ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

ΠΛΗΡΟΦΟΡΙΚΗ ΙΙ

Ενότητα 4: Αντικειμενοστραφής Προγραμματισμός (Μέθοδοι, Κλάσεις, Αντικείμενα)

Μιχάλης Δρακόπουλος

Σχολή Θετικών επιστημών

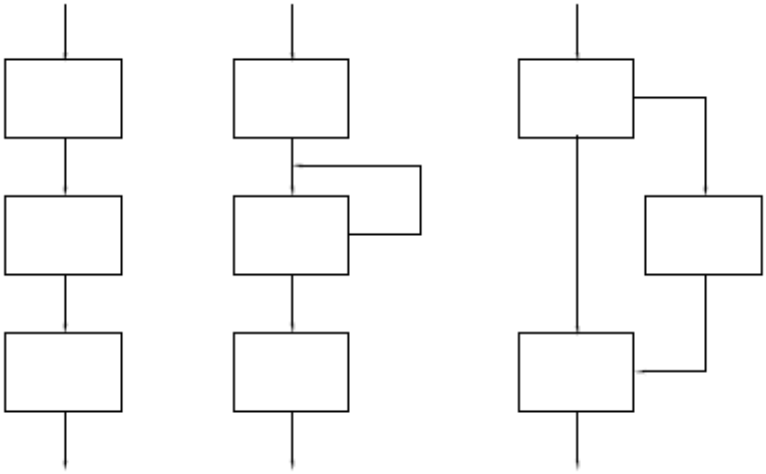
Τμήμα Μαθηματικών

ΠΛΗΡΟΦΟΡΙΚΗ ΙΙ (Java)
Ενότητα 4

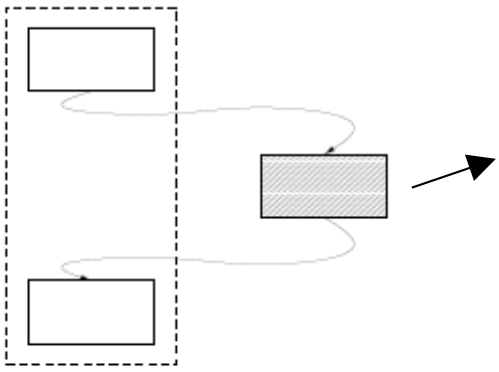
Αντικειμενοστραφής Προγραμματισμός (Μέθοδοι, Κλάσεις, Αντικείμενα)

Με τον όρο «ροή προγράμματος» αναφερόμαστε στη σειρά με την οποία εκτελούνται οι εντολές ενός προγράμματος. Μέχρι τώρα έχουμε αναφερθεί στις ακόλουθες μορφές ροής:

Σειριακή Επαναληπτική Υπό συνθήκη



Μια άλλη μορφή ροής ενός προγράμματος είναι αυτή που πραγματοποιείται μέσω της κλήσης μεθόδων:



μέθοδος (συνάρτηση) που μπορεί να βρίσκεται μέσα στην κλάση ή και μέσα σε κάποια άλλη κλάση.

Μέθοδοι

Μέθοδος ονομάζεται ένα σύνολο συγκεντρωμένων εντολών με χαρακτηριστικό όνομα. (Είναι το αντίστοιχο των «συναρτήσεων» άλλων γλωσσών προγραμματισμού)

Η φιλοσοφία των μεθόδων:

- Εντοπισμός κάποιου υπο-προβλήματος που χρειάζεται να επιλυθεί σαν μέρος του προγράμματος
- Αλγοριθμική επίλυση του υπο-προβλήματος και εγγραφή του αντίστοιχου κώδικα (μόνο μία φορά)
- Ονομασία του κώδικα του υπο-προβλήματος → μέθοδος
- Χρήση της μεθόδου (κλήση, εκτέλεση) (με το όνομά της) κάθε φορά που παρουσιάζεται το ίδιο υπο-πρόβλημα.

Βασικά χαρακτηριστικά μεθόδων:

- Επαναχρησιμοποιήσιμος κώδικας σε άλλα προγράμματα
- Μη επανάληψη κώδικα στο ίδιο πρόγραμμα
- «Μαύρα κουτιά» → ενδιαφέρει το τί κάνουν, όχι το πώς το κάνουν
- Βοηθούν στη λογική σχεδίαση του προγράμματος
- Αποκρύπτουν προγραμματιστικές λεπτομέρειες

Είδη μεθόδων:

- Ορισμένες από τον προγραμματιστή
- Βιβλιοθήκες (μέρος της Java), π.χ. println(), nextInt(), sqrt() κτλ.

Σχεδιασμός μεθόδων:

- Κάθε μέθοδος καλείται από κάποια άλλη μέθοδο
- Μπορεί να έχει δεδομένα εισόδου (τα παίρνει από τη μέθοδο που την καλεί)
- Μπορεί να έχει δεδομένα εξόδου (τα επιστρέφει στη μέθοδο που την καλεί)
- Έχει ένα όνομα (συνήθως μοναδικό, αλλά όχι πάντα → βλ. Ενότητα 5)
- Έχει ένα σύστημα ορισμού της επικοινωνίας της με το περιβάλλον

Τα δύο τελευταία δηλώνονται με τον ορισμό της μεθόδου στην πρώτη της γραμμή στον κώδικα.

Ορισμός μεθόδων:

```
[ορατότητα] <τύπος> <όνομα> ([παράμετροι])
     ↓          ↓
  public      void
  private     int
     .       double
     .         .
     .         .
```

Οι παράμετροι, εάν υπάρχουν, είναι π.χ. της μορφής:

```
... <όνομαΜεθόδου> (<τύπος> <παράμ1>, <τύπος> <παράμ2>, ...)
{
    ... x = παράμ1;
    ... y = 5 * παράμ2;
          ...
}
```

Τι είναι το κάθε στοιχείο του ορισμού:

i) **ορατότητα** (public/private): ορίζει το από πού είναι ορατή η μέθοδος.
public: η μέθοδος μπορεί να κληθεί και από μεθόδους άλλων κλάσεων, όχι μόνο της κλάσης μέσα στην οποία βρίσκεται.
private: η μέθοδος μπορεί να κληθεί μόνο από μεθόδους της ίδιας της κλάσης στην οποία βρίσκεται.

ii) **τύπος**: - void εάν δεν επιστρέφει τίποτα
- πρωτογενής τύπος (int, double, boolean, char, κτλ.) ανάλογα με τον τύπο της μεταβλητής που επιστρέφει
- String εάν επιστρέφει κάποιο αλφαριθμητικό

iii) **παράμετροι:** μεταβλητές εισόδου, οι οποίες έχουν συγκεκριμένες τιμές στη μέθοδο κλήσης της μεθόδου και χρησιμοποιούνται με αυτές τις τιμές μέσα στη μέθοδο. (Αν δεν υπάρχουν παράμετροι εισόδου, οι παρενθέσεις απλά μένουν κενές)

Η εντολή return:

- Χρησιμοποιείται στις μεθόδους συγκεκριμένου τύπου (δηλ. όχι void) για να επιστρέψει την τιμή της μεταβλητής εξόδου:

```
        return μεταβλητή;
ή      return έκφραση;
```

π.χ.
```
public int method1(int x)
{
   int y = 2*x;      // θα μπορούσε να γραφεί κατευθείαν:
   return y;         // return 2*x;
}
```

- Χρησιμοποιείται στις `void` μεθόδους σαν εντολή εξόδου από τη μέθοδο (και επιστροφής της ροής του κώδικα στην αρχική μέθοδο κλήσης).

  Η χρήση αυτή είναι σπάνια, συνήθως χρησιμοποιείται υπό συνθήκη και καλό είναι να αποφεύγεται (υπάρχουν καλύτεροι τρόποι που επιτυγχάνουν ανάλογο αποτέλεσμα)

Ένα απλό παράδειγμα μιας κλάσης με δύο μεθόδους (κλήση μεθόδου μέσα στην ίδια κλάση):

(Η μέθοδος `square` ορίζεται ως `static` για να μπορεί να την καλέσει απ' ευθείας η μέθοδος `main` η οποία είναι και αυτή `static`. Η λέξη `static` καθώς και το τι ακριβώς συμβαίνει με τις `static` μεθόδους και μεταβλητές, θα αναφερθούν σε επόμενη ενότητα, μετά της εισαγωγή στον αντικειμενοστραφή τρόπο προγραμματισμού)

```
import java.util.*;
public class Example
{

   public static double square(double x)
   {
      double s = x*x;
      return s;
   }

   public static void main(String [] args)
   {
      Scanner input = new Scanner(System.in);

      System.out.println("Δώσε έναν αριθμό:");
      double n = input.nextDouble();

      double z = square(n);

      System.out.println(n + " στο τετράγωνο: " + z);
   }

}
```

Παράδειγμα εκτέλεσης του προγράμματος:

```
> Δώσε έναν αριθμό:
> 5
> 5 στο τετράγωνο: 25
>
```

Άλλο ένα παράδειγμα με κλήση μεθόδου στην ίδια κλάση:

```
import java.util.*;
public class Circle
{

   //μέθοδος που υπολογίζει εμβαδόν κύκλου:
   public static double circleArea(double r)
   {
      return Math.PI*r*r;
   }

   // μέθοδος που υπολογίζει εμβαδόν δακτυλίου:
   public static double ringArea(double rIn, double rOut)
   {
      double innerArea = circleArea(rIn);
      double outterArea = circleArea(rOut);
      return (outterArea – innerArea);
   }

   // Η μέθοδος main:

   public static void main(String [] args)
   {
      Scanner input = new Scanner(System.in);

      double area;

      System.out.println("Ακτίνα 1?");
      double r1 = input.nextDouble();
      System.out.println("Ακτίνα 2?");
      double r2 = input.nextDouble();

      if (r1 < r2)
         area = ringArea(r1,r2);
      else
         area = ringArea(r2,r1);

      System.out.println("Το εμβαδόν του δακτυλίου είναι: " + area);
   }

}
```

ΚΛΑΣΗ: σύνθετος τύπος δεδομένων

Οι κλάσεις αποτελούν τη βασική δομή του αντικειμενοστραφούς προγραμματισμού. Ουσιαστικά μια κλάση είναι ο ορισμός ενός σύνθετου τύπου δεδομένων από τον προγραμματιστή. Οι μεταβλητές που υλοποιούν έναν τέτοιο σύνθετο τύπο δεδομένων (μια κλάση) ονομάζονται **αντικείμενα** (objects). Όπως δηλαδή οι απλές μεταβλητές είναι κάποιου συγκεκριμένου πρωτογενούς τύπου (π.χ., `int`, `double` κτλ), έτσι και τα αντικείμενα είναι σύνθετες μεταβλητές, τύπου κάποιας κλάσης.

Η μεγάλη διαφορά μεταξύ των πρωτογενών τύπων και των κλάσεων είναι ότι οι πρωτογενείς τύποι προορίζονται για τη δημιουργία απλών μεταβλητών που αποθηκεύουν ένα απλό δεδομένο (π.χ., έναν ακέραιο αριθμό, ένα χαρακτήρα κτλ), ενώ οι κλάσεις είναι πιο πολύπλοκες και προορίζονται για τη δημιουργία οντοτήτων (αντικειμένων) τα οποία "ομαδοποιούν" πολλά δεδομένα (χαρακτηριστικά) καθώς και συναρτήσεις (συμπεριφορές).

Όλα τα αντικείμενα που έχουν κοινά χαρακτηριστικά ανήκουν στην ουσία στην ίδια κλάση. Η κλάση είναι ένα "καλούπι" που ορίζει αυτά τα χαρακτηριστικά και το κάθε αντικείμενο που δημιουργείται με βάση αυτή την κλάση, έχει δυνατότητα να "αποθηκεύσει" συγκεκριμένες τιμές για καθένα από αυτά τα χαρακτηριστικά που ορίζει η κλάση του.

Έχει αναφερθεί σε προηγούμενη ενότητα ότι οι τύποι (μεταβλητών, μεθόδων, κτλ.) στη Java μπορούν γενικά (για διδακτικούς σκοπούς) να χωριστούν σε τρεις κατηγορίες:

Τύποι (types):

- Πρωτογενείς (primitive types) (`int`, `double`, `char`, κτλ.)
- String (αλφαριθμητικά)
- **Κλάσεις** → σύνθετη δομή σχεδιασμένη από τον προγραμματιστή

Όπως μια μεταβλητή, π.χ. `a`, μπορεί να είναι ακέραια:

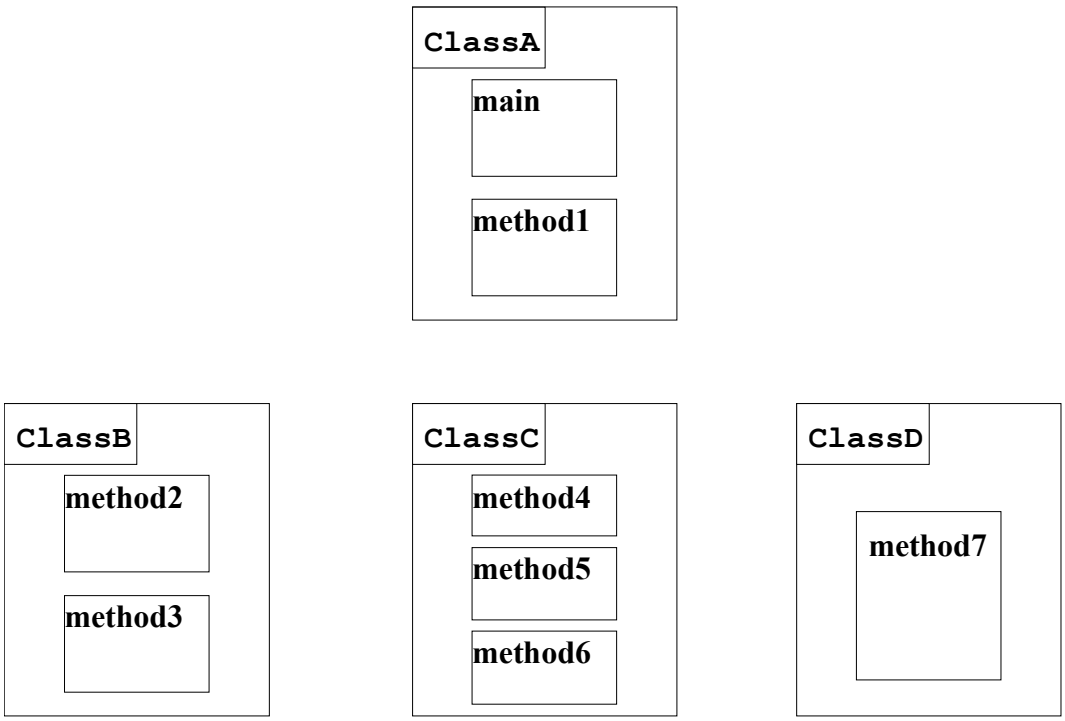
```
int a;
```

έτσι και ένα αντικείμενο, π.χ. `ferrari`, μπορεί να είναι της κλάσης `Car` (ένα *αντικείμενο (object)* λέγεται και *στιγμιότυπο (instance)* μιας κλάσης):

```
Car ferrari = new Car();
```

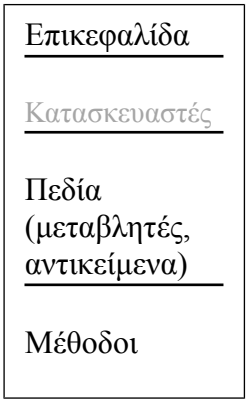
Δηλαδή, ενώ μια *μεταβλητή* είναι "κάποιου πρωτογενούς τύπου", ένα *αντικείμενο* είναι "τύπου κάποιας κλάσης". Ουσιαστικά, όπως προαναφέρθηκε, τα αντικείμενα είναι "σύνθετες μεταβλητές" και άρα οι κλάσεις είναι "σύνθετοι τύποι δεδομένων". *(Οι μεταβλητές τύπου String είναι στην ουσία αντικείμενα, αφού η String είναι μια κλάση. Περισσότερα για αυτό, στην Ενότητα 6).*

Ένα πρόγραμμα μπορεί να έχει πολλές κλάσεις και κάθε κλάση μπορεί να έχει πολλές μεθόδους. Η μέθοδος `main` είναι πάντα *μία* σε ένα πρόγραμμα. Η κλάση που την περιέχει λέγεται κλάση εφαρμογής. Οι υπόλοιπες κλάσεις λέγονται κλάσεις υποστήριξης. Δηλαδή, ένα πρόγραμμα μπορεί να έχει την ακόλουθη δομή:



Κλάση εφαρμογής: `ClassA`
Κλάσεις υποστήριξης: `ClassB, ClassC, ClassD`

ΔΟΜΗ ΚΛΑΣΗΣ:



Παράδειγμα Κλάσης:

Επικεφαλίδα: >
Πεδία: >
>
Μέθοδος: >
>
>

```
public class ToKelvin
{
    private double C2K = 273.15;

    public double returnKelvin (double tempC)
    {
        return (tempC + C2K);
    }
}
```

(Πλήρης) ορισμός μεταβλητής:

```
[ορατότητα] <τύπος> <όνομα> [= <τιμή>];
     ↓          ↓
  public       int
  private    double
     .          .
     .          .

π.χ.: public double var1;
      private int var2 = 5;
```

Οι μεταβλητές αυτές ορίζονται *εκτός μεθόδων (συνήθως στην αρχή της κλάσης)* και ονομάζονται ***μεταβλητές κλάσης***, αφού ανήκουν σε όλη την κλάση, δηλαδή σε όλες τις μεθόδους της. Οι μεταβλητές που ορίζονται *μέσα* σε μεθόδους ανήκουν μόνο στη μέθοδο στην οποία ορίζονται και ονομάζονται ***τοπικές μεταβλητές***. Έτσι, ορίζεται η έννοια της *εμβέλειας* μεταβλητής:

Εμβέλεια μεταβλητής (scope): τα σημεία μιας κλάσης στα οποία υφίσταται κάποια μεταβλητή (δηλ., εάν είναι μεταβλητή κλάσης ή τοπική μεταβλητή (μιας μεθόδου ή ενός βρόχου κτλ.)).

Ορατότητα (μεταβλητής ή μεθόδου): ορίζει σε ποιες κλάσεις ενός προγράμματος μπορεί να χρησιμοποιηθεί μια μέθοδος ή μια μεταβλητή. Όταν π.χ. μια μεταβλητή έχει ορατότητα `private` μπορεί να χρησιμοποιηθεί μόνο από μεθόδους της κλάσης στην οποία ορίζεται, ενώ αν έχει ορατότητα `public` μπορεί να χρησιμοποιηθεί από όλες τις κλάσεις του προγράμματος. Υπάρχουν και άλλες μορφές ορατότητας (που έχουν να κάνουν με την κληρονομικότητα και τα πακέτα), οι οποίες θα αναφερθούν σε επόμενες ενότητες.

Τύποι μεθόδων:

Στον *αντικειμενοστραφή προγραμματισμό*, διακρίνουμε τρεις βασικούς τύπους μεθόδων:

1) **Λειτουργικές μέθοδοι (Operation methods):**

```
public void opMethod()
{
      //δηλώσεις;
      //υπολογισμοί;
}
```

(είναι πάντα `void`, δεν επιστρέφουν τίποτα, απλά υπολογίζουν κάτι)

2) **Τροποποιητικές μέθοδοι (Modifier methods ή setter)**

```
private double var;

public void modMethod(double x)
{
   var = x;
}
```

(δέχονται πάντα κάποια παράμετρο εισόδου (π.χ. `x`), την τιμή της οποίας «περνάνε» σε κάποια `private` μεταβλητή της κλάσης τους (π.χ. `var`))

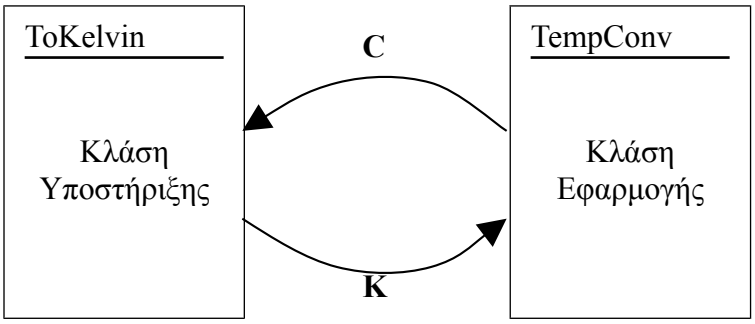
3) **Μέθοδοι πρόσβασης (Accessor methods ή getter)**

```
private int var2;

public int accessMethod()
{
   return var2;
}
```

(είναι πάντα `public`, επιστρέφουν (`return`) μια `private` μεταβλητή και είναι του ίδιου τύπου με τη μεταβλητή που επιστρέφουν – στο παραπάνω παράδειγμα `int`)

Συχνά υπάρχουν μέθοδοι που είναι συνδυασμοί των παραπάνω τύπων, κυρίως των τύπων (2) και (3). Η κλάση `ToKelvin` του παραπάνω παραδείγματος είναι μια *κλάση υποστήριξης*. Για να ολοκληρωθεί το πρόγραμμα πρέπει να φτιάξουμε μια *κλάση εφαρμογής* (`TempConv`), έτσι ώστε η κλάση εφαρμογής να "στέλνει" βαθμούς Κελσίου στην κλάση υποστήριξης και η κλάση υποστήριξης με τη σειρά της να τους μετατρέπει σε βαθμούς Kelvin και να τους επιστρέφει στην κλάση εφαρμογής:



Η κλάση εφαρμογής:

```
public class TempConv
{
   public static void main (String [ ] args)
   {
      // δημιουργία μεταβλητών:

      double C = 15.5;  // μια θερμοκρασία σε βαθμούς Κελσίου
      double K;

      // δημιουργία αντικειμένου της κλάσης ToKelvin:

      ToKelvin toK = new ToKelvin();

      // κλήση της μεθόδου returnKelvin με αποστολή  της θερμοκρασίας
      // σε βαθμούς Κελσίου και επιστροφή της σε βαθμούς Kelvin:

      K = toK.returnKelvin(C);

      // εκτύπωση στην οθόνη της θερμοκρασίας, σε Kelvin:

      System.out.println(K);

   }  // end main
}  // end class
```

Σε αυτό το παράδειγμα, βλέπουμε τον τρόπο κλήσης μιας μεθόδου από μια μέθοδο κάποιας άλλης κλάσης. Αναλυτικά:

Κλήση μεθόδου του αντικειμένου μιας κλάσης:

- Δημιουργία αντικειμένου της κλάσης:

<Κλάση> <αντικείμενο> = new <Κλάση>( );

Παράδειγμα από την παραπάνω κλάση (TempConv):

```
ToKelvin toK = new ToKelvin();
```

- Κλήση μεθόδου της κλάσης (στην ουσία, *του αντικειμένου* της κλάσης):

<αντικείμενο>.<μέθοδος>( );

Αν η μέθοδος δεν είναι void, δηλαδή αν επιστρέφει κάτι, μπορεί να κληθεί (και συνήθως καλείται) ως εξής:

<μεταβλητή> = <αντικείμενο>.<μέθοδος>( );

Παράδειγμα από την παραπάνω κλάση (TempConv):

```
K = toK.returnKelvin(C);
```

Παραδείγματα:

Κλάση υποστήριξης:

```
public class ClassA
{
     // μεταβλητές κλάσης:
   private int a = 5;
   private int b;
   private double c;

     // τροποποιητική μέθοδος:
   public void setB(int x)
   {
      b = x;
   }

     // λειτουργική μέθοδος:
   public void calcC()
   {
      c = 0.5*(a+b);
   }

     // μέθοδος πρόσβασης:
   public double getC()
   {
      return c;
   }
}
```

Κλάση εφαρμογής:

```
public class ClassB
{
   public static void main (String [ ] args)
   {
      // δημιουργία 4 αντικειμένων της ClassA:
      ClassA obj1 = new ClassA();
      ClassA obj2 = new ClassA();
      ClassA obj3 = new ClassA();
      ClassA obj4 = new ClassA();

      // κλήση διαφόρων μεθόδων για τα αντικείμενα:
      obj1.setB(10);
      obj1.calcC();
      obj2.setB(20);
      obj2.calcC();
      obj3.setB(15);
      obj4.calcC();

      // εκτύπωση στην οθόνη:
      System.out.println(obj1.getC());  // θα τυπώσει: 7.5
      System.out.println(obj2.getC());  // θα τυπώσει: 12.5
      System.out.println(obj3.getC());  // θα τυπώσει: 0
      System.out.println(obj4.getC());  // θα τυπώσει: 2.5

   }  // end method

}  // end class
```

Το `obj1.getC()` είναι 7.5 γιατί έχουμε καλέσει πριν τη μέθοδο `setB` για το συγκεκριμένο αντικείμενο (`obj1`) στέλνοντας στο `b` την τιμή 10, οπότε το `c` υπολογίζεται σαν 0.5*(5+10). Ομοίως για το `obj2.getC()`, μόνο που τώρα η τιμή του `b` είναι 20, οπότε το `c` γίνεται 12.5. Στην περίπτωση του `obj3.getC()`, η τιμή είναι 0 γιατί *δεν* έχουμε καλέσει τη μέθοδο `calcC` *για το αντικείμενο αυτό* (`obj3`), οπότε δεν έχει υπολογιστεί κάποια τιμή για το `c` αυτού του αντικειμένου. Αντίστοιχα, στην περίπτωση του αντικειμένου `obj4`, δεν στάλθηκε κάποια τιμή στη μεταβλητή `b` του αντικειμένου αυτού (μέσω της `setB(...)`), οπότε το `c` που υπολογίζεται είναι το 0.5*(5+0) = 2.5.

Έχοντας δηλώσει **`private`** τις μεταβλητές της κλάσης υποστήριξης (ClassA) και κυρίως τη μεταβλητή `c`, **δεν** επιτρέπεται στην `ClassB` να αλλάξει την τιμή τους όπως θέλει, άρα π.χ. η `c` αλλάζει *μόνο* όπως ο προγραμματιστής έχει επιλέξει στην `ClassA` (μέσω της `calcC()`). Αυτός θεωρείται **ασφαλής τρόπος προγραμματισμού** και αποτελεί μία από τις πιο βασικές αρχές του αντικειμενοστραφούς προγραμματισμού, την **ενθυλάκωση** (encapsulation).

Ένα άλλο χαρακτηριστικό παράδειγμα που δείχνει ότι οι μέθοδοι και οι μεταβλητές μιας κλάσης ανήκουν στην ουσία στα αντικείμενά της:

Αν έχουμε μια κλάση υποστήριξης:

```
public class Dog
{
  private String color;

  public void setColor(String a)
  {
    color = a;
  }

  public String accessColor()
  {
    return color;
  }
}
```

τότε από κάποια άλλη κλάση υποστήριξης ή από την κλάση εφαρμογής, μπορούμε να φτιάξουμε αντικείμενα της κλάσης `Dog`, να στείλουμε τα χρώματα των αντικειμένων αυτών, ή από κάπου αλλού να «διαβάσουμε» τα χρώματά τους:

```
// δημιουργία αντικειμένων της κλάσης Dog
Dog azor = new Dog();
Dog ivan = new Dog();

// αποστολή του χρώματος του κάθε αντικειμένου
azor.setColor("μαύρο");
ivan.setColor("άσπρο");
  .
  .
  .
// Όταν κάπου αλλού θέλουμε να ανακτήσουμε τα χρώματα
// των αντικειμένων που έχουμε φτιάξει:

System.out.println("Το χρώμα του Αζώρ είναι " + azor.accessColor());
System.out.println("Το χρώμα του Ιβάν είναι " + ivan.accessColor());
```

και αυτό που θα εκτυπωθεί στην οθόνη είναι το εξής:

```
Το χρώμα του Αζώρ είναι μαύρο
Το χρώμα του Ιβάν είναι άσπρο
```

Δηλαδή, η μεταβλητή `color` της κλάσης `Dog`, στην ουσία δεν έχει κάποια συγκεκριμένη τιμή. Συγκεκριμενοποιείται μόνο όταν φτιάξουμε κάποιο αντικείμενο της κλάσης `Dog` και έχει τόσα «αντίγραφα» όσα είναι τα αντικείμενα. Αφού συγκεκριμενοποιηθεί από τη δημιουργία αντικειμένου, το κάθε «αντίγραφό» της παίρνει συγκεκριμένη τιμή όταν καλέσουμε τη μέθοδο `setColor`.

Ένα άλλο παράδειγμα *(η κλάση Complex για μιγαδικούς αριθμούς)*

Στη Java δεν υπάρχει ειδικός τύπος για μεταβλητές *μιγαδικών αριθμών*. Αν θέλουμε να μπορούμε να διαχειριστούμε μιγαδικούς αριθμούς, θα πρέπει να φτιάξουμε ένα δικό μας "σύνθετο τύπο" που να περιέχει τα στοιχεία και τις πράξεις μιγαδικών αριθμών, δηλαδή μία **Κλάση**. Τα βασικά στοιχεία μιας τέτοιας κλάσης θα μπορούσαν να είναι τα εξής:

```
public class Complex
{
  private double real, imag; // το πραγματικό και το φανταστικό μέρος

  public void setValues(double x, double y)
  {
    real = x;
    imag = y;
  }

  public double getReal()
  {
    return real;
  }

  public double getImaginary()
  {
    return imag;
  }

  public String printComplex()
  {
    if (imag>0)
      return (real + " + " + imag + "i");
    else if (imag<0)
      return (real + " - " + (-imag) + "i");
    else
     return (real + ""); //προσθέτουμε κενό String για να μετατραπεί όλο σε String
  }
}
```

Μέχρι στιγμής, η κλάση για τους μιγαδικούς αριθμούς περιέχει τα εξής:

- μία μέθοδο για ανάθεση τιμών στο πραγματικό και στο φανταστικό μέρος ενός μιγαδικού (`setValues`)
- μία μέθοδο που δίνει πρόσβαση στην τιμή του πραγματικού μέρους ενός μιγαδικού (`getReal`)
- μία μέθοδο που δίνει πρόσβαση στην τιμή του φανταστικού μέρους ενός μιγαδικού (`getImaginary`)
- μία μέθοδο που επιστρέφει έναν μιγαδικό αριθμό στη μορφή $a + b\,i$ (`printComplex`)

Για την υλοποίηση άλλων ιδιοτήτων των μιγαδικών αριθμών, όπως π.χ. πράξεων μεταξύ μιγαδικών κτλ., απαιτούνται στοιχεία της Java που δεν έχουν διδαχθεί ακόμη. Επομένως, το παράδειγμα της κλάσης `Complex` θα συνεχιστεί σε επόμενο μάθημα. Με τα στοιχεία που έχει όμως μέχρι στιγμής η κλάση, μπορούμε να την χρησιμοποιήσουμε για να δίνουμε τιμές σε μιγαδικούς αριθμούς και να τους εμφανίζουμε στην οθόνη. Αυτό υλοποιείται π.χ., με την ακόλουθη *κλάση εφαρμογής*:

```
import java.util.*;
public class TestComplex
{
    public static void main(String [] args)
    {
        Scanner input = new Scanner(System.in);
        System.out.print("Δώσε το πραγματικό μέρος: ");
        double a = input.nextDouble();
        System.out.print("Δώσε το φανταστικό μέρος: ");
        double b = input.nextDouble();
        Complex c = new Complex();
        c.setValues(a, b);
        System.out.print("Ο μιγαδικός αριθμός είναι: ");
        System.out.println(c.printComplex());
    }
}
```

Παραδείγματα εκτέλεσης του προγράμματος:

```
Δώσε το πραγματικό μέρος: 2.4
Δώσε το φανταστικό μέρος: 4.6
Ο μιγαδικός αριθμός είναι: 2.4 + 4.6i

Δώσε το πραγματικό μέρος: 5
Δώσε το φανταστικό μέρος: -8.3
Ο μιγαδικός αριθμός είναι: 5.0 – 8.3i
```

Συνοψίζοντας, οι 4 γενικές περιπτώσεις μεθόδων (ως προς την είσοδο/έξοδο) και οι τρόποι κλήσης τους από μέθοδο της ίδιας κλάσης ή από μέθοδο άλλης κλάσης είναι οι εξής:

Μέθοδος στην κλάση MyClass	**Κλήση από τη MyClass**	**Κλήση από άλλη κλάση**

`public void method1()`
`{`
`  ...`
`}`
	`method1();`	`obj.method1();`

`public void method2(int x)`
`{`
`  ...`
`}`
	`int a = 5;`
	`method2(a);`
		`int b = 3;`
		`obj.method2(b);`

`public double method3()`
`{`
`  ...`
`}`
	`double x = method3();`	`double y = obj.method3();`

`public boolean method4(char c)`
`{`
`  ...`
`}`
	`char c = 'd';`
	`boolean b = method4(c);`
		`char d = 'a';`
		`boolean x = obj.method4(d);`

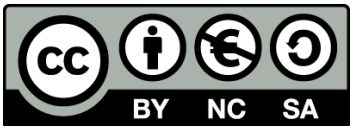
(Τα ονόματα και οι τιμές μεταβλητών καθώς και οι τύποι παραμέτρων εισόδου ή μεταβλητών επιστροφής των μεθόδων, είναι αυθαίρετα. ***Το `obj` είναι αντικείμενο της `MyClass`****. Επίσης, οι κλήσεις των μεθόδων της* `MyClass` *γίνονται προφανώς μέσα από άλλες μεθόδους.*

Σημειώματα

Σημείωμα Αναφοράς

Μιχάλης Δρακόπουλος. «Πληροφορική ΙΙ. Ενότητα 4: Αντικειμενοστραφής Προγραμματισμός (Μέθοδοι, Κλάσεις, Αντικείμενα)». Έκδοση: 1.0. Αθήνα 2014. Διαθέσιμο από τη δικτυακή διεύθυνση: http://opencourses.uoa.gr/courses/MATH106/.

Σημείωμα Αδειοδότησης



[1] http://creativecommons.org/licenses/by-nc-sa/4.0/

Ως **Μη Εμπορική** ορίζεται η χρήση:

- που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
- που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
- που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

Διατήρηση Σημειωμάτων

- Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:
- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.