως ένα αρχιτεκτονικό σχέδιο: περιγράφει τις ιδιότητες ενός αντικειμένου
  - Από μία κλάση μπορούν να δημιουργηθούν πολλά στιγμιότυπα (αντικείμενα)
  - Κάθε κλάση περιγράφει μία μοναδική οντότητα με ξεχωριστές ιδιότητες στο πρόγραμμα.
  - Παράδειγμα: Ο Άθως και ο Γιάννης είναι και οι δύο ανθοπώληδες

Κλάση

Στιγμιότυπα

```
public class Anthopolis{
…

Anthopolis Giannis;

Anthopolis Athos;
```

Ιδιότητες Αντικειμενοστρεφή Προγραμματισμού

- Ενθυλάκωση (Encapsulation)
  - Περιορίζει την πρόσβαση στα δεδομένα με τροποποιητές πρόσβασης.
  - Παράδειγμα: Το όνομα του Χρίστου μπορεί να το αλλάξει μόνο ο Χρίστος
- Αφαιρετικότητα (Abstraction)
  - Κάθε αντικείμενο έχει συγκεκριμένο ρόλο και ευθύνες.
  - Παράδειγμα: Ο Γιάννης και ο Άθως είναι μόνο ανθοπώληδες, ο Ανδρέας είναι μόνο αποστολέας.
  - Αυτό φυσικά εναπόκειται στις καλές σχεδιαστικές ικανότητες του προγραμματιστή

```
public class Person{
  private name;

  private changeName(String s){
      name=s;
}
…
```

```
public class Anthopolis{
   private createBouquet(int
         noOfFlowers, Color c){
      …
   }
…
public class Apostoleas{
   private sendBouquet(Flowers
f, Address a){
      …
   }
…
```

Ιδιότητες Αντικειμενοστρεφή Προγραμματισμού

- Κληρονομικότητα
  - Περιορίζει την πρόσβαση στα δεδομένα με τροποποιητές πρόσβασης
  - Παράδειγμα: Ο Ανδρέας είναι άτομο, συνεπώς κληρονομεί τα χαρακτηριστικά ενός ατόμου.
- Πολυμορφισμός
  - Υποστηρίζει πολλές μεθόδους με το ίδιο όνομα, αλλά με διαφορετικές παραμέτρους
  - Παράδειγμα: Ο Ανδρέας μπορεί να στείλει τα λουλούδια με διάφορους τρόπους:
    - Με ποδήλατο
    - Με αυτοκίνητο
    - Με αυτοκίνητο σε συγκεκριμένη ώρα

```
public class Person{
  protected name;

 protected changeName(String s){
       name=s;
}
…
public class Anthopolis inherits
Person{
…
```

```
public class Anthopolis{
 private sendFlowers (
       (Flowers x, Bicycle c){…}
 private sendFlowers (
       (Flowers x, Car c){…}
 private sendFlowers (
       (Flowers x, Car c, Time
t){…}
…
```

Παραδείγματα: Ένα απλό πρόγραμμα JAVA

```
public class HelloWorld{

  //Αρχικό Σημείο του Προγράμματος
  //Ο ερμηνευτής της JAVA εκτελεί το
  //πρόγραμμα κάνωντας κλήση στην
  //μέθοδο main
  public static void main(String[] args){

    System.out.println("Hello World!!!");

  }

}
```

→ **Παράδειγμα Δήλωσης**

Παραδείγματα: Κλάση

```
class Circle {
  //Η ακτίνα αυτού του κύκλου
  double radius = 1.0;

  //Δημιούργησε ένα αντικείμενο τύπου κύκλος
  Circle () {
  };

  //Δημιούργησε ένα αντικείμενο τύπου κύκλος
  //με συγκεκριμένη ακτίνα
  Circle (double newRadius) {
    radius = newRadius;
  };

  //Επέστρεψε το εμβαδό αυτού του κύκλου
  double getArea () {
    return radius * radius * π;
  }

}
```

Δεδομένα/ Μεταβλητές

Κατασκευαστές

Μέθοδοι

Παραδείγματα: Αντικείμενα

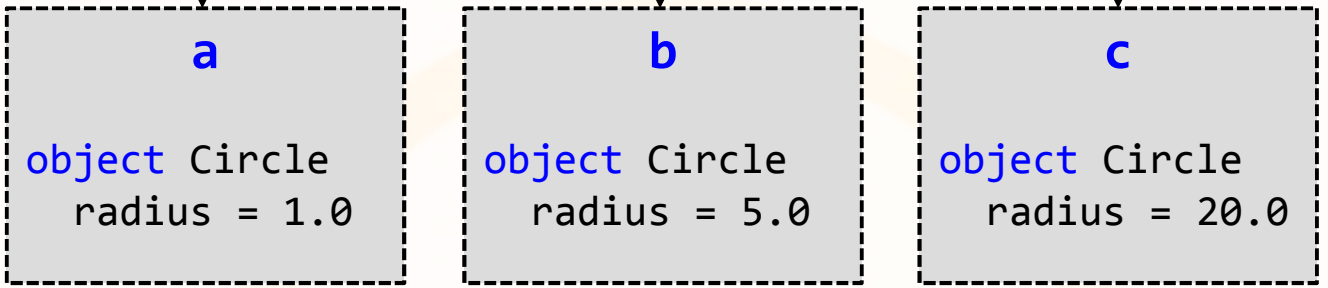
Κλάση Circle

```
class Circle {
  //Η ακτίνα αυτού του κύκλου
  double radius = 1.0;

  //Δημιούργησε ένα αντικείμενο τύπου κύκλος
  Circle () {
  };

  //Δημιούργησε ένα αντικείμενο τύπου κύκλος
  //με συγκεκριμένη ακτίνα
  Circle (double newRadius) {
    radius = newRadius;
  };

  //Επέστρεψε το εμβαδό αυτού του κύκλου
  double getArea () {
    return radius * radius * π;
  }

}
```

Κλάση Test: περιλαμβάνει την μέθοδο main

```
public class Test{

  public static void main(String[] args){
    Circle a = new Circle();
    Circle b = new Circle(5);
    Circle c = new Circle(20);


…
```

3 αντικείμενα της κλάσης Circle



Κατασκευή αντικειμένων

- Η δήλωση ενός αντικειμένου κάποιου τύπου, π.χ., Circle a δεν σημαίνει και δέσμευσή του στη μνήμη.
- Η κατασκευή/δημιουργία ενός νέου αντικειμένου γίνεται με τη χρήση της εντολής **new**.
  - `Circle a = new Circle();`
  - `Circle a = new Circle(5.0);`
  - `String s = new String(‘‘asdf’’);` ⇔ `String s = ‘‘asdf’’;`
- Με την κλήση του new, εκτελούνται κάποιες ειδικές μέθοδοι που ονομάζονται **κατασκευαστές (constructors)** και είναι υπεύθυνοι για την δημιουργία του αντικειμένου και αρχικοποίηση των πεδίων του.
- Παράδειγμα:

```
class Circle {
  …
  Circle () {
  };
  …
```

Κατασκευαστής

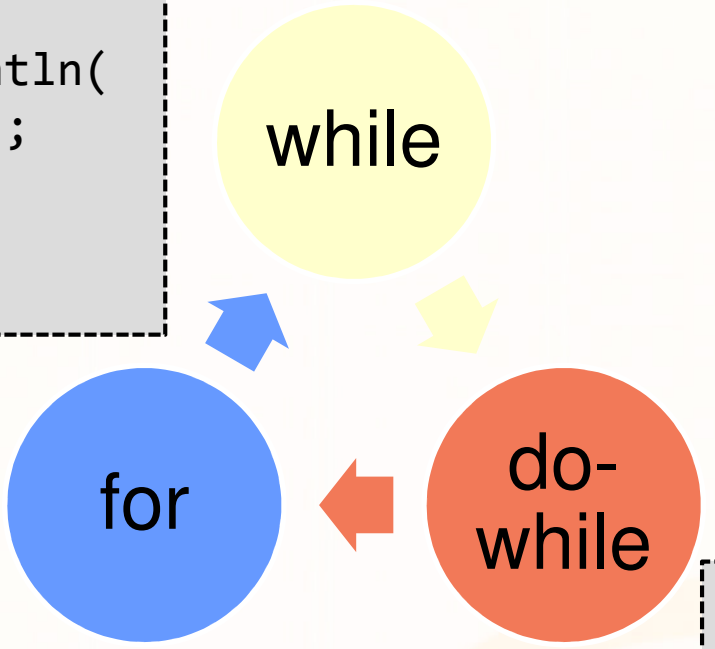
Μετατροπές Τύπων/Μεταβλητών (συν.)

- **Έμμεση Μετατροπή (implicit casting)**
  - Παράδειγμα 1
    - double d = 3;
    - Διεύρυνση/Μεγέθυνση τύπου int → double
    - Κανένα πρόβλημα στη μεταγλώττιση
  - Παράδειγμα 2
    - int i = 3.0;
    - Πρόβλημα στην μεταγλώττιση double → int
    - Συμβαίνει όταν γίνεται ανάθεση ενός πιο μικρού τύπου σε μεγαλύτερο (συρρίκνωση)
- **Άμεση Μετατροπή (explicit casting)**
  - int i = (int) 3.0;
  - Κανένα πρόβλημα στη μεταγλώττιση
  - Συρρίκνωση δεδομένων
  - Μπορεί να γίνει αφαίρεση κλάσματος/δεκαδικών, π.χ., int i = (int) 3.9; → i=3.
- Τι θα γίνει με το ακόλουθο; int x = 5 / 2.0; →→int x = (int) (5 / 2.0);

Παραδείγματα προγραμματισμού με βρόγχους

Παράδειγμα <while>

```
int i = 0;
while (i < 10) {
  System.out.println(
          "while");
  i ++;
}
```



Παράδειγμα <do-while>

```
int i = 0;
do {
  System.out.println(
          "do-while");
  i++;
} while (i < 10);
```

Παράδειγμα <for>

```
for ( int i=0; i<10; i++) {
  System.out.println(
          "for");
}
```

Παραδείγματα: break, continue και return

```
for (i = 0; i < SIZE; i++) {
  …
  break;
  …
}
…
```

```
for (i = 0; i < SIZE; i++) {
  …
  continue;
  …
}
…
```

```
search1:   //search1 is a label
for (i = 0; i < SIZE; i++) {
  for (j = 0; j < SIZE; j++) {
    …
    break search1;
    …
  }
  …
}
…
```

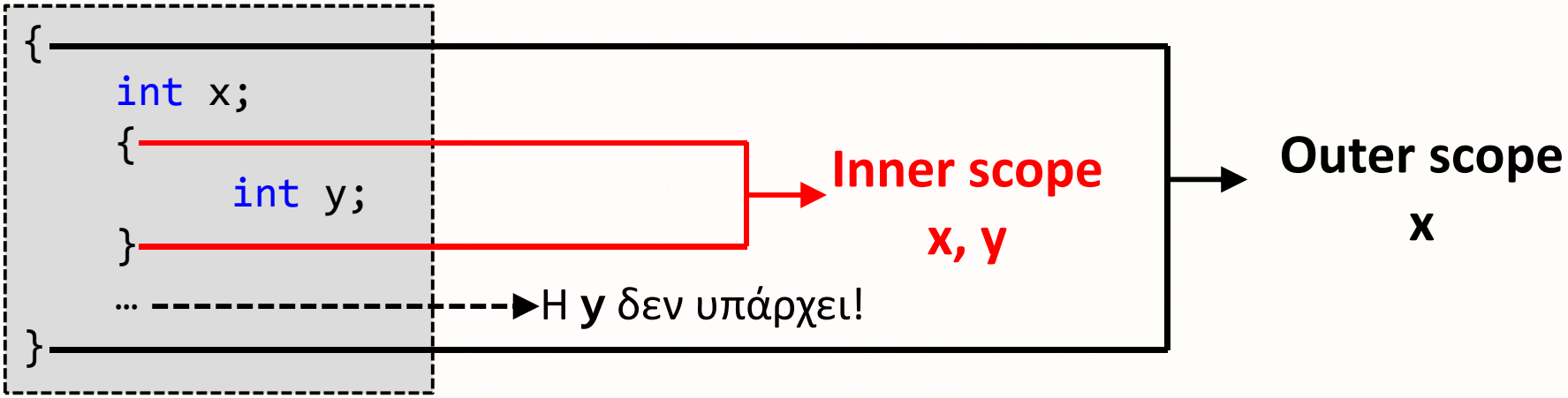
```
search1:   //search1 is a label
for (i = 0; i < SIZE; i++) {
  for (j = 0; j < SIZE; j++) {
    …
    continue search1;
    …
  }
  …
}
…
```

```
someMethod() {
    … return; … }
```

→ **Exit someMethod()**

Πεδίο/Εμβέλεια ισχύος (scope)

- Οι μεταβλητές που ορίζονται σε κάποιο Πεδίο/Εμβέλεια ισχύος (scope), είναι υπαρκτές μέχρι το τέλος του πεδίου αυτού. Παράδειγμα:

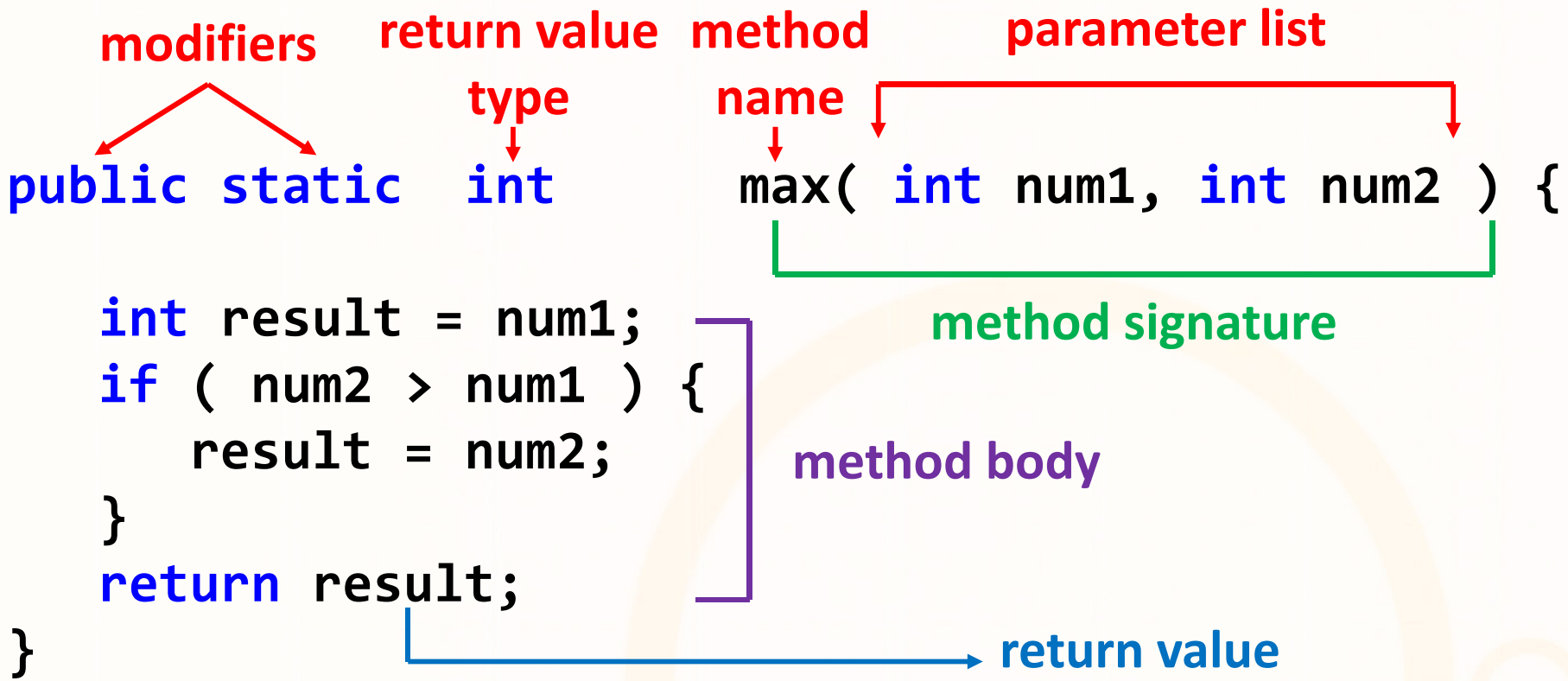


- Σε αντίθεση με την C και C++, το ακόλουθο είναι συντακτικό σφάλμα

```
{
    int x;
    {
        int x;
    }
}
```

Μέθοδοι (methods)

- **Μέθοδος:** μία συλλογή από ομαδοποιημένες δηλώσεις οι οποίες εκτελούν κάποια (ες) λειτουργία (ες).
- Η **υπογραφή μίας μεθόδου** αποτελείται από το **όνομά της** και τη **λίστα με της παραμέτρους** που δέχεται.
- Σύνταξη:



Boxing

- **Οι συλλογές ΔΕΝ δέχονται πρωτόγονους τύπους αλλά μόνο δείκτες σε αντικείμενα (reference types).**
- Ερώτηση: Τι γίνεται όταν χρειαζόμαστε πρωτόγονους τύπους;
- Απάντηση: Boxing

Ορισμοί:

- **Box**
  Ένα στιγμιότυπο μίας κλάσης περιτυλίγματος (wrapper) η οποία αποθηκεύει την τιμή ενός πρωτόγονου τύπου.
- **Boxing**
  Δημιουργία ενός box για μία τιμή πρωτόγονος τύπου
- **Unboxing**
  Επιστροφή της τιμής πρωτόγονου τύπου από το box

Manual/Auto boxing and unboxing

- Οι αρχέγονοι τύποι δεν μπορούν να χρησιμοποιηθούν στις περισσότερες περιπτώσεις -**you need a "wrapper"**
  - myVector.add(new Integer(5));
- Αντίστοιχα δεν επιτρέπεται να χρησιμοποιηθεί ένα αντικείμενο εκεί που χρειάζεται αρχέγονος τύπος. --**you need to "unwrap" it**
  - int n = ((Integer)myVector.lastElement()).intValue();

- Integer iNumber = new Integer(10); → Manual boxing
- Integer iNumber = 10; → Auto-boxing
- iNumber = new Integer(iNumber.intValue()++); → Manual unboxing
- iNumber++; → Auto-unboxing

Το <this>

- **<this>:** είναι μία λέξη κλειδί η οποία αναφέρεται στο συγκεκριμένο στιγμιότυπο (αντικείμενο) που χρησιμοποιείται μέσα από μία μέθοδο.
- Είναι ένα χειριστήριο προς το ίδιο το αντικείμενο.
- Πολλαπλές χρήσεις:

1. Υποδηλώνει ότι θα χρησιμοποιηθεί η μεταβλητή της κλάσης και όχι κάποια static μεταβλητή ή παράμετρος από συνάρτηση

```
class Circle {
  double radius;
  Circle(  double radius) {

      this.radius = radius;
  }
```

Το <this> (συν.)

2. Κλήση κατασκευαστών από κατασκευαστές

```
class Circle {
  double radius;

  Circle() {
    this( 1.0 );
  }

  Circle(double radius) {
    this.radius = radius;
  }
```

3. Πέρασμα χειριστηρίου της κλάσης σαν παράμετρο σε μία μέθοδο

```
obj.someMethod( this ) ;
```

4. Επιστροφή χειριστηρίου από μία μέθοδο

```
ClassName someMethod(){
  return this ;
}
```

Το <this> (συν.)

Ερώτηση: Ποια βήματα θα εκτελεστούν με τον κώδικα στα δεξιά, αν δημιουργήσουμε ένα καινούριο αντικείμενο Circle χωρίς παραμέτρους;

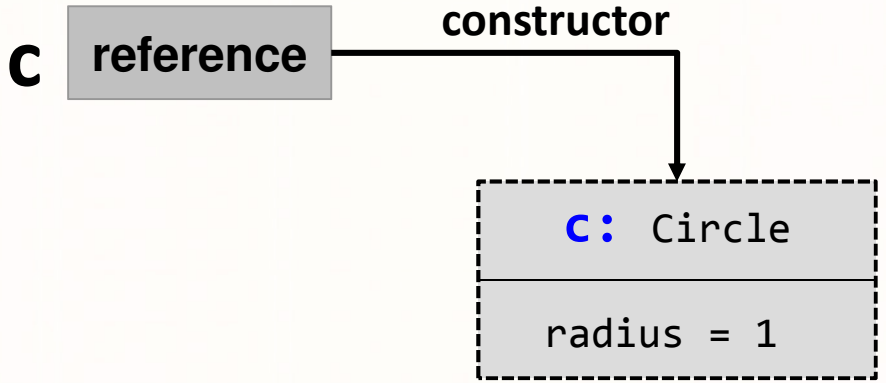
1. `Circle()` → `this( 1.0 )`
2. `Circle(double radius=1.0)`
   → `this(radius=1.0, "no name");`
3. `Circle(double radius=1.0, String name="no name")`

```
class Circle {
  double radius;
  String name;

 Circle() {
    this( 1.0 );
 }

 Circle(double radius) {
   this(radius,
        "no name");
 }

 Circle(double radius,
        String name) {
   this.radius = radius;
   this.name = name;
  }

}
```

Ανάθεση: Αρχέγονοι Τύποι vs. Τύποι Αντικειμένων

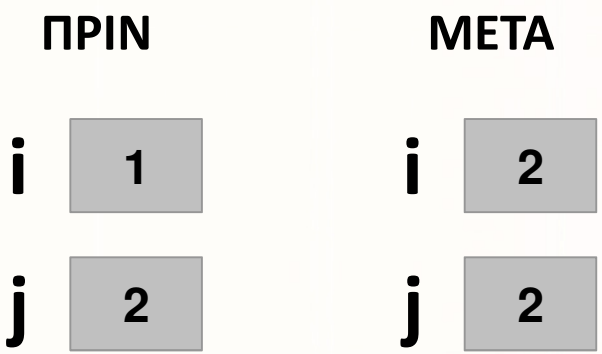
Primitive type e.g., int i = 1

Object (reference) type e.g., Circle c



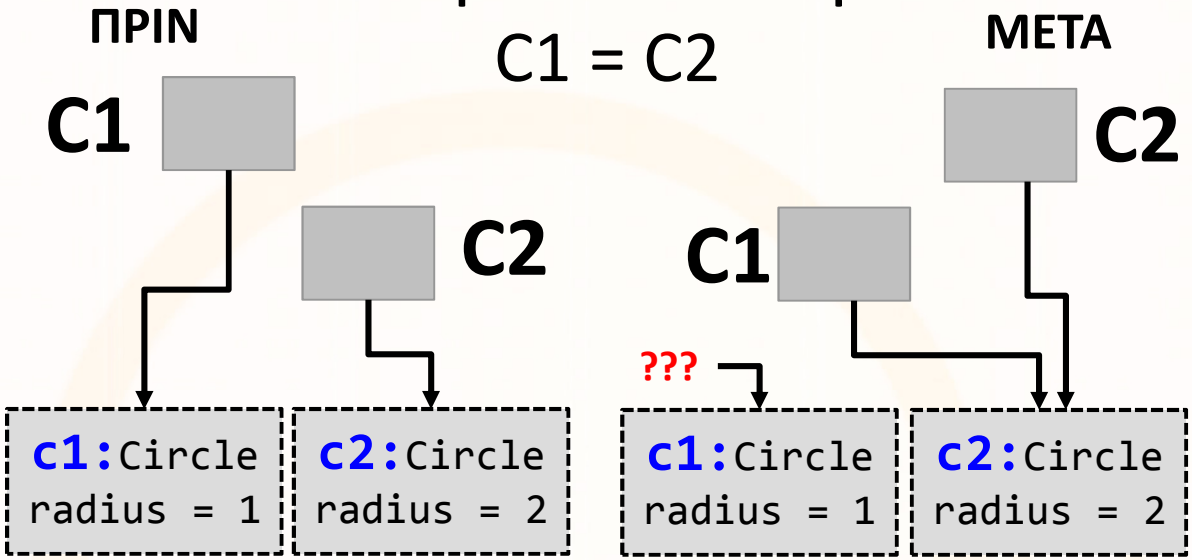
Ανάθεση σε Πρωτόγονο Τύπο

i=j



Ανάθεση σε Τύπο Αντικείμενο

C1 = C2



Αποκομιστής Σκυβάλων (garbage collector)

- Στο προηγούμενο παράδειγμα, μετά την ανάθεση c1=c2 δεν υπάρχει μεταβλητή που να δείχνει στο αντικείμενο του c1.
- Το αντικείμενο αυτό είναι για τα σκουπίδια ☺
- Η JVM συλλέγει αυτόματα τα σκουπίδια με τον garbage collector.
  - Αυτό σημαίνει αυτόματη διαχείριση μνήμης αντίθετα με C++.
  - Επίσης σημαίνει περισσότερο χρόνο για προγραμματισμό
- Ένα αντικείμενο θεωρείται επιλέξιμο από τον garbage collector εάν:
  - Όλες οι αναφορές του είναι null
  - Το αντικείμενο είναι δημιουργημένο σε ένα block και η αναφορά είναι out-of-scope μόλις τελειώσει η εκτέλεση του block
  - Το αντικείμενο έχει αναφορά από ένα αντικείμενο πατέρα μόνο, και όλες οι αναφορές στον πατέρα είναι null (αυτό ισχύει αναδρομικά)
  - Το αντικείμενο αναφέρεται μόνο μέσω ενός WeakHashMap
- Ο garbage collector έχει άμεση σχέση με τα διάφορες στάδια του κύκλος ζωής ενός αντικειμένου

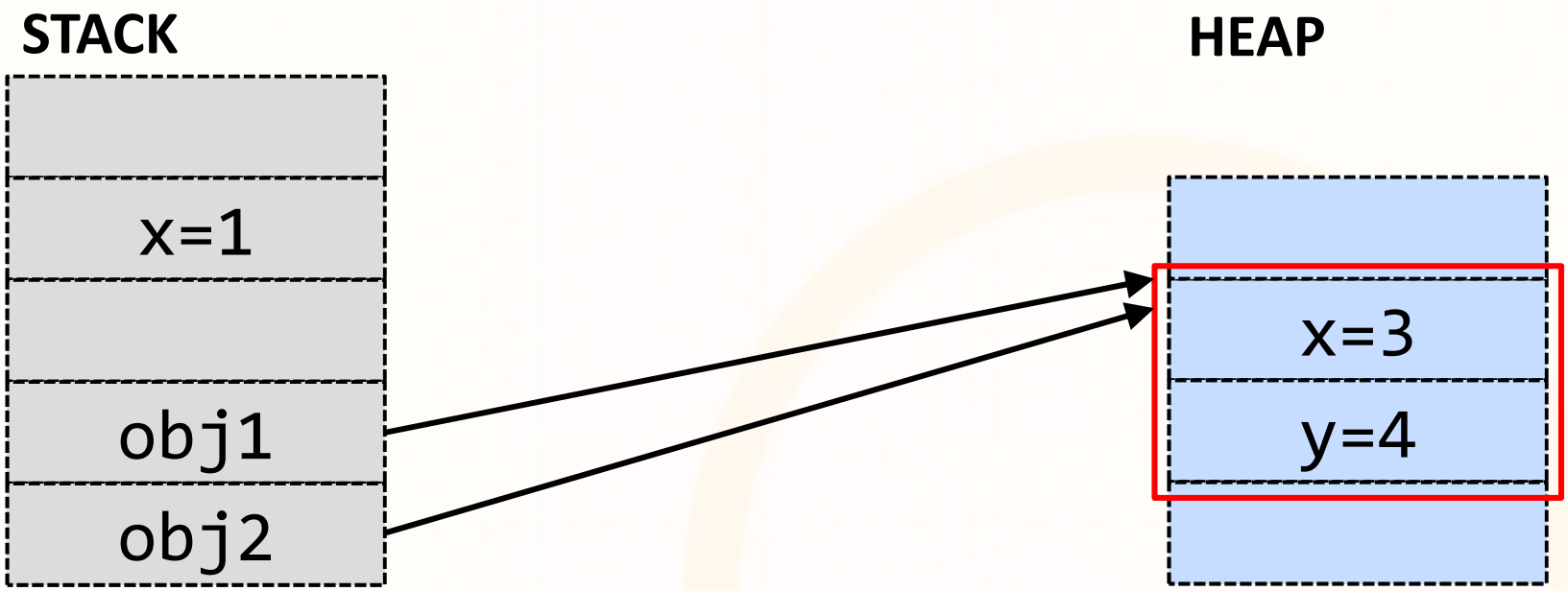
Διαχείριση Μνήμης

- **Στοίβα (stack)**
  - Μέρος της μνήμης που παρέχει ταχύτητα.
  - Μεγαλώνει/Μικραίνει ανάλογα με τις μεθόδους που καλούνται
  - Αποθηκεύει τους τοπικούς (method/block) αρχέγονους τύπος
  - Αποθηκεύει τις τοπικές (method/block) reference μεταβλητές
- **Σωρός (Heap)**
  - Πιο αργό μέρος της μνήμης που παρέχει όμως χωρητικότητα
  - Αποθηκεύει τις μεταβλητές του αντικειμένου (instanse variables)
  - Αποθηκεύει αντικείμενα που δημιουργούνται με το new
  - Μεγαλώνει/Μικραίνει ανάλογα με τα αντικείμενα που δημιουργούνται
- **Στατική περιοχή (Static Area) (μέρος του Heap)**
  - Αποθηκεύει global και static μεταβλητές

Διαχείριση Μνήμης (συν.)

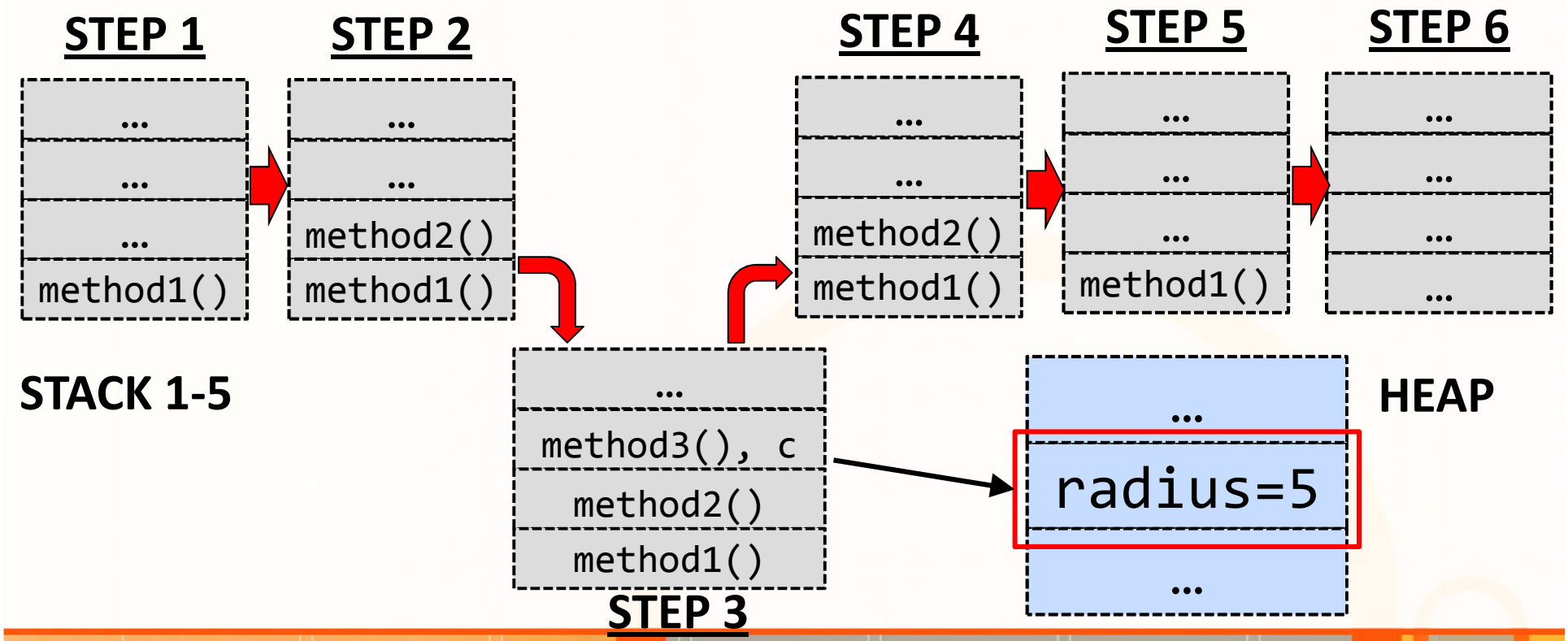
```
class Obj{
  int x;   int y;
  Obj (int a, int y) {
    x=a;
    y=b;
  }
}
```

```
…
{
 int x = 1;
 Obj obj1 = new Obj(3, 4);
 Obj obj2 = obj1;
}
…
```



Διαχείριση Μνήμης (συν.)

```
class StackMethods{
  void method1(){ method2(); }
  void method2(){ method3(); }
  void method3(){ Circle c = new Circle( 5 ); }
}
```



Βιβλιοθήκες/Πακέτα (packages)

- (Συνήθως) Ομαδοποιούμε τις κλάσεις σύμφωνα με τα αντικείμενα που αναπαριστούν και τις λειτουργίες τους.
- Ένα “πακέτο” (βιβλιοθήκη) JAVA απαρτίζεται από μια ομάδα κλάσεων, οι οποίες ανήκουν στον ίδιο χώρο ονομάτων (namespace).
- Για τη δημιουργία μιας βιβλιοθήκης Java, πρέπει να χρησιμοποιήσουμε την λέξη-κλειδί **package** μαζί με το όνομα της βιβλιοθήκης, στην αρχή κάθε αρχείου, το οποίο θέλουμε να εντάξουμε στην βιβλιοθήκη.
- Έτσι, δηλώνοντας : **package mypackage**
  F στην αρχή κάποιων αρχείων Java, ορίζουμε ότι τα αρχεία αυτά ανήκουν στην ίδια βιβλιοθήκη, η οποία έχει το όνομα mypackage.
- Η δημιουργία μιας βιβλιοθήκης Java («πακέτου») δεν προϋποθέτει ούτε συνεπάγεται την τοποθέτηση αρχείων-κλάσεων σε κάποιο ενιαίο αρχείο.

Βιβλιοθήκες/Πακέτα (packages) (συν.)

Παράδειγμα: package test.animals;

```
package test.animals;

public class Tiger {

  public String name;

  public Tiger(){ }

  public Tiger(String name){
    this.name = name;
  }

  public String getName(){
    return name;
  }

}
```

```
package test.animals;

public class Lion {

  public String name;

  public Lion(){ }

  public Lion(String name){
    this.name = name;
  }

  public String getName(){
    return name;
  }

  public String isKing(){
    return "I am the king";  }
}
```

Βιβλιοθήκες/Πακέτα (packages) (συν.)

- Η Java μας δίνει τη δυνατότητα να επαναχρησιμοποιήσουμε τις κλάσεις μιας βιβλιοθήκης JAVA, με χρήση της εντολής **import** στον κώδικα των αρχείων στα οποία θέλουμε να κάνουμε χρήση των κλάσεων της βιβλιοθήκης.
- **Τρόπος 1 (χρήση import):**

```
import test.animals.Lion;
…
Lion l = new Lion();
```

- **Τρόπος 2 (χωρίς τη χρήση import):**

```
test.animals.Lion l = new
test.animals.Lion();
```

```
package test.animals;

public class Lion {

  public String name;

  public Lion(){ }

  public Lion(String name){
    this.name = name;
  }

  public String getName(){
    return name;
  }

  public String isKing(){
    return "I am the king";  }
}
```

Βιβλιοθήκες/Πακέτα (packages) (συν.)

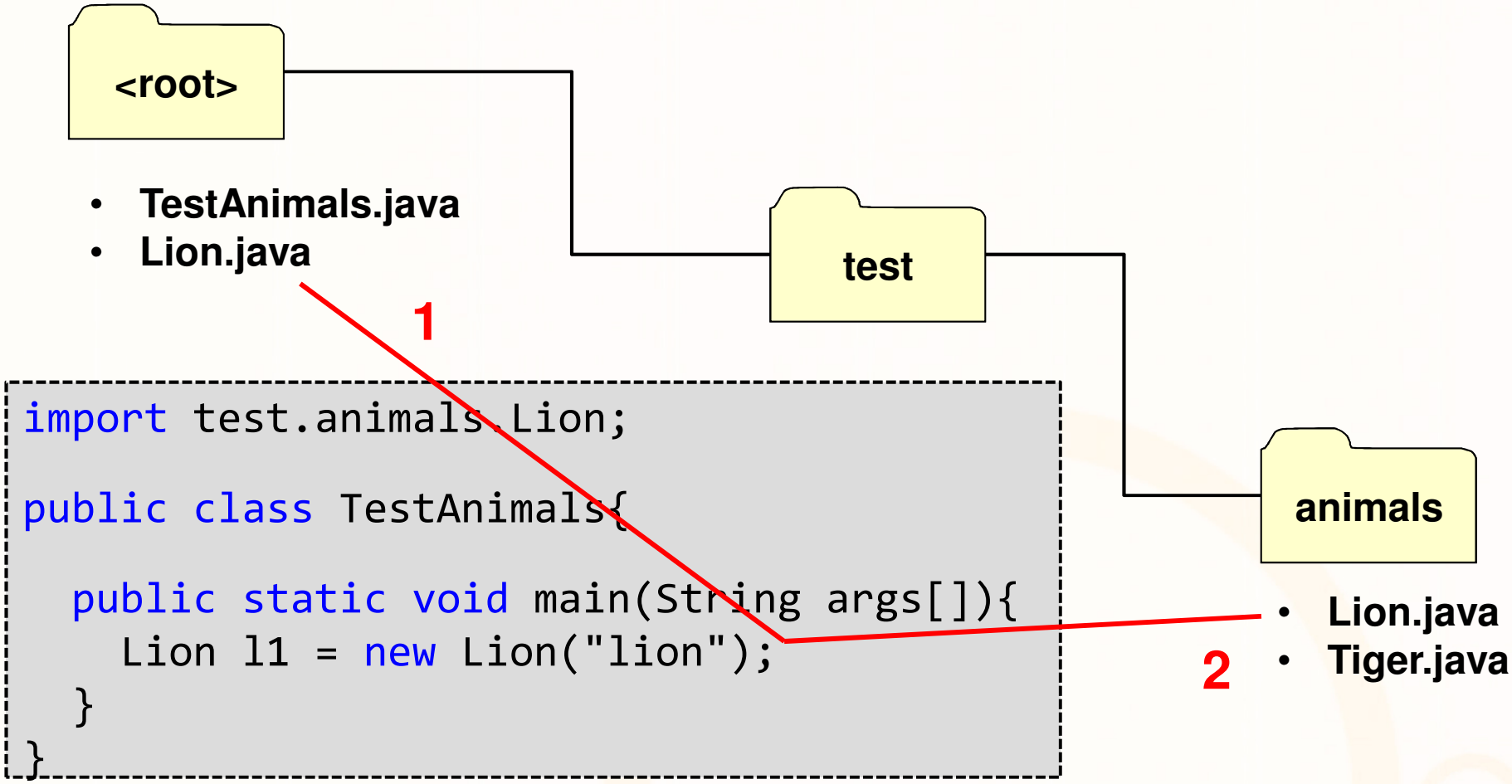
- Παράδειγμα:

```
import test.animals.Lion;
import test.animals.Tiger;
// OR import test.animals.*;

public class TestAnimals{

  public static void main(String args[]){
    Lion l1 = new Lion("lion");
    // OR test.animals.Lion l1 =
                    new test.animals.Lion("lion");

    Tiger t1 = new Tiger("tiger");
    // OR test.animals.Tiger t1 =
                    new test.animals.Tiger(“tiger");
  }
}
```

Παράδειγμα: Σύγκρουση Ονομάτων

Παράδειγμα: name clash

- **Ερώτηση:** Ποια κλάση Lion θα χρησιμοποιηθεί;



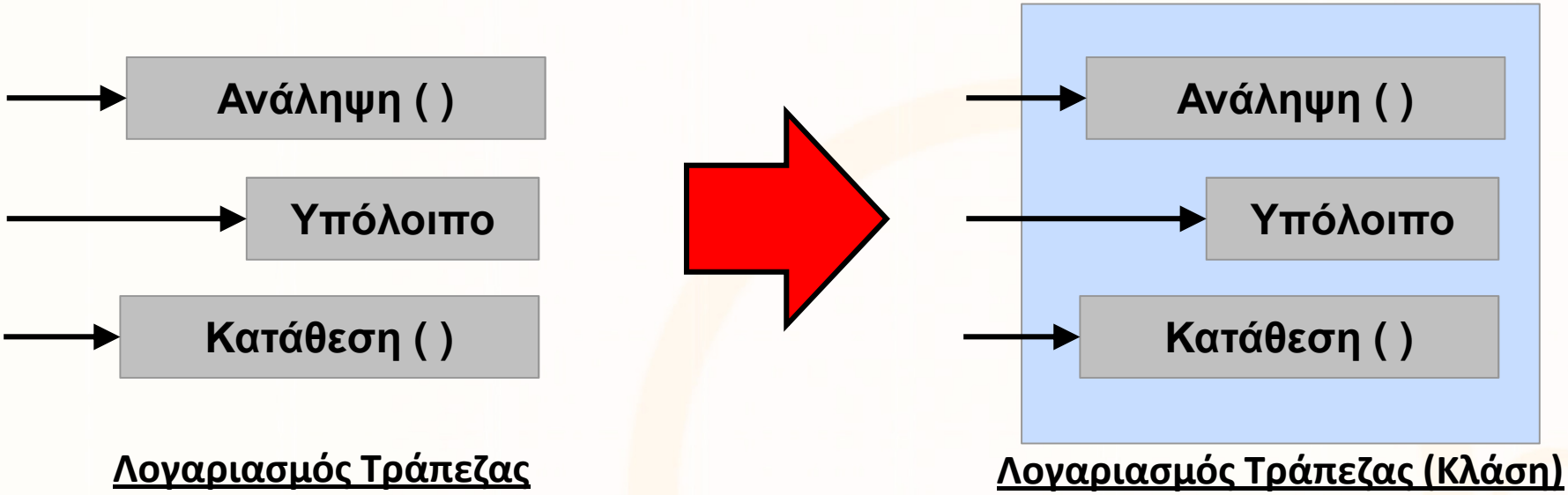
Ενθυλάκωση (encapsulation) (συν.)

Η Ενθυλάκωση (encapsulation) έχει δύο έννοιες:

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

   Η κλάση σαν ένας τύπος δεδομένων περιλαμβάνει:

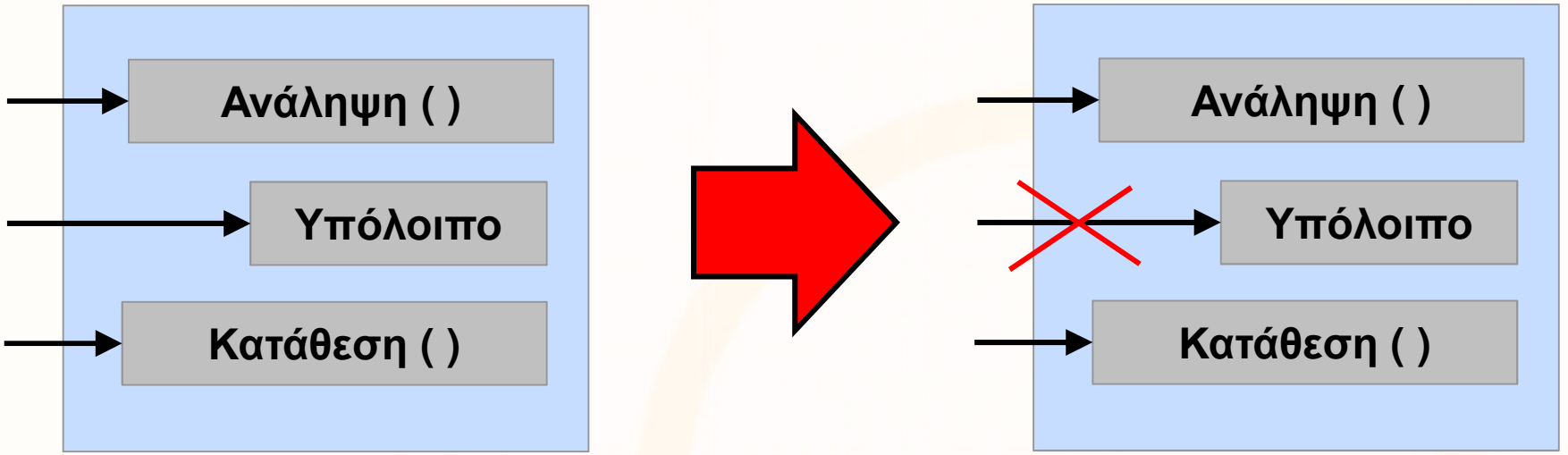
   - Μεταβλητές ή πεδία τα οποία αντιπροσωπεύουν διάφορες ιδιότητες
   - Μεθόδους οι οποίες χρησιμοποιούνται για να κατασκευάσουν στιγμιότυπα της κλάσης (constructors) και Μεθόδους οι οποίες χρησιμοποιούνται για να αναπαραστήσουν τις λειτουργίες της κλάσης (methods)



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



Τροποποιητές Πρόσβασης (Access Modifiers)

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

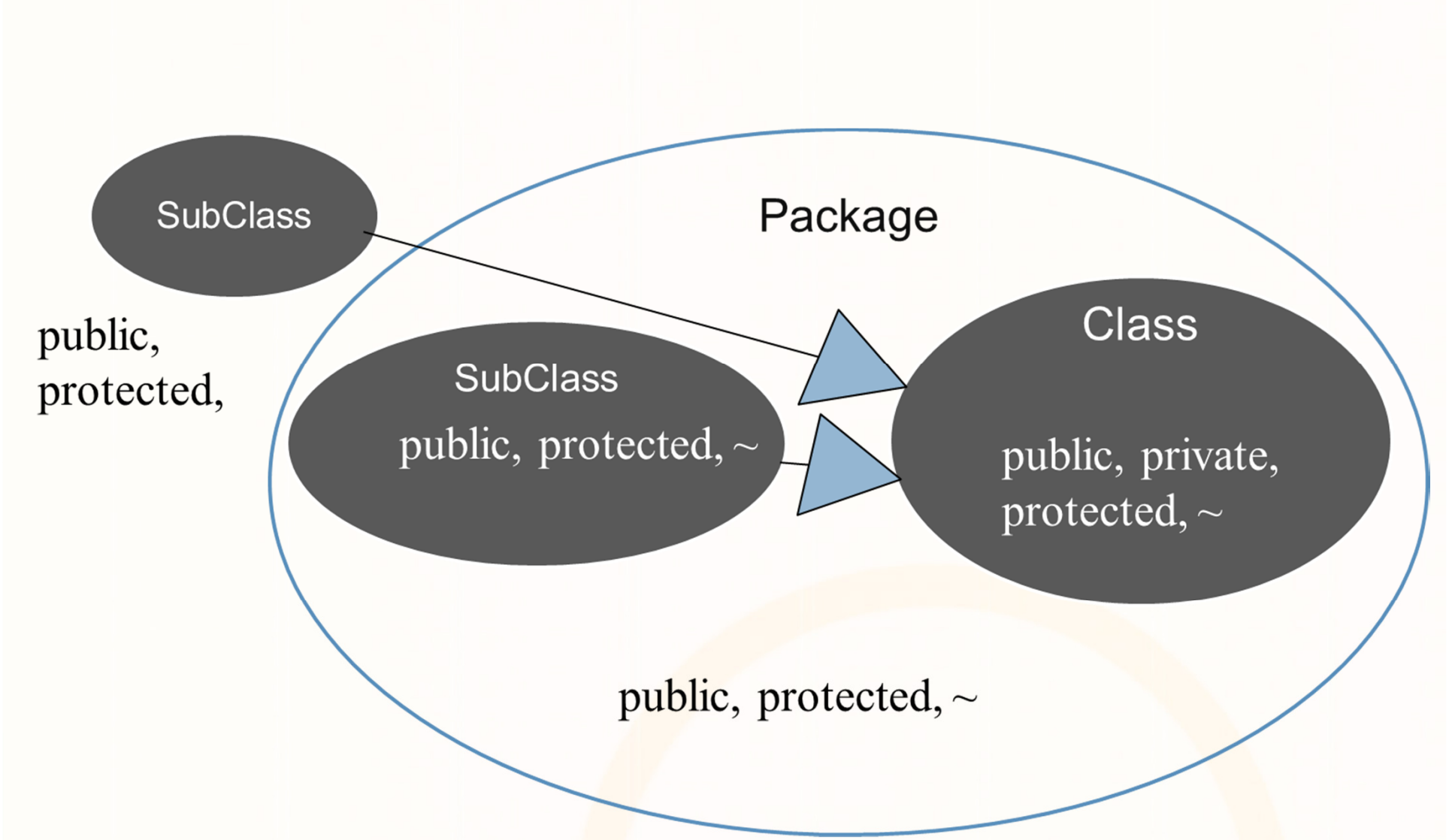
Υπάρχουν 4 τροποποιητές πρόσβασης:

- **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: Σύγκριση

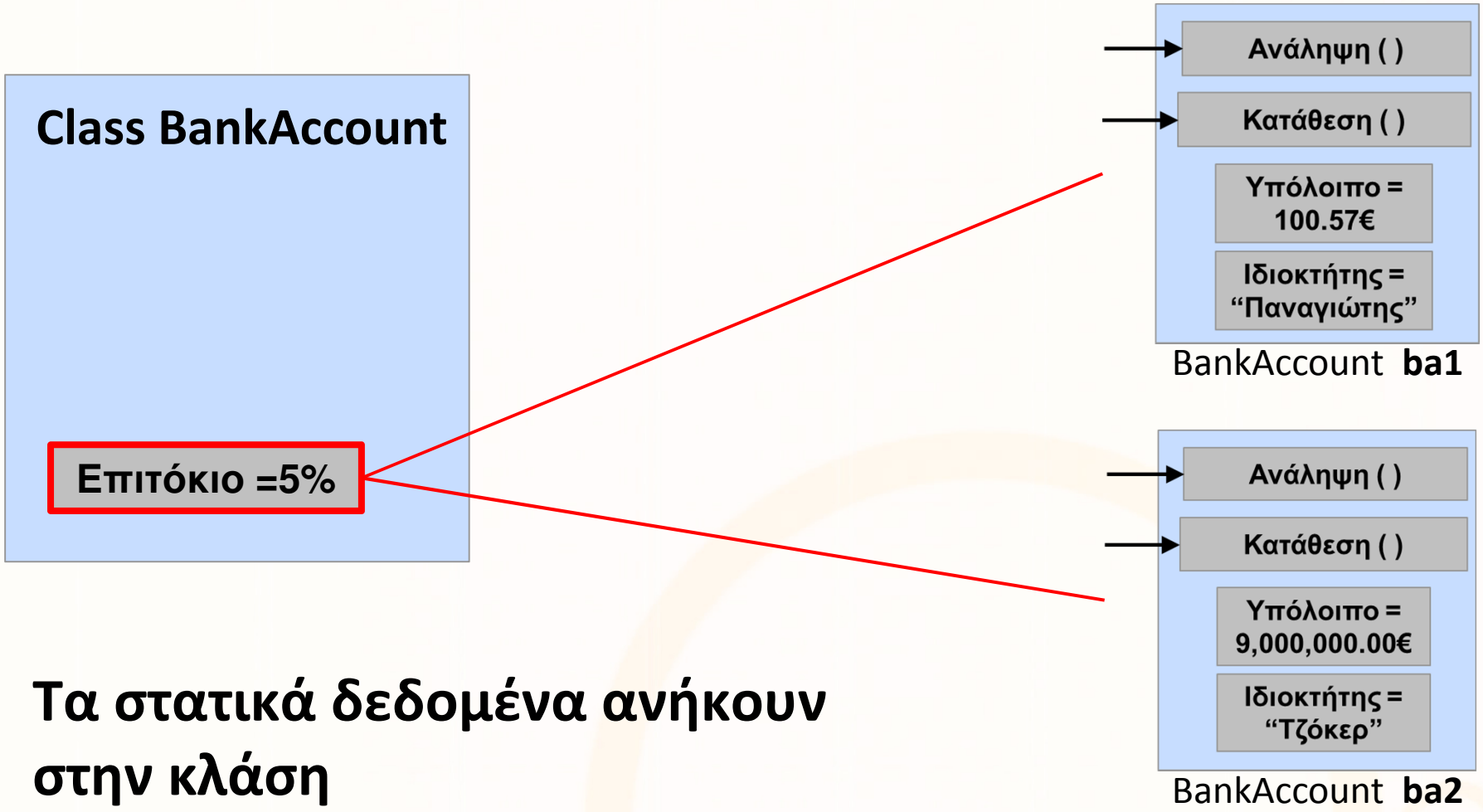
Τροποποιητής Πρόσβασης	Ίδια Κλάση	Ίδιο Πακέτο	Υποκλάση	Άλλα Πακέτα
public	√	√	√	√
protected	√	√	√	
(default) friendly	√	√		
private	√			

Τροποποιητές Πρόσβασης: Σύγκριση (συν.)



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: 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: Παράδειγμα 1

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

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

    // OK var2 is static
    var2 = 5;
  }

}
```

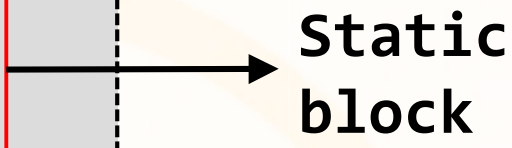
Παράδειγμα 2: Στατικές Μέθοδοι

```
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();
 }
}
```



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() {}
}
```

Παράδειγμα: Βιβλίο Διευθύνσεων

Αναφορά Απαιτήσεων

- **Βιβλίο Διευθύνσεων: το πρόγραμμα αυτό διατηρεί και συντηρεί ένα βιβλίο διευθύνσεων.**
- Το βιβλίο διευθύνσεων θα είναι μία συλλογή από οντότητες οι οποίες αναπαριστούν άτομα με πληροφορίες για όνομα, επίθετο, διεύθυνση, πόλη, επαρχία, ταχυδρομικό κώδικα και τηλέφωνο
- Το πρόγραμμα πρέπει να υποστηρίζει προσθήκη ενός ατόμου, ενημέρωση υφιστάμενου ατόμου (εκτός από το όνομα) και διαγραφή ατόμου
- <u>Επιπρόσθετες λειτουργίες:</u>
  - Ταξινόμηση των ατόμων αλφαβητικά με επίθετο, όνομα ή ταχ. κώδικα
  - Εκτύπωση όλων των ατόμων σε μορφή “mailing label”
  - Δημιουργία ενός καινούριου και άνοιγμα ενός υφιστάμενου βιβλίου διευθύνσεων από τον δίσκο με τη χρήση διαπροσωπείας με τις συνηθισμένες επιλογές κάτω από το File (New, Open, Close, Save, Save As)

Αναφορά Απαιτήσεων (συν.)

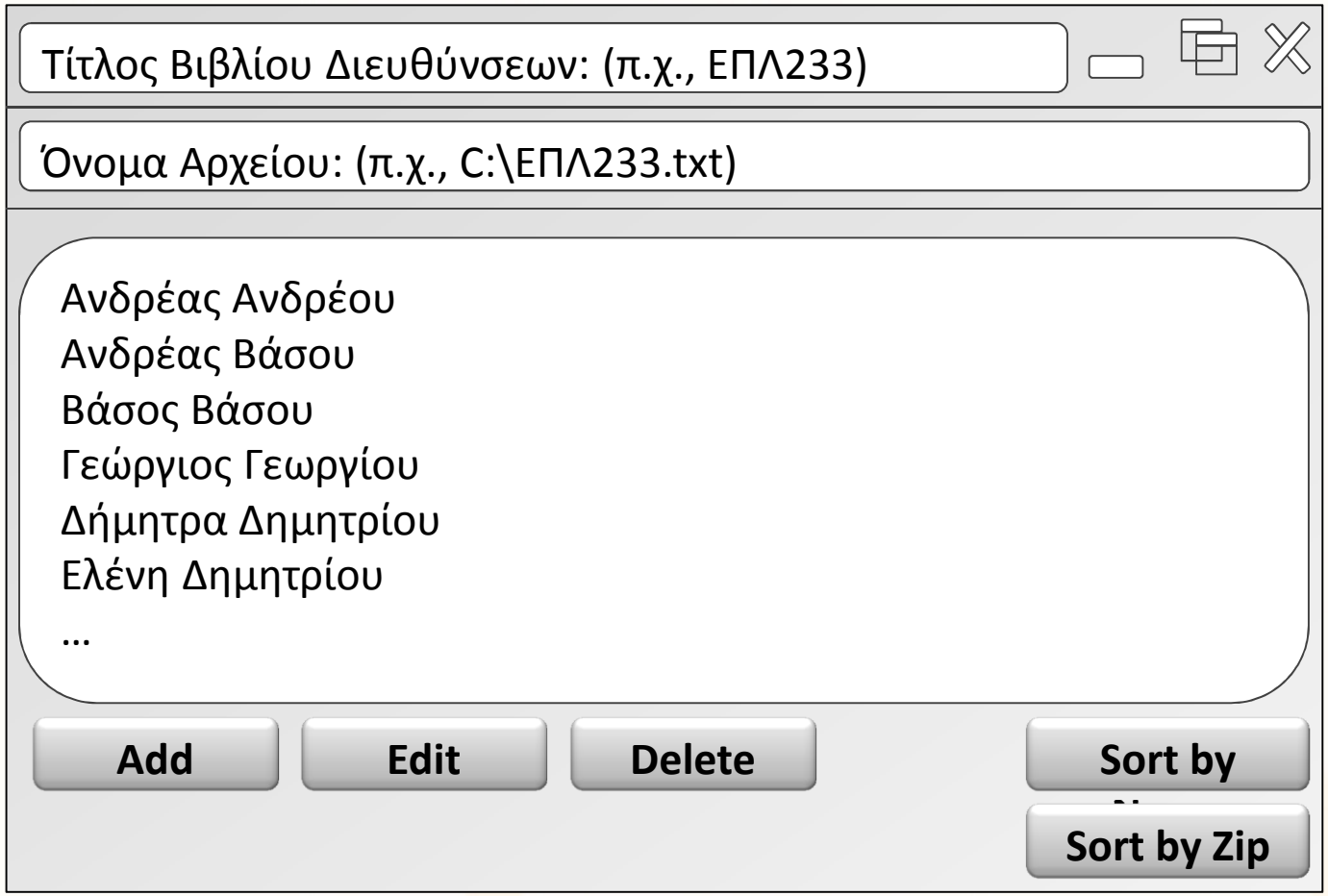
- Επιπρόσθετες λειτουργίες (συν.):
  - Το πρόγραμμα θα πρέπει να επεξεργάζεται, ανά πάσα στιγμή, μόνο ένα βιβλίο διευθύνσεων. Στην περίπτωση που ο χρήστης επιλέξει «Νέο Βιβλίο Διευθύνσεων» ή «Άνοιγμα Υφιστάμενου Βιβλίου Διευθύνσεων» τότε το πρόγραμμα θα σταματάει/ κλείνει το υφιστάμενο/τρέχον βιβλίο διευθύνσεων
  - Σε κάποια φάση στο μέλλον, το πρόγραμμα πιθανόν να μπορεί να υποστηρίζει να τρέχουν πολλά γραμματοκιβώτια ταυτόχρονα, το καθένα στο δικό του παράθυρο. Σε αυτή την περίπτωση, όταν ο χρήστης επιλέξει «Νέο Βιβλίο Διευθύνσεων» ή «Άνοιγμα Υφιστάμενου Βιβλίου Διευθύνσεων» τότε το πρόγραμμα δεν θα επηρεάζει τα τρέχοντα βιβλία διευθύνσεων
  - Το πρόγραμμα θα παρακολουθεί τις αλλαγές που γίνονται στο βιβλίο διευθύνσεων και θα παρέχει στο χρήστη την δυνατότητα να αποθηκεύσει τις αλλαγές στο υφιστάμενο ή σε νέο βιβλίο διευθύνσεων

Αναφορά Απαιτήσεων (συν.)

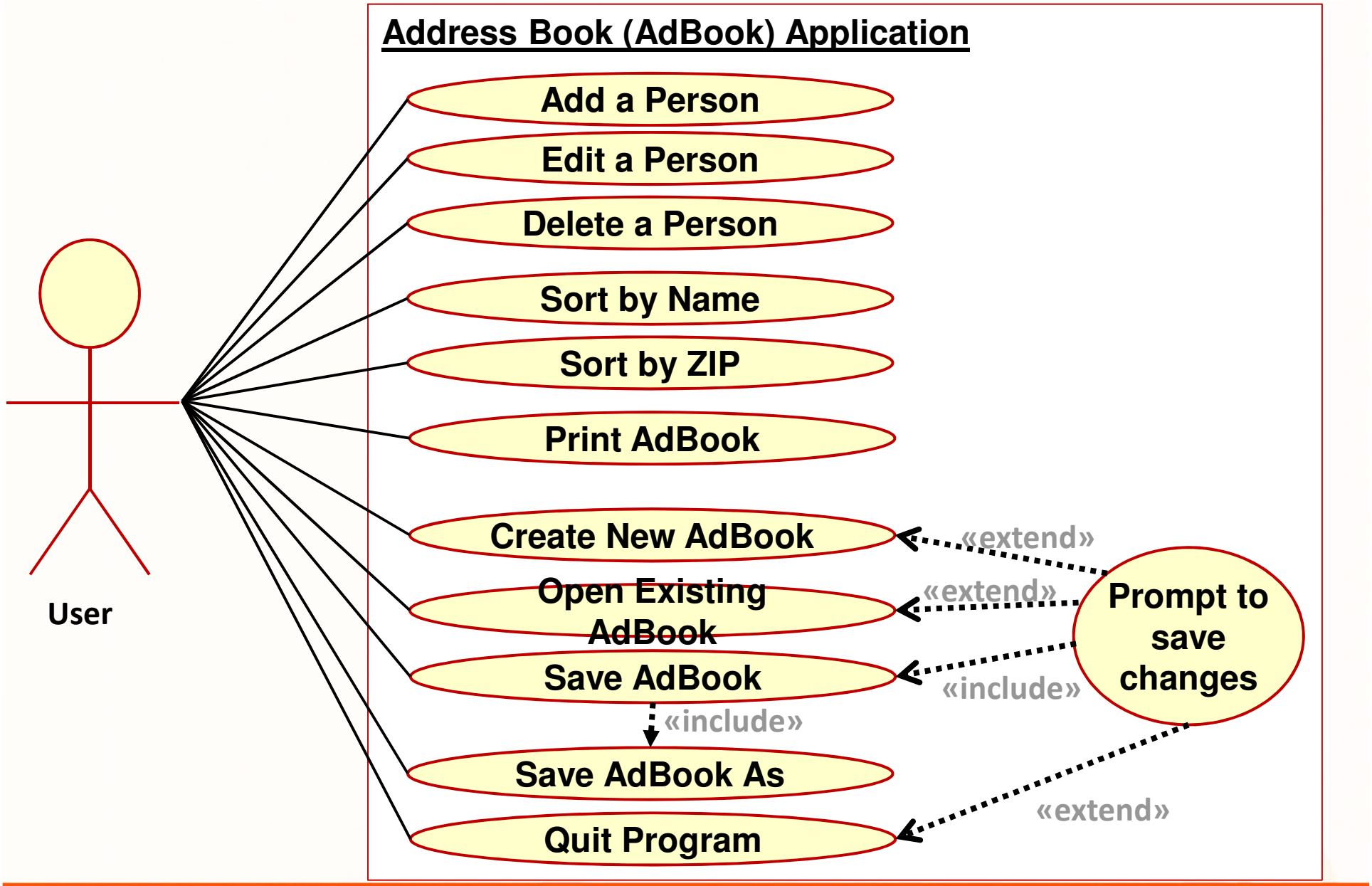
- Επιπρόσθετες λειτουργίες (συν.):
  - Το πρόγραμμα θα παρακολουθεί το αρχείο από το οποίο έχει διαβαστεί ή που έχει αποθηκευτεί πιο πρόσφατα, θα παρουσιάζει το όνομα του αρχείου στον τίτλο του παραθύρου και θα το χρησιμοποιεί όταν ο χρήστης επιλέξει το «Save»
  - Όταν ένα καινούριο παράθυρο δημιουργηθεί, τότε το παράθυρο θα παρουσιάζει τον τίτλο «Untitled» και η επιλογή «Save» μετατρέπεται σε «Save As» και ο χρήστης θα πρέπει να δώσει ένα όνομα αρχείου

Αναφορά Απαιτήσεων (συν.)

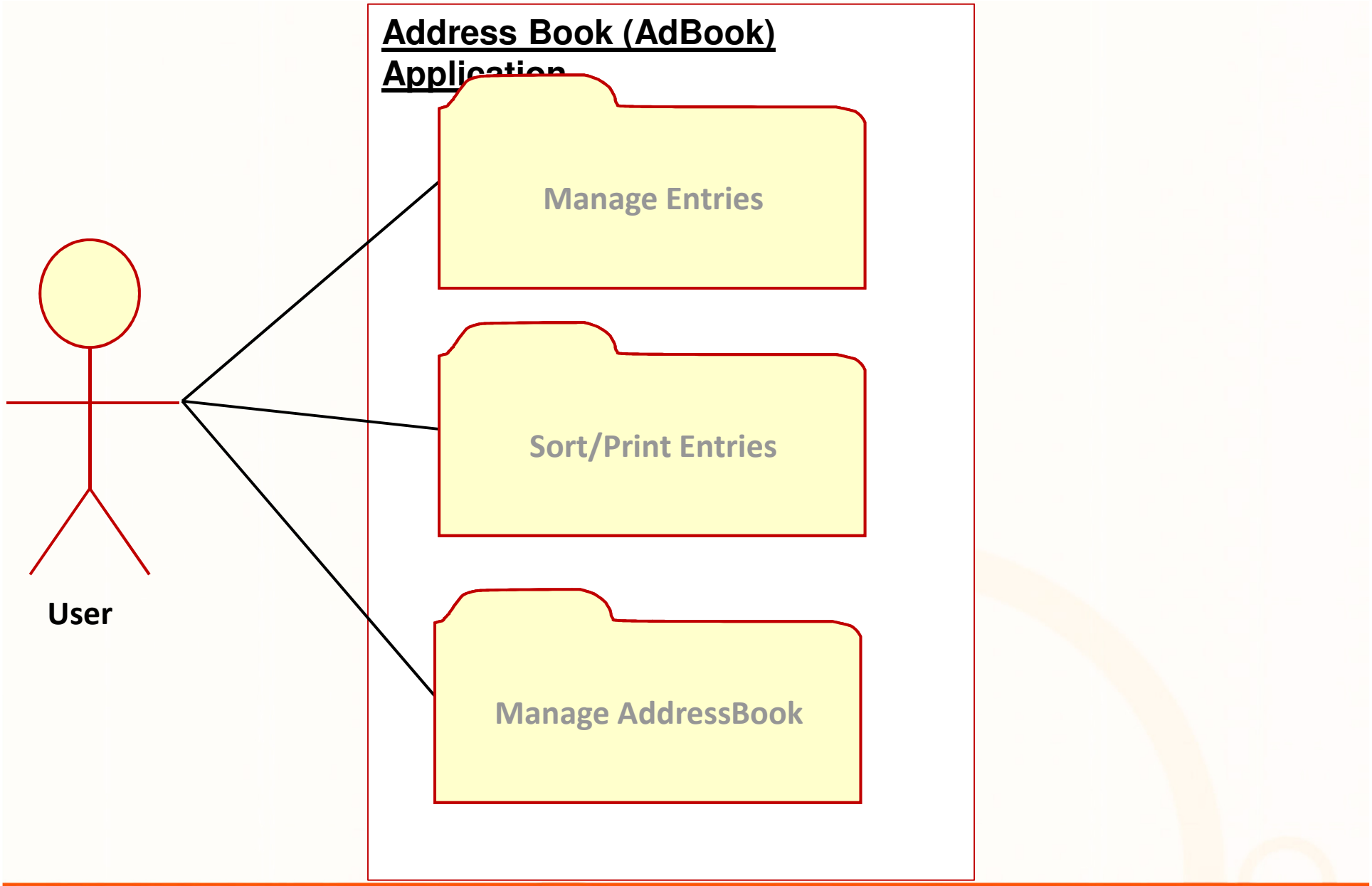
- Ανάλυση Διαπροσωπείας:



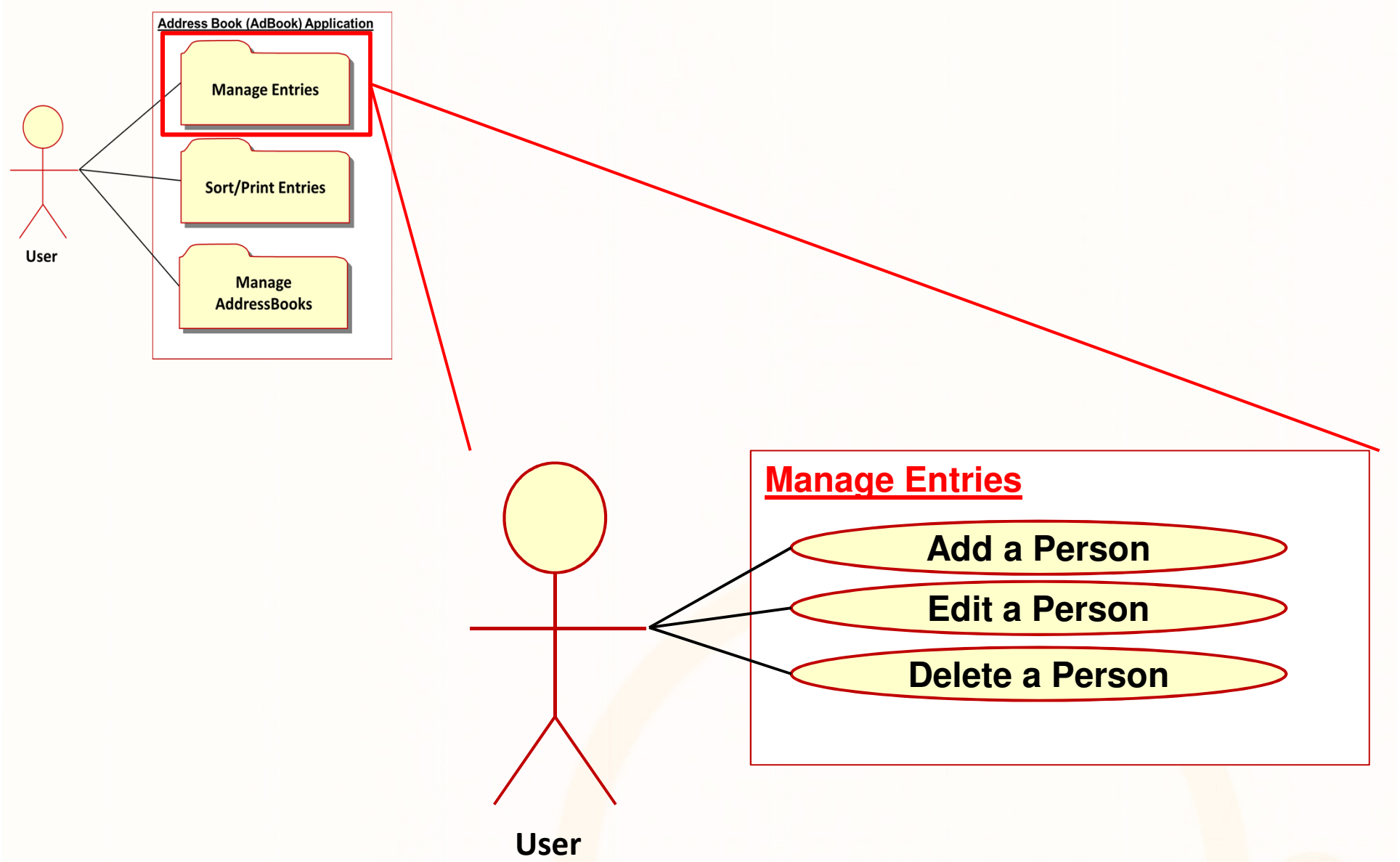
Βιβλίο Διευθύνσεων: Use Case Diagram



Βιβλίο Διευθύνσεων: Use Case Package Diagram



Βιβλίο Διευθύνσεων: Use Case Diagram (packets)



Βιβλίο Διευθύνσεων: Use Cases

- **Add Person**

  Το εν λόγω use case ξεκινά όταν ο χρήστης κάνει κλικ στο κουμπί "Προσθήκη" στο κύριο παράθυρο. Εμφανίζεται ένα παράθυρο διαλόγου, με τίτλο "Νέο Άτομο", το οποίο περιέχει πεδία στα οποία πρέπει να συμπληρωθεί το όνομα και το επώνυμο του νέου ατόμου και άλλες πληροφορίες. Το παράθυρο διαλόγου μπορεί να κλείσει είτε κάνοντας κλικ στο κουμπί "ΟΚ" ή "Άκυρο". Εάν πατηθεί το κουμπί "ΟΚ", ένα νέο άτομο προστίθεται στο τέλος του βιβλίου διευθύνσεων, και το όνομα του ατόμου, προστίθεται στο τέλος της λίστας των ονομάτων στο κύριο παράθυρο. Εάν πατηθεί το κουμπί "Άκυρο", δεν γίνονται αλλαγές ούτε στο βιβλίο διευθύνσεων, ούτε στο κύριο παράθυρο.

Βιβλίο Διευθύνσεων: Use Cases

- **Edit Person**

  Το εν λόγω use case ξεκινά όταν ο χρήστης επιλέξει είτε ένα όνομα από τη λίστα των ονομάτων στο κύριο παράθυρο και, στη συνέχεια κάνει κλικ στο κουμπί "Επεξεργασία", είτε κάνει διπλό κλικ σε ένα όνομα. Σε κάθε περίπτωση, ένα παράθυρο διαλόγου, με τίτλο "Επεξεργασία ονόματος του ατόμου", εμφανίζεται περιέχοντας πληροφορίες σχετικές με το επιλεγμένο πρόσωπο (εκτός από το όνομα του ατόμου, το οποίο εμφανίζεται μόνο στον τίτλο). Ο χρήστης μπορεί στη συνέχεια να επεξεργαστεί τα πεδία με τα χαρακτηριστικά του ατόμου. Το παράθυρο διαλόγου μπορεί να κλείσει είτε κάνοντας κλικ στο "ΟΚ" ή "Άκυρο". Εάν πατηθεί το κουμπί "ΟΚ", η καταχώρηση στο βιβλίο διευθύνσεων για το επιλεγμένο πρόσωπο ενημερώνεται με τις αλλαγές που έγιναν από το χρήστη. Εάν πατηθεί το κουμπί "Άκυρο", δεν γίνονται αλλαγές στο βιβλίο διευθύνσεων.

Βιβλίο Διευθύνσεων: Use Cases

- **Sort by Name**

  Το εν λόγω use case ξεκινά όταν ο χρήστης κάνει κλικ στο κουμπί “Ταξινόμηση κατά Όνομα” στο κύριο παράθυρο. Οι καταχωρήσεις στο βιβλίο διευθύνσεων ταξινομούνται αλφαβητικά κατά όνομα, και ο κατάλογος στο κύριο παράθυρο ενημερώνεται ώστε να αντικατοπτρίζει τη ταξινόμηση.

Βιβλίο Διευθύνσεων: Use Cases

- **Prompt to Save Changes Extension**

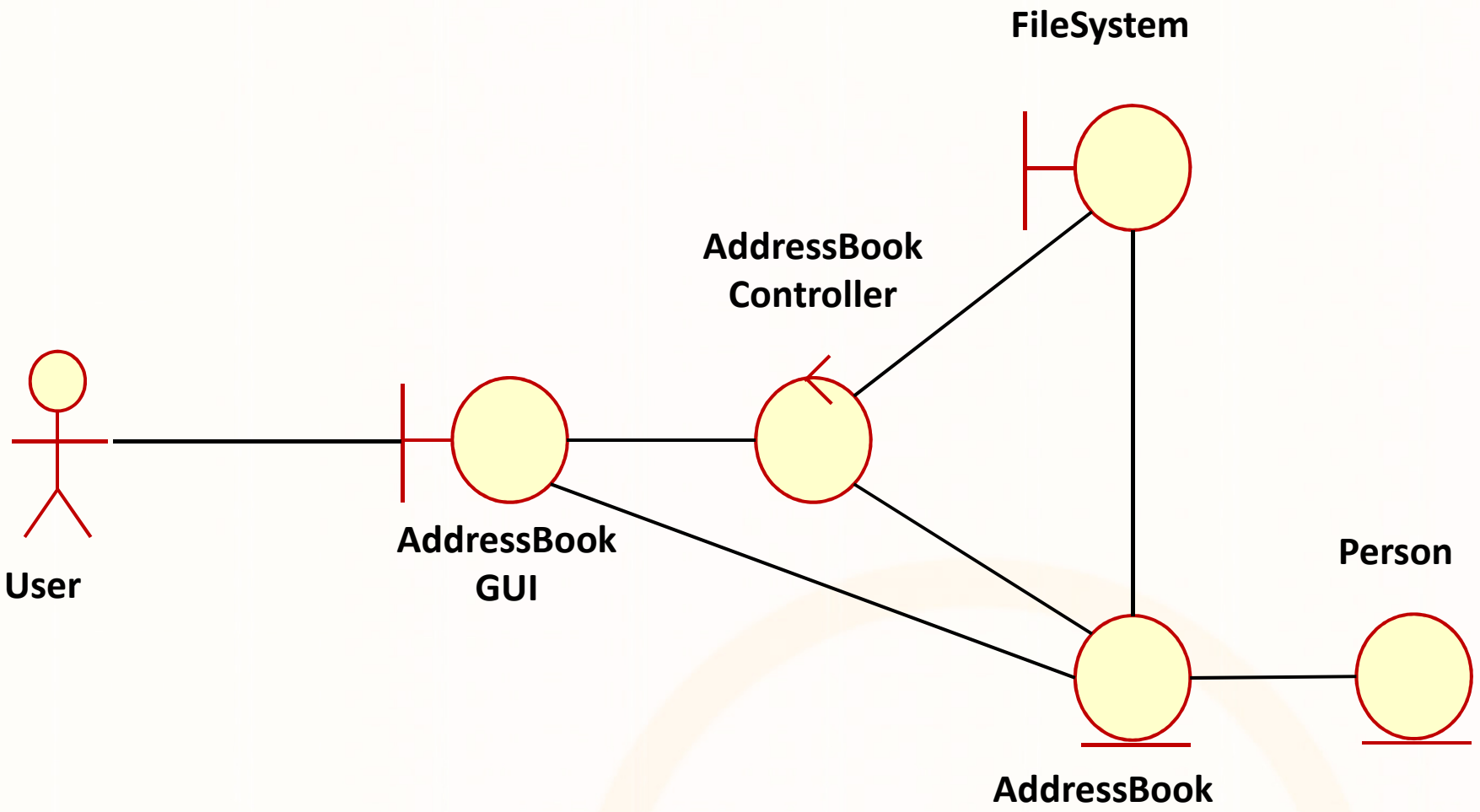
  Το εν λόγω use case ξεκινά μέσα από τη Δημιουργία νέου βιβλίου διευθύνσεων, Άνοιγμα υπάρχον βιβλίο διευθύνσεων, ή Έξοδος προγράμματος, εφόσον το τρέχον βιβλίο διευθύνσεων έχει αλλάξει από τη τελευταία επιτυχημένη λειτουργία New, Open, Save, ή Save As. Ένα παράθυρο διαλόγου εμφανίζεται, ενημερώνοντας τον χρήστη ότι υπάρχουν μη αποθηκευμένες αλλαγές, και ζητώντας από το χρήστη είτε να αποθηκεύσει τις αλλαγές, να μην αποθηκεύσει τις αλλαγές, ή να ακυρώσει τη λειτουργία. Αν ο χρήστης επιλέξει να αποθηκεύσει τις αλλαγές, το use case Αποθήκευση Βιβλίου Διευθύνσεων εκτελείται (το οποίο μπορεί να οδηγήσει στην εκτέλεση του use case Αποθήκευση Βιβλίου Διευθύνσεων ως, σε περίπτωση που δεν υπάρχει τρέχον αρχείο). Εάν ο χρήστης δεν επιλέξει να αποθηκεύσει τις αλλαγές, η αρχική λειτουργία απλά επαναλαμβάνεται. Εάν ο χρήστης επιλέξει να ακυρώσει (ή ακυρώσει το διάλογο Αποθήκευση αρχείου), η αρχική λειτουργία ακυρώνεται.

Βιβλίο Διευθύνσεων: Class Analysis

- Το πρόγραμμα, **ανά πάσα στιγμή** διαχειρίζεται **ένα βιβλίο διευθύνσεων (AddressBook)**. → **οντότητα**
- Κάθε άτομο **(Person)** μπορεί να αναπαρασταθεί σαν μία **οντότητα** με τα χαρακτηριστικά της → **οντότητα**
- Μία **διαπροσωπεία (AddressBookGUI)** θα υπάρχει **μεταξύ χρήστη και συστήματος**. Ο χρήστης δεν θα μπορεί να έχει πρόσβαση διαφορετικά με το σύστημα → **σύνορο**
- Μία **διαπροσωπεία/αντικείμενο (FileSystem)** θα υπάρχει **μεταξύ του συστήματος και του λειτουργικού** ώστε το σύστημα να μπορεί να διαχειρίζεται αρχεία → **σύνορο**
- Ένα αντικείμενο διαχειριστής **(AddressBookController)** χρειάζεται για να αναλάβει την **εκτέλεση των use cases** σε ανταπόκριση των διαφόρων λειτουργιών που θα επιλέξει ο χρήστης από το μενού. → **controller**

  *(Για ένα τέτοιο μικρό πρόβλημα, ένας controller είναι αρκετός.)

Βιβλίο Διευθύνσεων: Analysis Class Diagram



Βιβλίο Διευθύνσεων: Class Analysis (συν.)

- Το use case “Add a Person” έχει σχέση με την εισαγωγή νέας πληροφορίας (δηλ. νέο άτομο) από τον χρήστη. Τέλος, θα καλεστεί το αντικείμενο AddressBook ώστε να εισάξει το νέο άτομο στη συλλογή.
- Το use case “Edit a Person” έχει σχέση με την παρουσίαση της τρέχουσας πληροφορίες κάποιου ατόμου μέσω του αντικειμένου AddressBook και παροχή της ικανότητας αλλαγή των στοιχείων αυτού. Στο τέλος, θα καλεστεί το αντικείμενο AddressBook για να αποθηκεύει τις αλλαγές.
- Το use case “Delete a Person” έχει σχέση με την διαγραφή ενός υπάρχον ατόμου. παρουσίαση της τρέχουσας. Στο τέλος, θα καλεστεί το αντικείμενο AddressBook για να αφαιρέσει το άτομο από την συλλογή.
- Το use case “Sort by Name” θα καλέσει το αντικείμενο AddressBook για να ταξινομήσει τη συλλογή των ατόμων με το όνομα και επίθετο τους.

Βιβλίο Διευθύνσεων: Class Analysis (συν.)

- Το use case “Create New Address Book” δημιουργεί ένα καινούριο αντικείμενο τύπου AddressBook.
- Το use case “Open Existing Address Book” θα ζητήσει από τον χρήστη ένα αρχείο και μετά θα χρησιμοποιήσει το αντικείμενο FileSystem για να διαβάσει ένα αντικείμενο τύπου AddressBook που έχει αποθηκευτεί στο αρχείο.
- Το use case “Save Address Book” θα χρησιμοποιήσει το αντικείμενο FileSystem για να αποθηκεύσει ένα αντικείμενο τύπου AddressBook στο πιο πρόσφατο αρχείο που έχει διαβαστεί ή αποθηκευτεί). Αν δεν υπάρχει κάποιο αρχείο τότε θα εκτελέσει το use case “Save Address Book As”
- Το use case “Print Address Book” θα ζητήσει από το αντικείμενο AddressBook να τυπώσει την συλλογή της σύμφωνα με την τρέχουσα ταξινόμηση.
- Το use case “Quit Program” δεν έχει σχέση με τα υπόλοιπα αντικείμενα.

Βιβλίο Διευθύνσεων: Κάρτες CRC

FileSystem	
Responsibilities - Διάβασε ένα αποθηκευμένο βιβλίο διευθύνσεων από το αρχείο - Αποθήκευσε ένα βιβλίο διευθύνσεων σε αρχείο - ...	**Collaborators** - AddressBook - AddressBook

Βιβλίο Διευθύνσεων: Κάρτες CRC

Person	
Responsibilities - Δημιουργία καινούριου ατόμου με τα στοιχεία του - Επιστροφή στοιχείων ατόμου - Ενημέρωση στοιχείων ατόμου (εκτός ονόματος) – …	**Collaborators**

Βιβλίο Διευθύνσεων: Κάρτες CRC

AddressBookGUI	
Responsibilities - Παρακολούθηση του τρέχον AddressBook - Παρουσίαση λίστα με ονόματα ατόμων - Παρουσίαση του τίτλου του AddressBook - …	**Collaborators** - AddressBook - AddressBook - AddressBook

Βιβλίο Διευθύνσεων: Κάρτες CRC

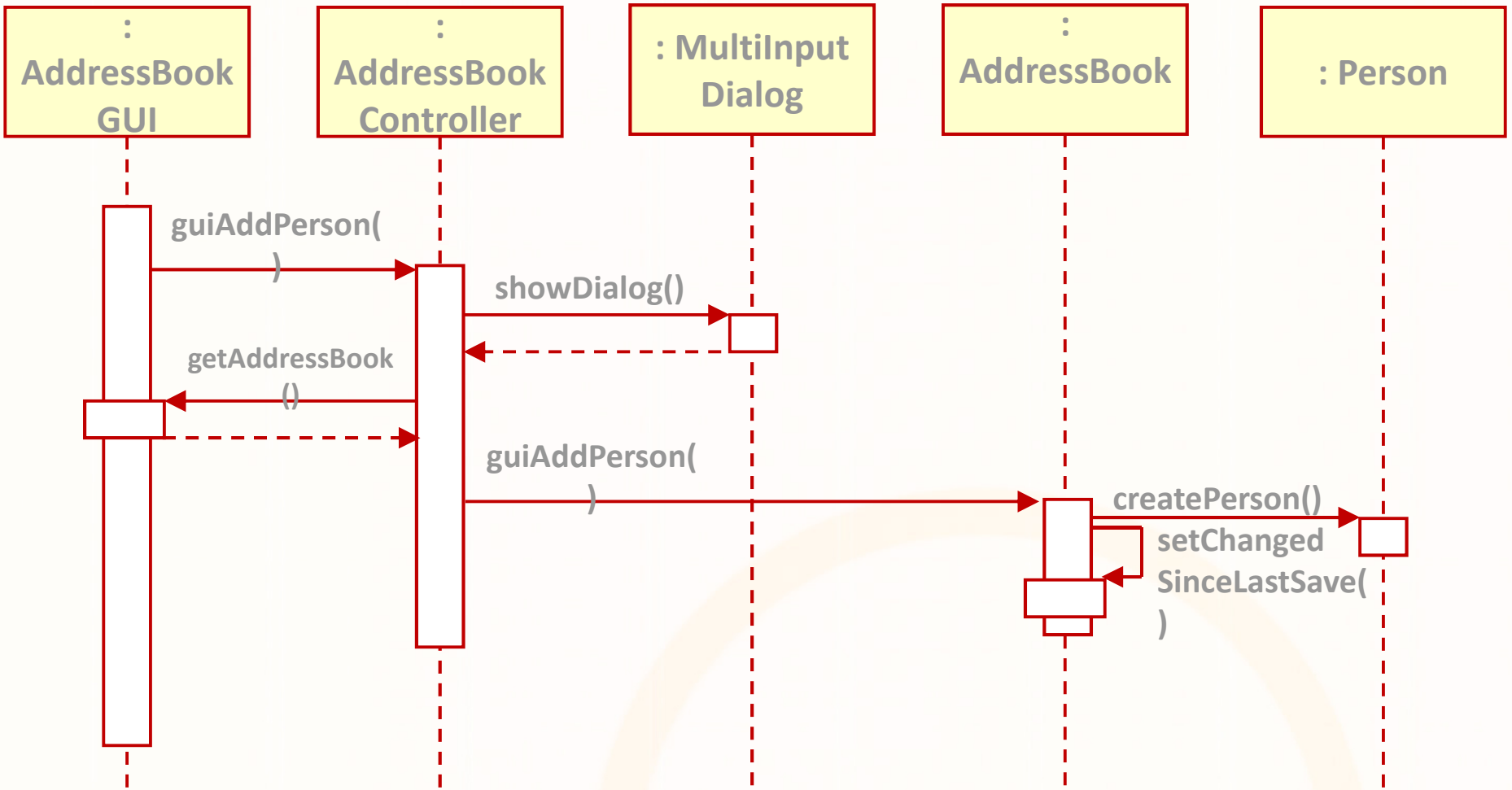
AddressBookControll	
Responsibilities Επιτρέπει την εκτέλεση των use case: - Add/Edit/Delete Person - Sort Entries (Name/Zip) - Create new AddressBook - Open Existing AddressB - Save AddressBook - …	**Collaborators** - AddressBook - AddressBook - AddressBook - FileSystem - FileSystem

Βιβλίο Διευθύνσεων: Κάρτες CRC

AddressBook	
Responsibilities - Άσκηση για το σπίτι	**Collaborators** - Άσκηση για το σπίτι

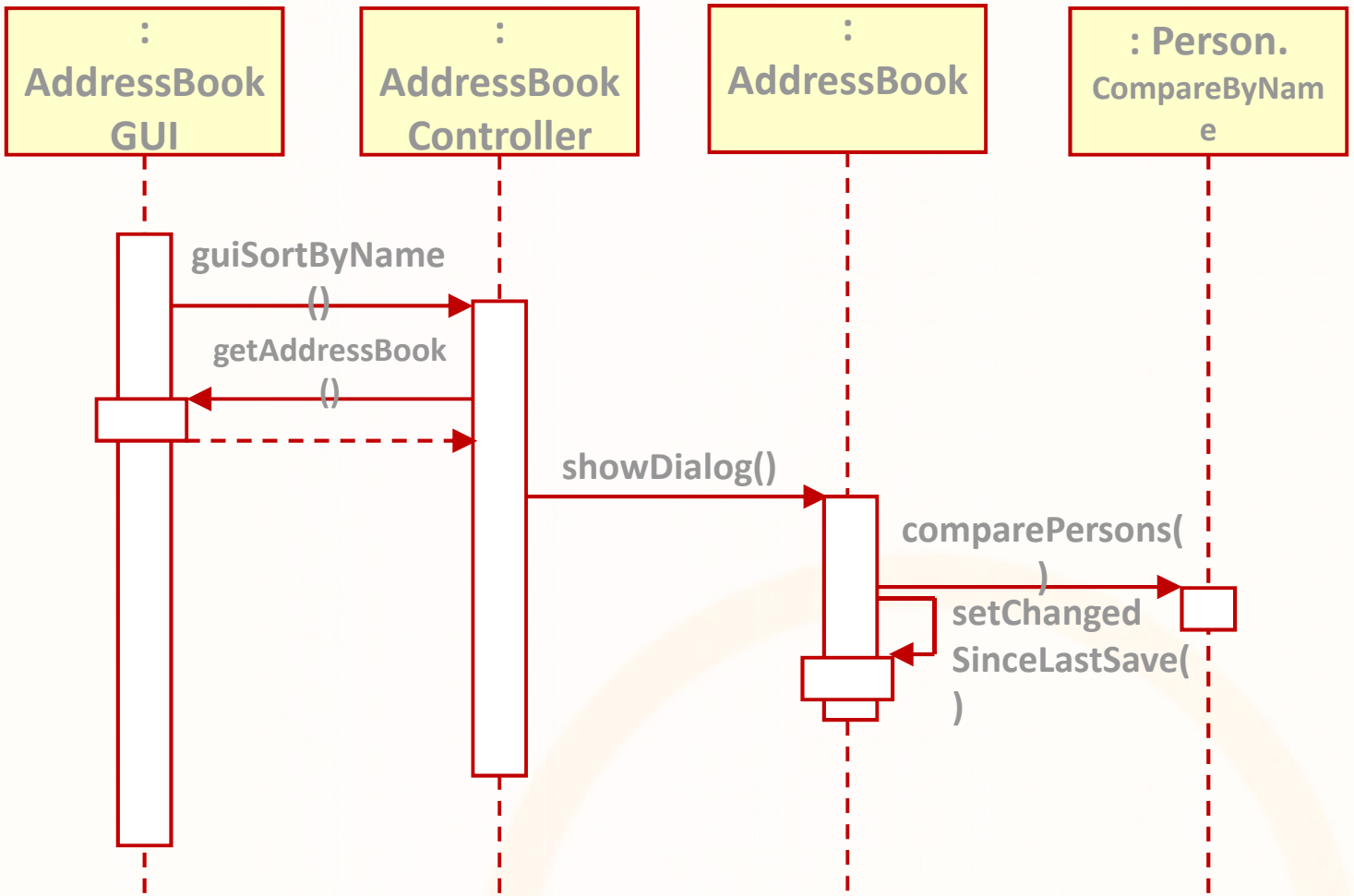
Βιβλίο Διευθύνσεων: Sequence Diagrams

Use case: Add Person

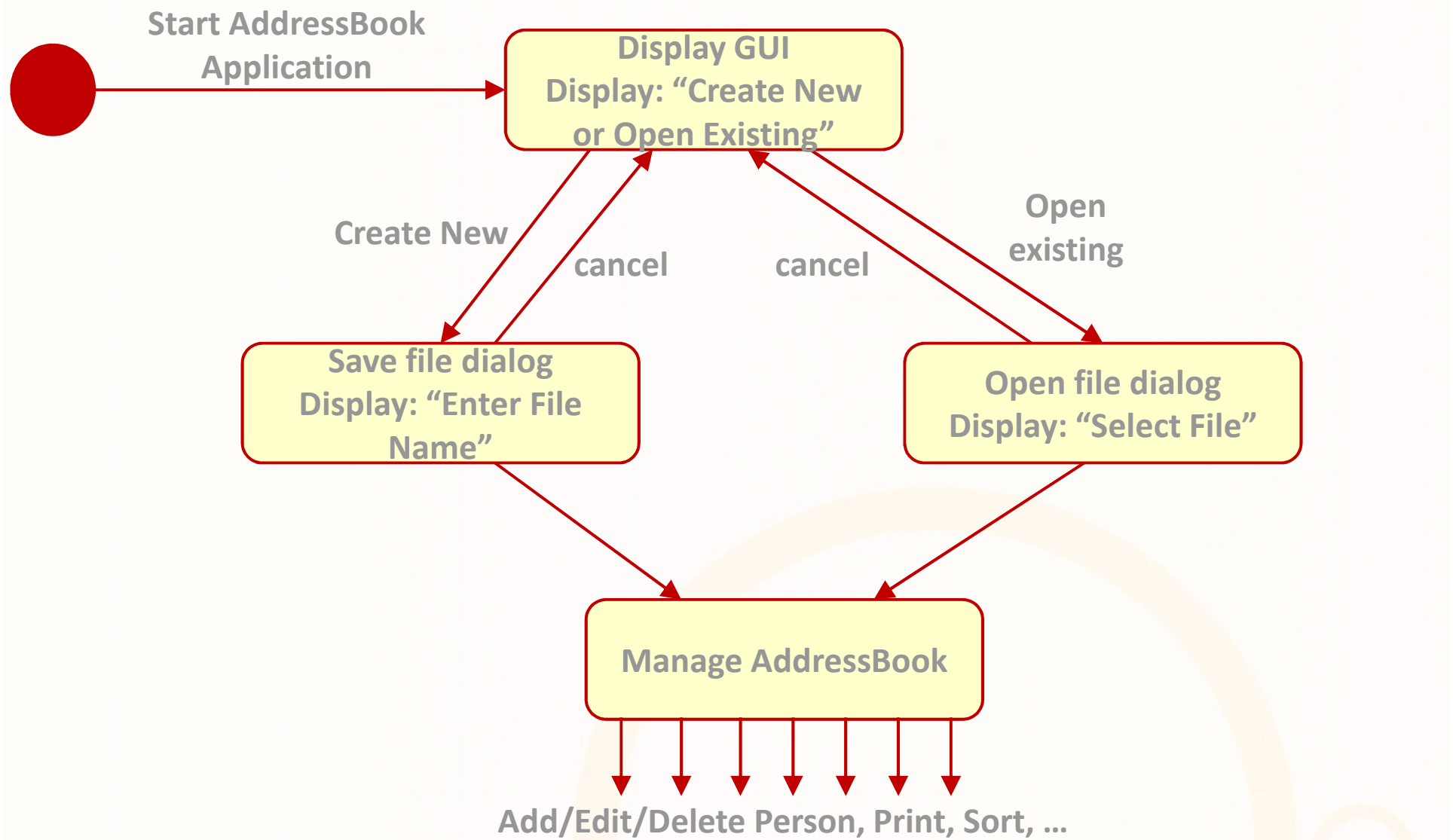


Βιβλίο Διευθύνσεων: Sequence Diagrams

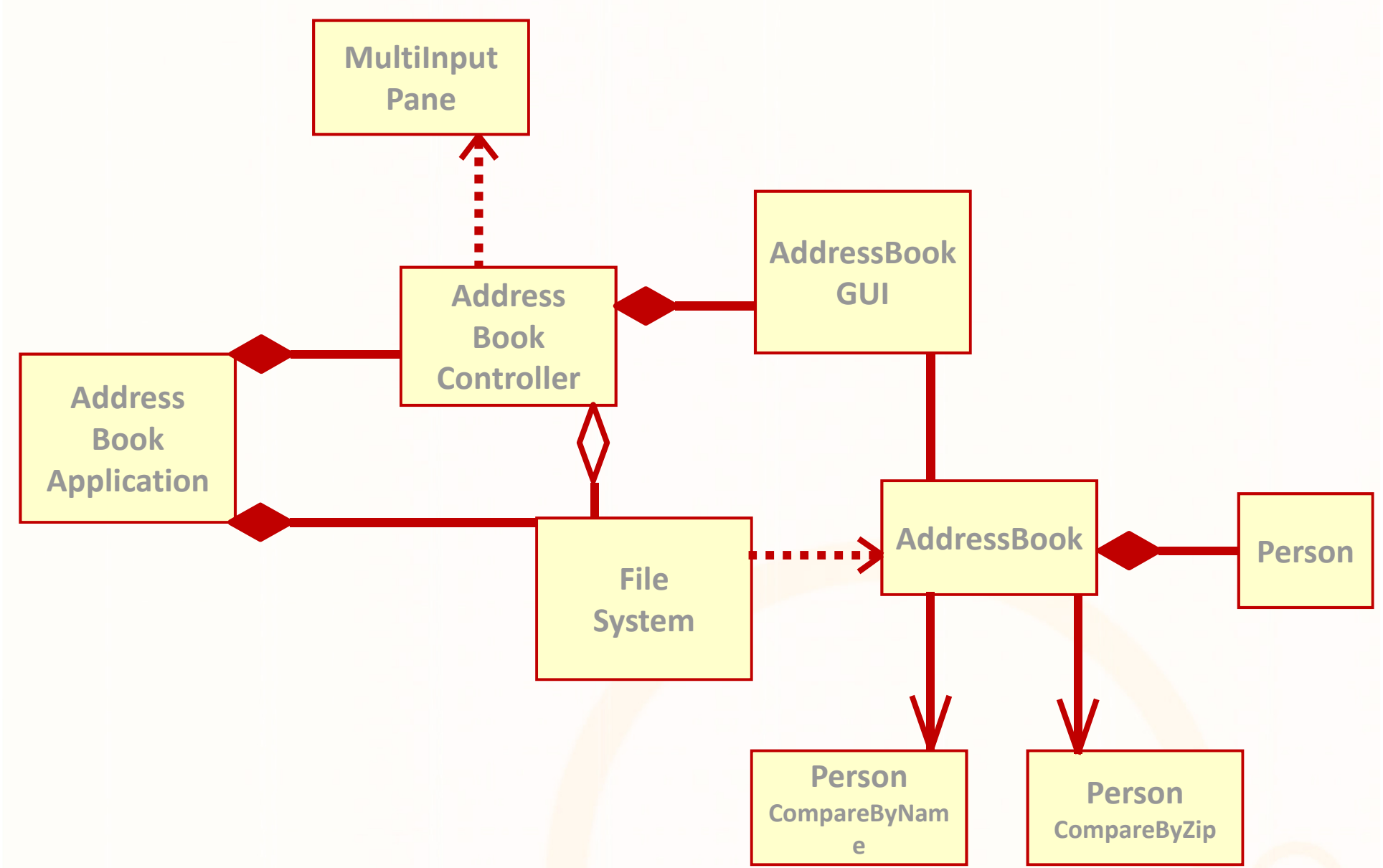
Use case: Sort by Name



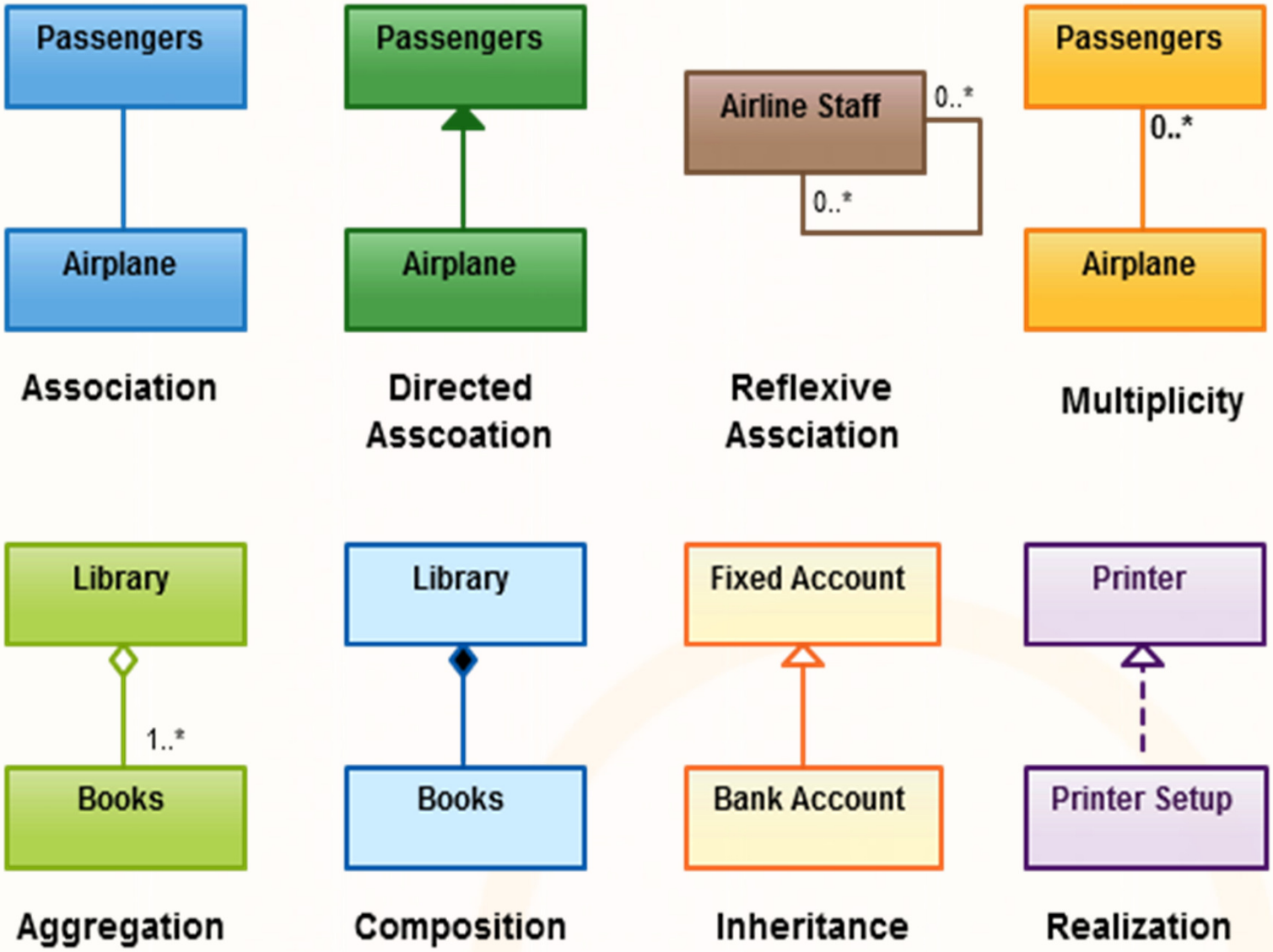
Βιβλίο Διευθύνσεων: state diagram



Βιβλίο Διευθύνσεων: Class Diagram



Παραδείγματα: Σχέσεις



Βιβλίο Διευθύνσεων: Class Diagrams (αναλυτικά)

Person
- firstName: String - lastName: String - address: String - city: String - state: String - zip: String - phone: String
+ Person(String firstName, String lastName, String address, String city, String state, String zip, String phone) + getFirstName(): String + getLastName(): String + getAddress(): String + getCity(): String + getState(): String + getZip(): String + getPhone(): String

Βιβλίο Διευθύνσεων: Class Diagrams (αναλυτικά)

FileSystem
+ readFile(File file): AddressBook + saveFile(AddressBook addressBook, File file)

Βιβλίο Διευθύνσεων: Class Diagrams (αναλυτικά)

AddressBookApplication
- fileSystem: FileSystem - controller: AddressBookController
+ main() + quitApplication()

Βιβλίο Διευθύνσεων: Class Diagrams (αναλυτικά)

AddressBook
- collection: Person [ ] or Vector or ArrayList - count: int - file: File - changedSinceLastSave: boolean
+ AddressBook() + getNumberOfPersons(): int + addPerson(String firstName, String lastName, …) + getFullNameOfPerson(int index): String + updatePerson(int index, String address, String city, String state, …) + removePerson(int index) + sortByName() + sortByZip() + printAll() + sortByName() + getFile(): File + getTitle(): String …

Βιβλίο Διευθύνσεων: Class Diagrams (αναλυτικά)

AddressBookGUI
Εργασία για το σπίτι
Εργασία για το σπίτι

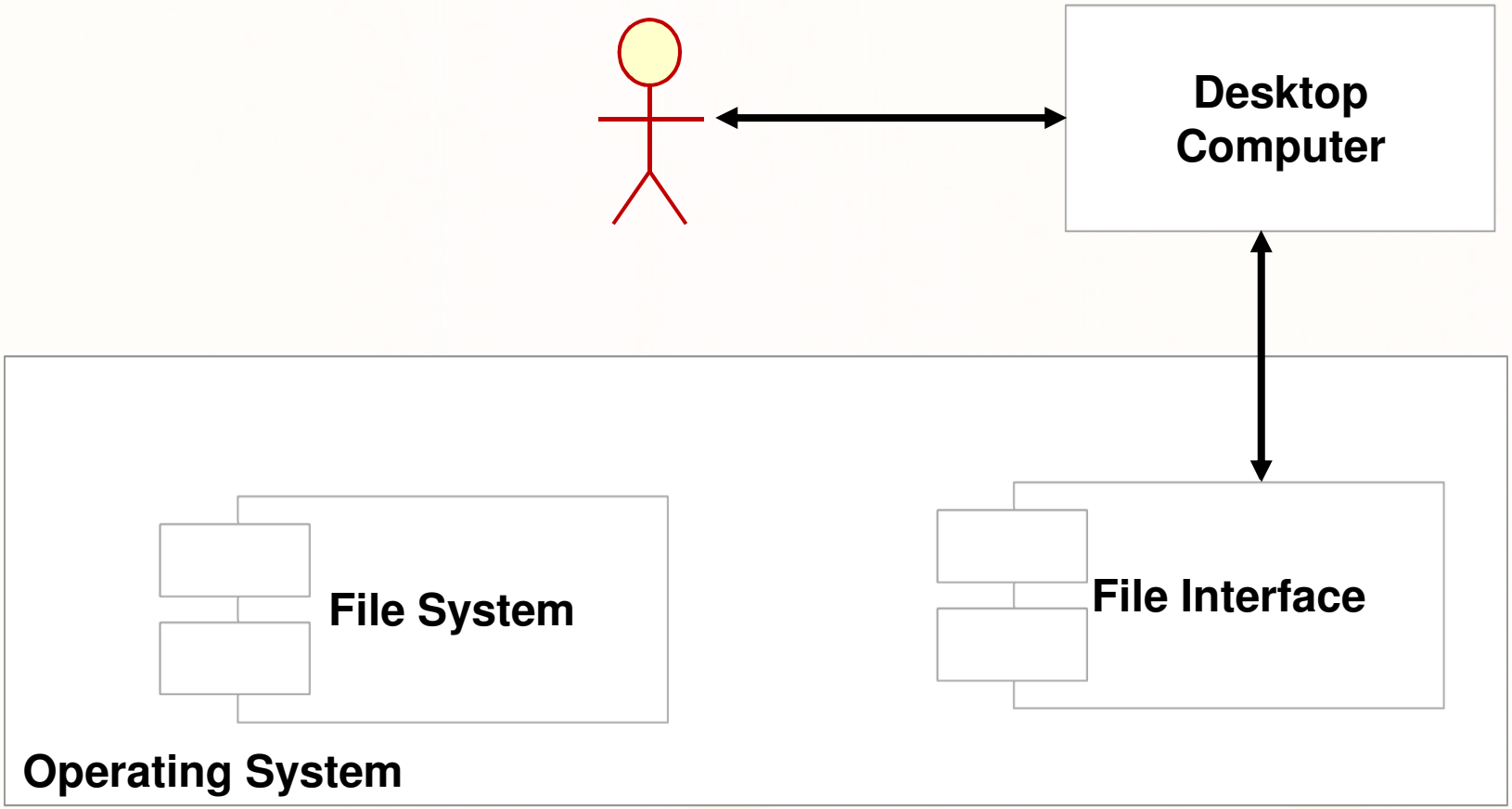
Βιβλίο Διευθύνσεων: Υλοποίηση

```
import java.io.*;
import java.util.*;

public class AddressBook {

    private Vector collection;
    private File file;
    private boolean changedSinceLastSave;

    public AddressBook() {
        collection = new Vector();
        file = null;
        changedSinceLastSave = false;
    }

    public int getNumberOfPersons() {
        return collection.size();
    }

    public void addPerson(String s, String
s1, String s2, String s3, String s4, String
s5, String s6)    {
        collection.addElement(new Person(s,
s1, s2, s3, s4, s5, s6));
        setChangedSinceLastSave(true);
    }
    public String getFullNameOfPerson(int i)
{
        Person person =
(Person)collection.elementAt(i);
        return person.getFirstName() + " " +
person.getLastName();
    }
public void updatePerson(int i, String s,
String s1, String s2, String s3, String s4)
{

(Person)collection.elementAt(i)).update(s,
s1, s2, s3, s4);
        setChangedSinceLastSave(true);
```

Βιβλίο Διευθύνσεων: Υλοποίηση

```
public class Person {

    private String firstName;
    private String lastName;
    private String address;
    private String city;
    private String state;
    private String zip;
    private String phone;

    public Person(String s, String s1, String
s2, String s3, String s4, String s5, String
s6)
    {       firstName = s;      lastName = s1;
        address = s2; city = s3; state = s4;
        zip = s5; phone = s6;  }

    public String getFirstName()    {
        return firstName;
    }
    public String getAddress()    {
        return address;
    }
    …
    public void update(String s, String s1,
String s2, String s3, String s4)
    {
        address = s;
        city = s1;
        state = s2;
        zip = s3;
        phone = s4;
    }

}
```

Βιβλίο Διευθύνσεων: Deployment Diagram



Βιβλίο Διευθύνσεων: Συντήρηση/Εξέλιξη

- Το use case Print Entries τυπώνει τα ονόματα στο System.out. Διαφορετικά θα μπορούσε να γράφει σε αρχείο.
- Το πρόγραμμα μπορεί να διαχειρίζεται πολλαπλά βιβλία διευθύνσεων. Αυτό συμπεριλαμβάνει τις εξής αλλαγές:
  - Το use case Create New Address Book και Open Existing Address Book Use Cases δεν θα κλείνουν το τρέχον address book. Αντίθετα θα δημιουργούν ένα καινούριο αντίγραφο του GUI και θα το διαχειρίζονται ξεχωριστά.
  - Θα δημιουργηθεί ένα καινούριο use case (Close Address Book) το οποίο θα εκτελείται όταν κλείνει ένα address book αντί του use case Quit Program.