Διάλεξη 12: Κληρονομικότητα (Inheritance)

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

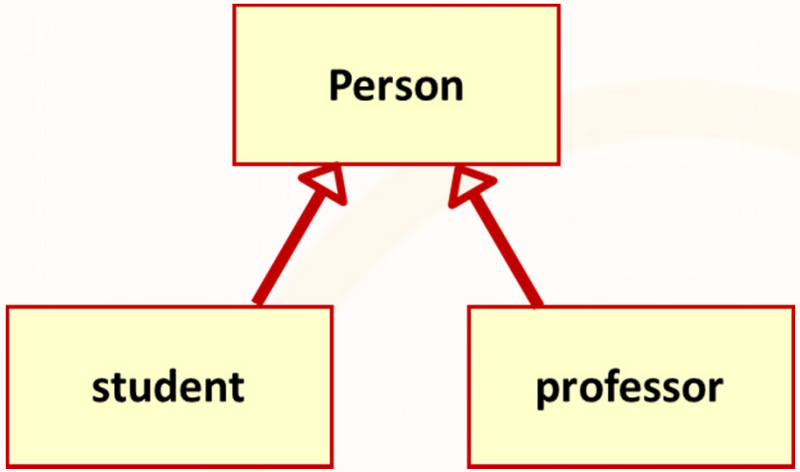
- Κληρονομικότητα και Επαναχρησιμοποίηση
- Υποκλάσεις/Υπερκλάσεις
- Απόκρυψη ονομάτων
- Το super, protected
- Απενεργοποίηση κλάσεων

Διδάσκων: Παναγιώτης Ανδρέου

Κληρονομικότητα

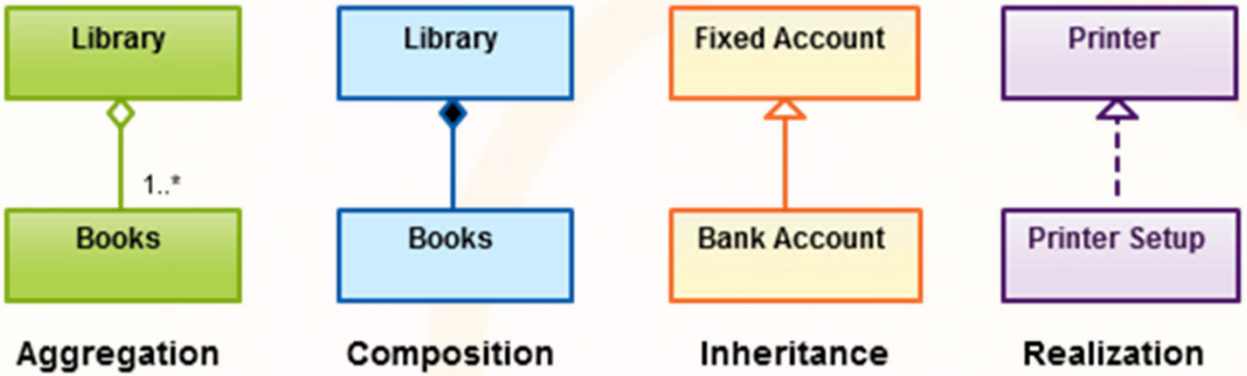
Δύο βασικές έννοιες:

- Ένα είδος σχέσης
- Ένας προγραμματιστικός μηχανισμός για επαναχρησιμοποίηση



Επαναχρησιμοποίηση λειτουργιών (Reusability)

- **Συνάθροιση (aggregation):** σε μια κλάση εντάσσουμε ως στοιχεία χειριστήρια άλλων κλάσεων. Τα στοιχεία αυτά είναι ανεξάρτητα
- **Σύνθεση (composition):** σε μια κλάση εντάσσουμε ως στοιχεία χειριστήρια άλλων κλάσεων. Τα στοιχεία αυτά παύουν να υπάρχουν όταν χαθεί το κυρίως αντικείμενο
- **Κληρονομικότητα (inheritance):** επεκτείνουμε υπάρχουσες κλάσεις, είτε προσθέτοντάς τους καινούρια στοιχεία, είτε αλλάζοντας την λειτουργικότητα μεθόδων τους
- **Πραγματοποίηση (realization):** η λειτουργικότητα που καθορίζεται από μία κλάση πρέπει να υλοποιηθεί από κάποια άλλη



Προγραμματιστικός Μηχανισμός Κληρονομικότητας

- **Μία κλάση μπορεί να κληρονομήσει όλα τα "επιτρεπτά" στοιχεία (πεδία και μεθόδους) από τον "πατέρα"** της από τον πατέρα του πατέρα της, κτλ.

- Μία υποκλάση μπορεί να **προσθέσει καινούρια πεδία**
- Μία υποκλάση μπορεί να **προσθέσει καινούριες μεθόδους**
- Μία υποκλάση μπορεί να **επεκτείνει υφιστάμενες μεθόδους (overloading)**
- Μία υποκλάση μπορεί να **επανακαθορίσει υφιστάμενες μεθόδους (overriding)**

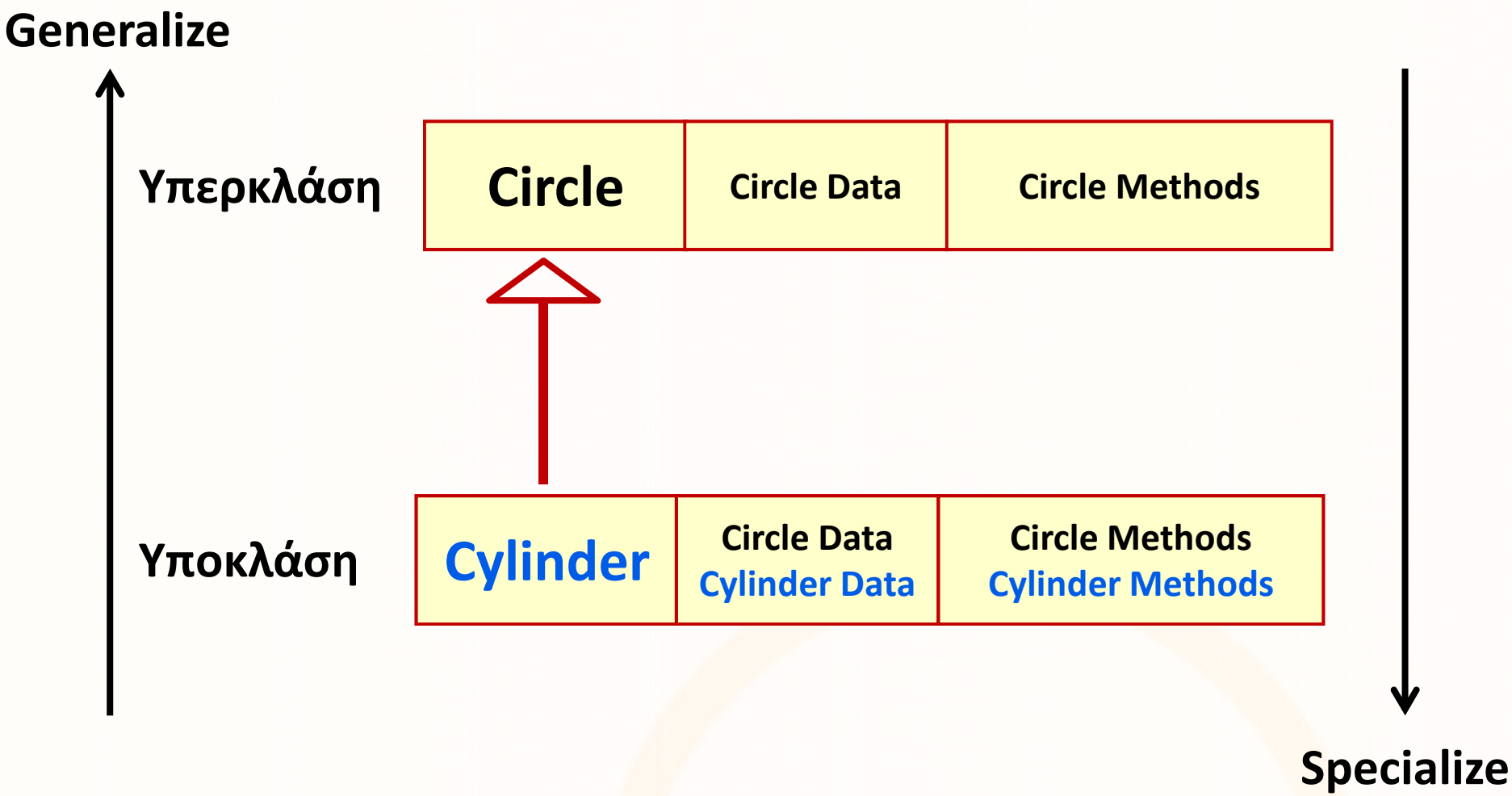
Βασικές Αρχές Κληρονομικότητας

- **Κάθε κλάση**, εκτός από την Object, **κληρονομεί από ακριβώς μία κλάση** (αντίθετα με την C++ που επιτρέπει πολλαπλή κληρονομικότητα)
- **Αν μία κλάση δεν δηλώσει κάποια υπερκλάση** (κλάση πατέρας) τότε εξορισμού **κληρονομεί από την Object**
  Συνεπώς το `class C { … }`
  σημαίνει `class C extends Object { … }`
- Σύνταξη:
  `class <όνομα κλάσης> extends <όνομα υπερκλάσης> {…}`
- **Κάθε υποκλάση κληρονομεί τα πάντα εκτός από τους constructors**
- **Αν μία κλάση δεν δηλώσει κάποια υπερκλάση** (κλάση πατέρας) τότε εξορισμού **κληρονομεί από την Object**

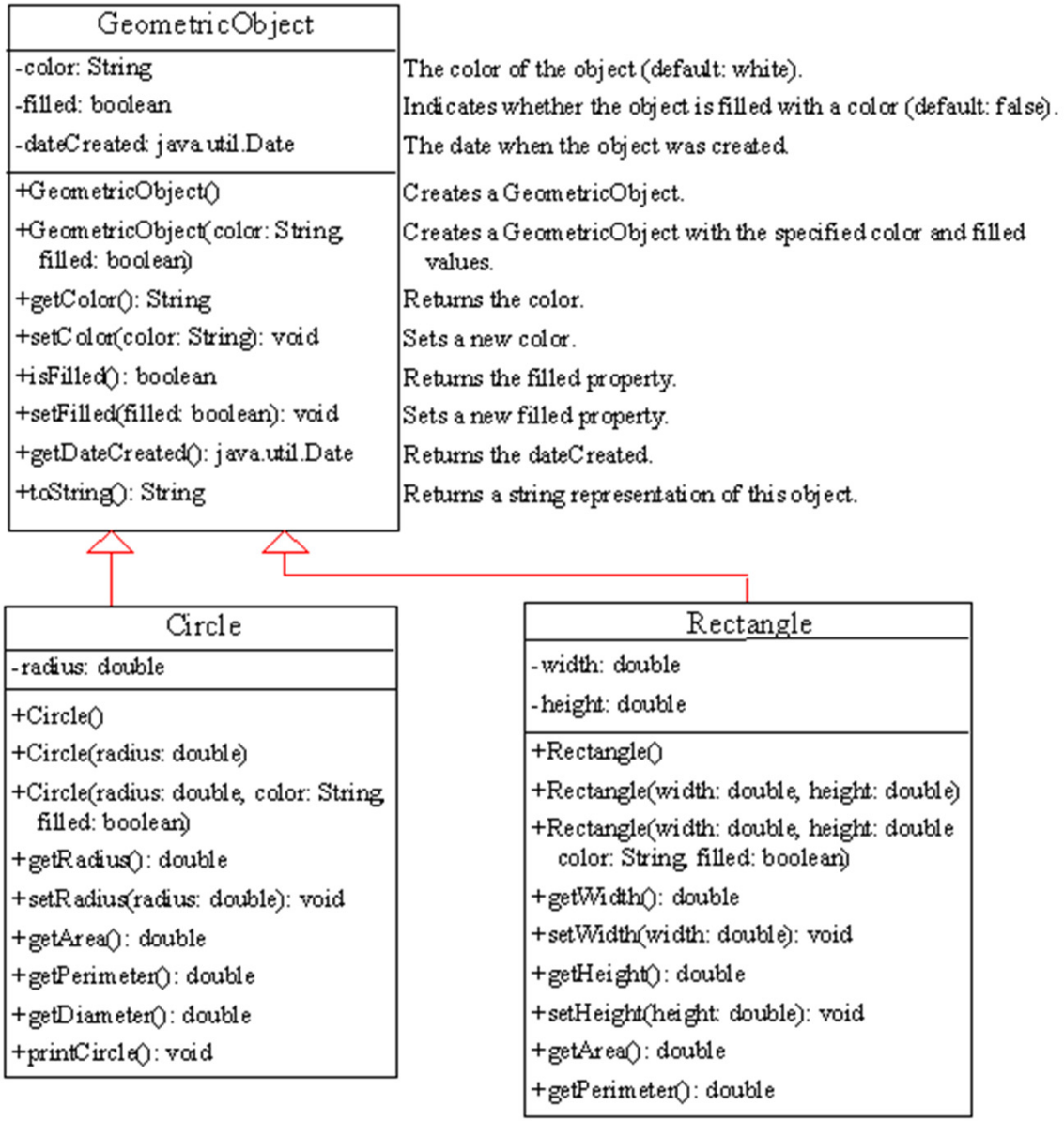
Βασικές Αρχές Κληρονομικότητας (συν.)

- Επειδή **η κλάση Object** βρίσκεται στην κορυφή της ιεραρχίας, είναι η μόνη κλάση που **δεν έχει υπερκλάση**
- **Κάθε άλλη κλάση έχει σαν πρόγονο την Object**
- **Τα πεδία και μεθόδοι της object** (π.χ., μέθοδος toString()) **κληρονομούνται από όλους τους απογόνους της Object**

Υποκλάσεις και Υπερκλάσεις



Υποκλάσεις και Υπερκλάσεις (συν.)

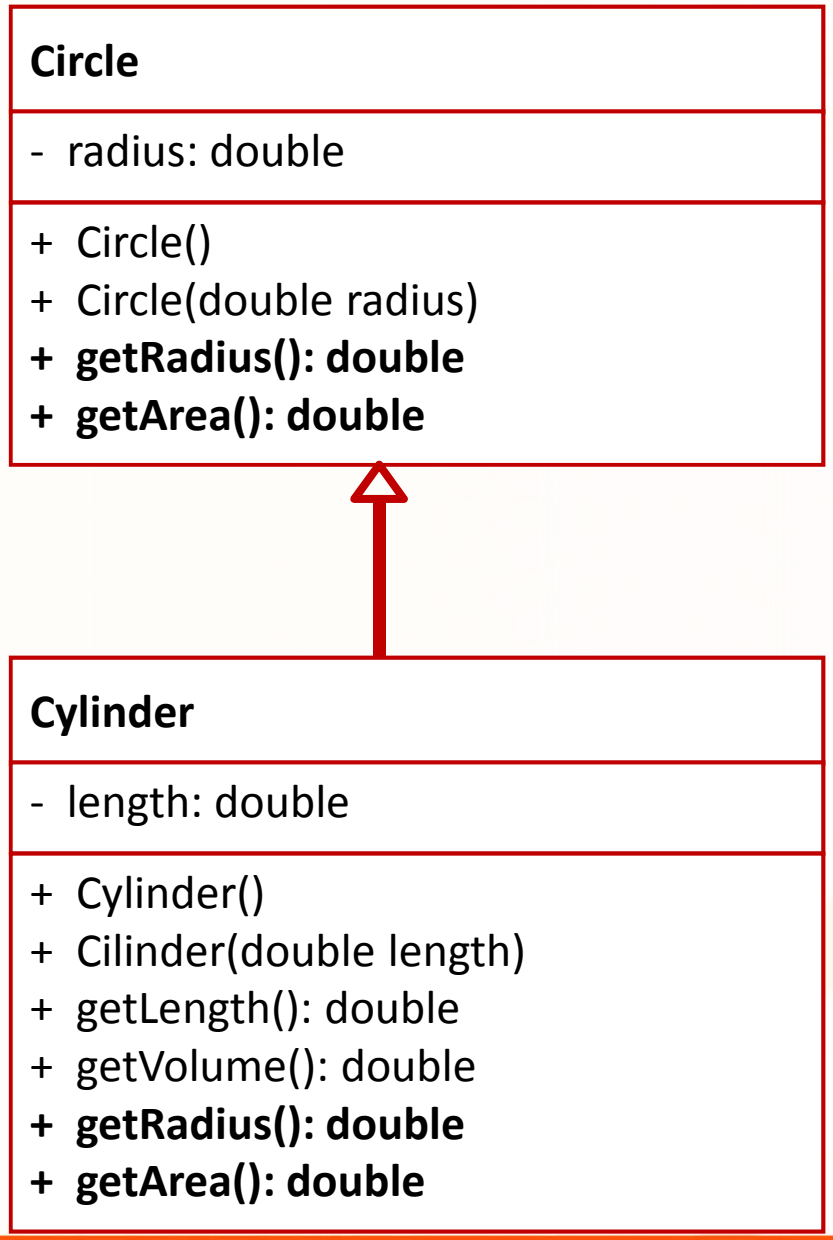


Υποκλάσεις και Υπερκλάσεις (συν.)

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

  // Δημιούργησε ένα κύκλο
  public Circle () {radius = 1.0;}

  // Δημιούργησε ένα κύκλο
  // συγκεκριμένη ακτίνα
  public Circle(double newRadius)
  {
    radius = newRadius;
  }

  // Επέστρεψε την ακτίνα
  public double getRadius() {
    return radius;
  }

  // Επέστρεψε το εμβαδό
  public double getArea() {
    return radius * radius * π;
  }
}
```

```
class Cylinder extends Circle {
  // Το μήκος αυτού του κύλινδρου
  private double length;

  // Δημιούργησε ένα κύλινδρο
  public Cylinder() {
    super();
    this.length= 1.0;}

  // Δημιούργησε ένα κύλινδρο με
  // συγκεκριμένο μήκος
  public Cylinder(int length) {
    this.length= length; }

  // Επέστρεψε το μήκος
  public double getLength() {
    return length; }

  // Επέστρεψε τον όγκο
  // αυτού του κύλινδρου
  public double getVolume () {
    return getArea() * length;}
}
```

Επέκταση λειτουργιών



- **Η υποκλάση κληρονομεί όλες τις μεθόδους (που επιτρέπει) η υπερκλάση**
- **...και επεκτείνει με περισσότερες λειτουργίες**
- Δηλαδή, τροποποιεί την λειτουργικότητα
- Πρακτικά, ένα αντικείμενο της υπερκλάσης περιτυλίγεται (wrapped) μέσα στην υποκλάση
- Το αντικείμενο της υπερκλάσης δημιουργείται, άμεσα ή έμμεσα

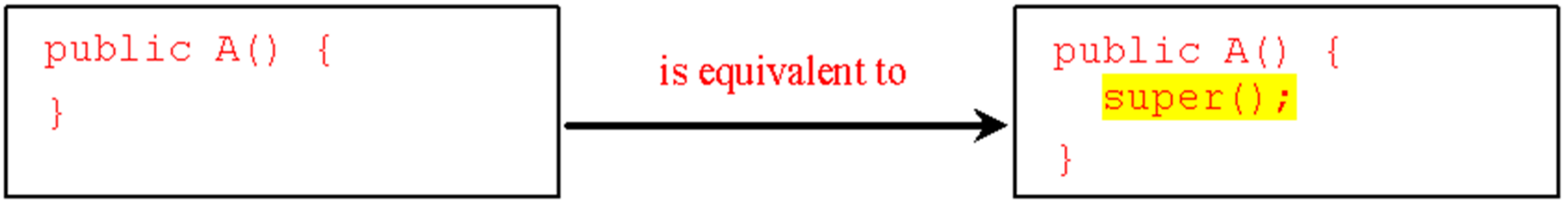
Χειριστήριο Υπερκλάσης (super)

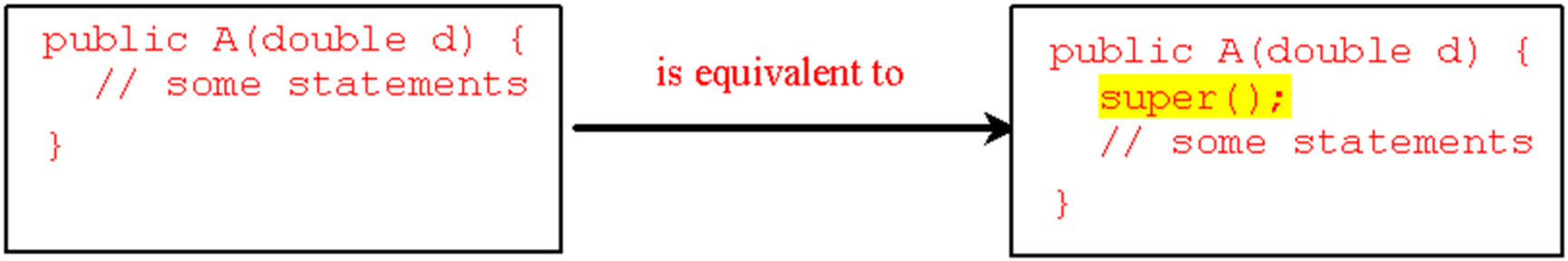
- Η λέξη κλειδί **super** αναφέρεται στην υπερκλάση
- Μπορεί να χρησιμοποιηθεί με δύο τρόπους:
  - Για το **κάλεσμα του constructor** της υπερκλάσης
  - Για το **κάλεσμα μία μεθόδου** της υπερκλάσης
- Στην περίπτωση που καλείται ο constructor της κλάσης τότε πρέπει να είναι η πρώτη δήλωση στον κώδικα.

- Σημείωση: Η λέξη κλειδί **this** αναφέρεται στην υποκλάση και είναι ένα χειριστήριο προς το ίδιο το αντικείμενο.

Κατασκευαστής Υπερκλάσης

- **Οι κατασκευαστές της υπερκλάσης δεν κληρονομούνται**
- **Επικαλούνται άμεσα** (explicitly) **και έμμεσα** (implicitly)

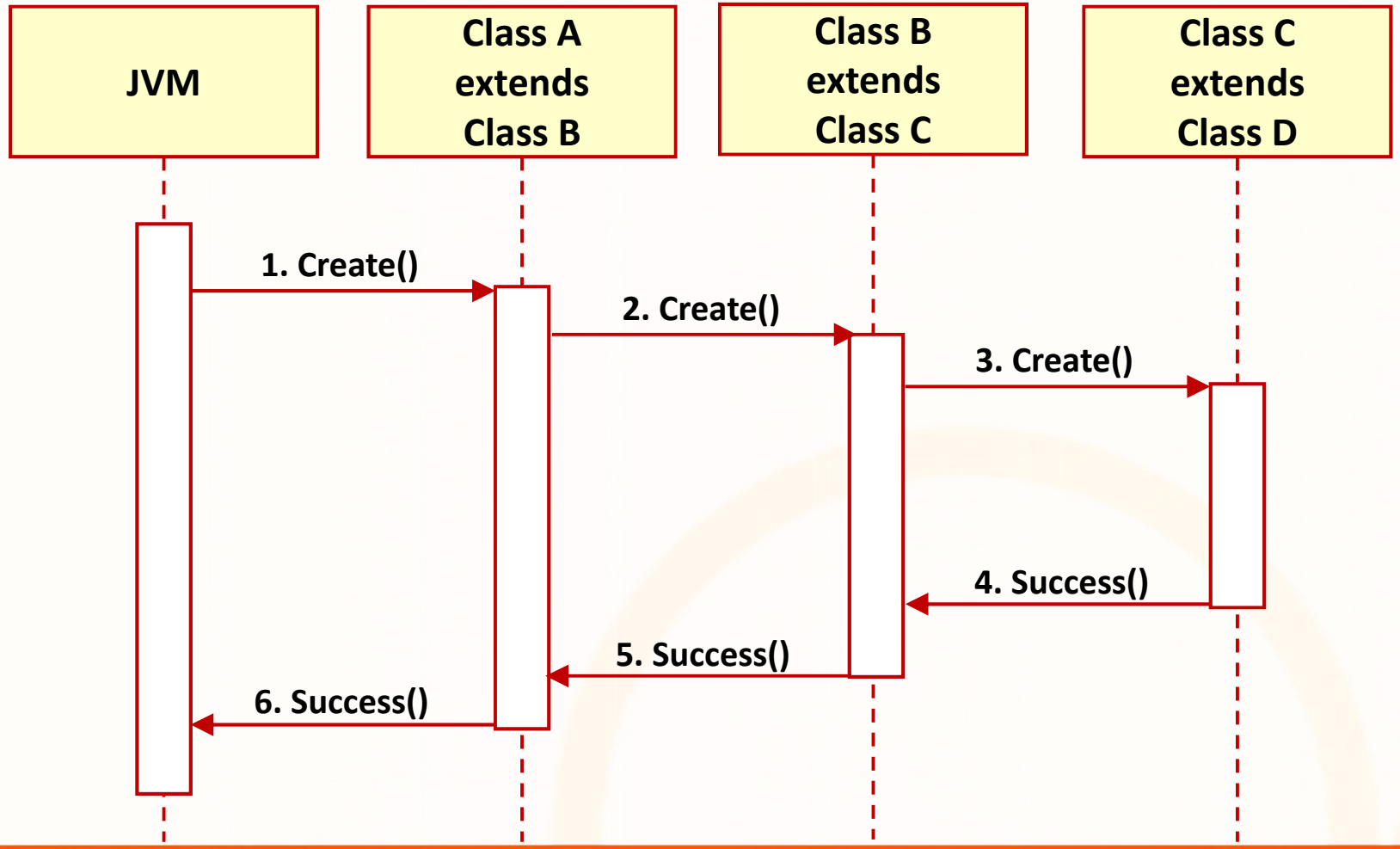




- **Στην περίπτωση που δεν καλούνται άμεσα τότε καλείται έμμεσα ο no-arg constructor της υπερκλάσης**

Αλυσίδωση Κατασκευαστών (Constructor Chaining)

- Η κατασκευή ενός στιγμιότυπου μίας υποκλάσης καλεί τους κατασκευαστές όλων των υπερκλάσεων που βρίσκονται στον μονοπάτι μέχρι τον πιο μακρύ πρόγονο



Παράδειγμα: Αλυσίδωση Κατασκευαστών

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  }

  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
  }
}

class Employee extends Person {
  public Employee() {
    this("(2) Invoke Employee’s overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }

  public Employee(String s) {
    System.out.println(s);
  }
}

class Person {
  public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
}
```

Παράδειγμα: Αλυσίδωση Κατασκευαστών

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  }

  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
  }
}

class Employee extends Person {
  public Employee() {
    this("(2) Invoke Employee’s overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }

  public Employee(String s) {
    System.out.println(s);
  }
}

class Person {
  public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
}
```

Παράδειγμα: Αλυσίδωση Κατασκευαστών

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  }

  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
  }
}

class Employee extends Person {
  public Employee() {
    this("(2) Invoke Employee's overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }

  public Employee(String s) {
    System.out.println(s);
  }
}

class Person {
  public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
}
```

3. Invoke Employee no-arg constructor

Παράδειγμα: Αλυσίδωση Κατασκευαστών

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  }

  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
  }
}

class Employee extends Person {
  public Employee() {
    this("(2) Invoke Employee's overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }

  public Employee(String s) {
    System.out.println(s);
  }
}

class Person {
  public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
}
```

4. Invoke Employee (String) constructor

Παράδειγμα: Αλυσίδωση Κατασκευαστών

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  }

  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
  }
}

class Employee extends Person {
  public Employee() {
    this("(2) Invoke Employee’s overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }

  public Employee(String s) {
    System.out.println(s);
  }
}

class Person {
  public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
}
```

5. Invoke Person() constructor

Παράδειγμα: Αλυσίδωση Κατασκευαστών

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  }

  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
  }
}

class Employee extends Person {
  public Employee() {
    this("(2) Invoke Employee’s overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }

  public Employee(String s) {
    System.out.println(s);
  }
}

class Person {
  public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
}
```

6. Execute println

Παράδειγμα: Αλυσίδωση Κατασκευαστών

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  }

  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
  }
}

class Employee extends Person {
  public Employee() {
    this("(2) Invoke Employee’s overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }

  public Employee(String s) {
    System.out.println(s);
  }
}

class Person {
  public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
}
```

Παράδειγμα: Αλυσίδωση Κατασκευαστών

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  }

  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
  }
}

class Employee extends Person {
  public Employee() {
    this("(2) Invoke Employee's overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }

  public Employee(String s) {
    System.out.println(s);
  }
}

class Person {
  public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
}
```

Παράδειγμα: Αλυσίδωση Κατασκευαστών

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  }

  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
  }
}

class Employee extends Person {
  public Employee() {
    this("(2) Invoke Employee’s overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }

  public Employee(String s) {
    System.out.println(s);
  }
}

class Person {
  public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
}
```

Που είναι το λάθος;

```
public class Apple extends Fruit { }

class Fruit {
  public Fruit(String name) {
    System.out.println(
        "Fruit's constructor is invoked");
  }
}
```

Τι θα συμβεί;

```
class Instrument {
  Instrument() {
    System.out.println("Instrument()");
  }
  Instrument( int i ) {
    System.out.println("Instrument(int i)");
  }
}
public class Wind extends Instrument {
  Wind() { System.out.println("Wind()"); }
  Wind( int i ) {
    // super();
    this();
    System.out.println("Wind(int i)");
  }
  public static void main(String args[]) {
      Wind flute = new Wind(8);
  }
}
```

Απόκρυψη Ονομάτων (Name Hiding)

Πεδία

- Μέσα σε μία κλάση, κάθε πεδίο πρέπει να έχει μοναδικό όνομα
- Μέσα από την κληρονομικότητα προκύπτουν περιπτώσεις που ένα πεδίο στην υποκλάση μπορεί να έχει το ίδιο όνομα με ένα πεδίο στην υπερκλάση
- Σε αυτή την περίπτωση, το πεδίο στην υποκλάση κρύβει/επισκιάζει (hides) το πεδίο στην υπερκλάση
- Για να έχουμε πρόσβαση στο πεδίο της υπερκλάσης πρέπει να χρησιμοποιήσουμε την λέξι κλειδί super

Μεθόδοι

- Παρόμοια υπάρχει περίπτωση μία μέθοδος στην υποκλάση να έχει την ίδια υπογραφή με μία μέθοδο της υπερκλάσης
- Σε αυτή την περίπτωση, η μέθοδος της υποκλάσης υπερσκελίζει (overrides) την μέθοδο της υπερκλάσης

Παράδειγμα Υπερσκέλισης (Overriding)

```
public class Circle extends GeometricObject {
  // Other methods are omitted

  /** Override the toString method
      defined in GeometricObject */
  public String toString() {
    return super.toString() +
           "\nradius is " + radius;
  }
}
```

Σύνθεση και Κληρονομικότητα

- **Σύνθεση :**
  - Η σύνθεση χρησιμοποιείται γενικότερα όταν χρειαζόμαστε τα χαρακτηριστικά μιας υπάρχουσας κλάσης και όχι την διαπροσωπεία της.
  - Αντιστοιχεί σε **has-a** σχέσεις ανάμεσα στις κλάσεις .
- **Κληρονομικότητα :**
  - Χρησιμοποιείται όταν θέλουμε να φτιάξουμε μια ειδική περίπτωση κάποιας κλάσης.
  - Αν μια κλάση Β κληρονομεί από μια κλάση Α , η σχέση τους συνοψίζεται ως εξής : Η νέα κλάση Β είναι ένας τύπος της υπάρχουσας κλάσης Α .
  - Αντιστοιχεί σε **is-a** σχέσεις ανάμεσα στις κλάσεις.

Σύνθεση και Κληρονομικότητα

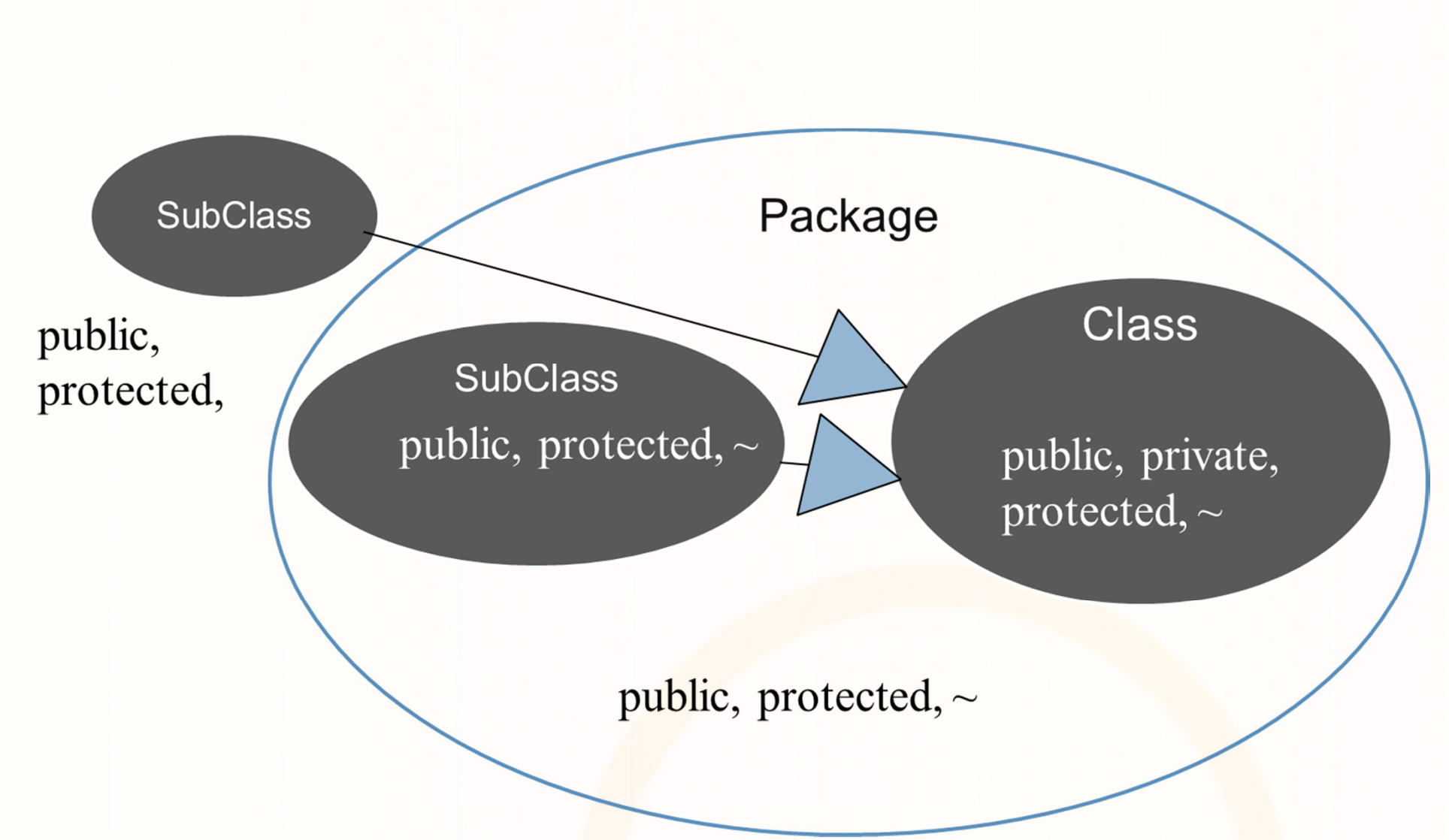
```
public class PlaceSetting extends Custom {
   Spoon sp;
   Fork frk;
   Knife kn;
   DinnerPlate pl;
   PlaceSetting(int i) {
        super(i + 1);
        sp = new Spoon(i + 2);
        frk = new Fork(i + 3);
        kn = new Knife(i + 4);
        pl=new DinnerPlate(i + 5);
   }
   public static void main (String[] args) {
        PlaceSetting x=new PlaceSetting(9);
      }
}
```

- Ο μεταφραστής μας εξαναγκάζει να αρχικοποιήσουμε τους κληροδότες, αλλά εμείς πρέπει να φροντίσουμε για την αρχικοποίηση των αντικειμένων-στοιχείων.

Τροποποιητές Πρόσβασης: Σύγκριση

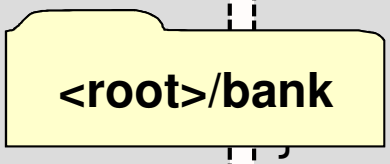
Τροποποιητής Πρόσβασης	Ίδια Κλάση	Ίδιο Πακέτο	Υποκλάση	Άλλα Πακέτα
public	√	√	√	√
protected	√	√	√	
(default) friendly	√	√		
private	√			

Τροποποιητές Πρόσβασης: Σύγκριση (συν.)

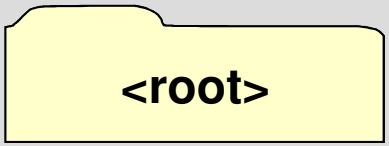


protected: Παράδειγμα 1

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




```
package bank;
public class SavingsAccount
extends BankAccount{
   public SavingsAccount(){
      balance2 = 0.0; }
}
```




```
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
   }
}
```

protected: Παράδειγμα 2

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

<root>/bank

```
package client;
import  bank.BankAccount;
public class Client extends
BankAccount {
   private String name;
   public Client(){
      name = “test”;
   }
  … }
```

<root>/client

```
import bank.*;
public class TestBankAccount {
  public static void main(String args[]) {
    BankAccount ba1 = new BankAccount();
    //compile error constructor BankAccount() does not exist
    Client c = new Client();
    // OK. Client() exists AND Client has access to
    // BankAccount() even though it is in a different package
  }
}
```

<root>

Απενεργοποίηση της Κληρονομικότητας

- Με τη χρήση του τροποποιητή **final**

```
public final class Math { }

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