Αντικειμενοστρεφής Προγραμματισμός

Παναγιώτης Αδαμίδης
adamidis@ihu.gr

Κληρονομικότητα

Όσο πιο σύντομα αρχίσετε να γράφετε κώδικα, τόσο πιο **πολύ χρόνο** θα χρειαστείτε για να τελειώσετε το πρόγραμμα.
Η προσεκτική σχεδίαση του προγράμματος πρέπει να προηγείται του κώδικα. Αυτό ισχύει ιδιαίτερα για αντικειμενοστρεφή προγράμματα.

Βασικές Έννοιες Α.Π.

- Ενθυλάκωση (Encapsulation)
- Αφαίρεση (Abstraction)
- Κληρονομικότητα (Inheritance)
- Πολυμορφισμός (Polymorphism)

Ενθυλάκωση

- Βασική έννοια: Κλάση (Class)
- Υλοποίηση: Αντικείμενο (Object) - στιγμιότυπο (instance) μιας κλάσης
- Χαρακτηριστικά/Ιδιότητες: Μεταβλητές (instance variables)
- Συμπεριφορά/Λειτουργία: Μέθοδοι (methods)
- Πρόσβαση: default, private, public, protected

Ενθυλάκωση

- Πεδία/χαρακτηριστικά `private` και πρόσβαση στα πεδία μέσω `public` μεθόδων. Η δήλωση ενός πεδίου ως private δεν επιτρέπει την πρόσβαση σε αυτό εκτός κλάσης, κρύβοντάς το μέσα στην κλάση (απόκρυψη δεδομένων – data hiding)
- Η πρόσβαση επιτρέπεται μόνο με συγκεκριμένο τρόπο που ορίζει ο προγραμματιστής.
- Κύριο όφελος: Δυνατότητα τροποποίησης του κώδικα της κλάσης (πχ. τύπος πεδίων) χωρίς να επηρεάζει τον κώδικα των χρηστών της κλάσης. Αυτό προσφέρει δυνατότητα συντήρησης, ευελιξία και επεκτασιμότητα στον κώδικα μας.

Αφαίρεση (Abstraction)

- Διαδικασία κατά την οποία αφαιρούνται κάποιες ιδιότητες από κάποια έννοια για γίνει πιο απλή.
- Μέσα από τη διαδικασία της αφαίρεσης, ένας προγραμματιστής κρύβει όλα τα χαρακτηριστικά των αντικειμένων τα οποία δεν είναι χρήσιμα στην εφαρμογή που υλοποιεί, προκειμένου να μειωθεί η πολυπλοκότητα και να αυξηθεί η αποτελεσματικότητα.

Αφαίρεση (Abstraction)

- Φυσική απόρροια της ενθυλάκωσης
- Η αφαίρεση επιτρέπει να δηλώσουμε ποιες θα είναι οι λειτουργίες αδιαφορώντας για την υλοποίησή τους.
- Η εφαρμογή της αφαίρεσης σημαίνει ότι κάθε αντικείμενο θα πρέπει να εμφανίζει **μόνο** ένα μηχανισμό χρήσης υψηλού επιπέδου.
- Αυτός ο μηχανισμός θα πρέπει να κρύβει τις λεπτομέρειες υλοποίησης και να αποκαλύπτει μόνο λειτουργίες σχετικές με άλλα αντικείμενα.
- Θα πρέπει να είναι εύκολος στη χρήση και να αλλάζει σπάνια.

Κληρονομικότητα (Inheritance)

- Επαναχρησιμοποίηση λογισμικού
- Κληρονομικότητα είναι η δημιουργία μιας νέας κλάσης (**υποκλάση-subclass**) από μια προϋπάρχουσα (**υπερκλάση-superclass**).
- Διαδικασία κατά την οποία ένα αντικείμενο αποκτά τις ιδιότητες κάποιου άλλου.
- Με τη χρήση της κληρονομικότητας η πληροφορία γίνεται διαχειρίσιμη σε μια ιεραρχική σειρά.
- Λέξεις κλειδιά: `extends`, `implements`

Κληρονομικότητα (Inheritance)

- Η κληρονομικότητα εκφράζει σχέσεις "is-a" μεταξύ δύο κλάσεων (→ αντικειμένων).
- Με χρήση κληρονομικότητας στις παραγόμενες κλάσεις (υποκλάσεις) επαναχρησιμοποιείται κώδικας των γονικών (υπερκλάσεις).
- Στιγμιότυπα των υποκλάσεων μπορούν να χρησιμοποιηθούν όπου απαιτούνται στιγμιότυπα των υπερκλάσεων.
- Στην Java, ή έννοια "is-a" βασίζεται στην κληρονομικότητα κλάσεων (using extends) or στην υλοποίηση διασυνδέσεων (using implements).
- Οι υποκλάσεις εξειδικεύουν τη συμπεριφορά των υπερκλάσεών τους.

Πολυμορφισμός

- Συναρτησιακός / διαδικαστικός προγραμματισμός: διαφορετικές συναρτήσεις με διαφορετικό όνομα για να κάνεις διαφορετικά πράγματα
- Πολυμορφισμός: μία αναφορά σε ένα αντικείμενο, πολλές μορφές. Μία μέθοδος μπορεί να έχει πολλές υλοποιήσεις (επομένως διαφορετική λειτουργία), ανάλογα με τον τύπο του αντικειμένου που την καλεί.
- Στον αντικειμενοστρεφή προγραμματισμό αυτό γίνεται όταν μια αναφορά υπερκλάσης χρησιμοποιείται για να αναφερθεί σε ένα αντικείμενο υποκλάσης.
- Στατικός (overloading) - Δυναμικός (overriding)

Ιεραρχία Τύπων

- Χρησιμοποιείται για να ορίσει οικογένειες τύπων οι οποίες αποτελούνται από ένα υπερτύπο και τους υποτύπους του. Η ιεραρχία μπορεί να επεκτείνεται σε πολλά επίπεδα.
- Μερικές οικογένειες τύπων χρησιμοποιούνται για να παρέχουν πολλαπλές υλοποιήσεις ενός τύπου: οι υποτύποι παρέχουν διαφορετικές υλοποιήσεις του υπερτύπου.
- Πιο γενικά, οι υποτύποι επεκτείνουν την συμπεριφορά του υπερτύπου τους (π.χ. παρέχοντας επιπλέον μεθόδους).
- Η αρχή της υποκατάστασης (αντικείμενα του υπερτύπου μπορούν να αντικατασταθούν από αντικείμενα του υποτύπου χωρίς να επηρεάσουν την ορθότητα και τη συμπεριφορά του προγράμματος) παρέχει αφαίρεση με περιγραφή για οικογένειες τύπων, απαιτώντας από τους υποτύπους να συμπεριφέρονται σύμφωνα με την περιγραφή του υπερτύπου τους.

Επαναχρησιμοποίηση λογισμικού

- Προσέγγιση στις διαδικαστικές γλώσσες (π.χ. C): Αντιγραφή κώδικα και μετατροπή του.
- Επαναχρησιμοποίηση κώδικα με δημιουργία νέων κλάσεων με μία από τις παρακάτω προσεγγίσεις:
- Σύνθεση (Composition): Δημιουργία αντικειμένων κλάσεων που ήδη υπάρχουν μέσα σε μία νέα κλάση. Εδώ χρησιμοποιείτε την λειτουργικότητα του κώδικα και όχι την δομή του (έχει_ένα, has_a).
- Κληρονομικότητα (Inheritance): Δημιουργία νέας κλάσης ως ένας τύπος μιας κλάσης που ήδη υπάρχει. Στην υπάρχουσα δομή μίας κλάσης προσθέτετε κώδικα χωρίς να τροποποιήσετε την προϋπάρχουσα κλάση. Είναι ένα από τα βασικά στοιχεία του αντικειμενοστραφούς προγραμματισμού (είναι_ένα, is_a).

Κλάση Object (...8)

Modifier and Type	Method and Description
protected Object	clone() Creates and returns a copy of this object.
boolean	equals(Object obj) Indicates whether some other object is "equal to" this one.
protected void	finalize() Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
Class<?>	getClass() Returns the runtime class of this Object.
int	hashCode() Returns a hash code value for the object.
void	notify() Wakes up a single thread that is waiting on this object's monitor.
void	notifyAll() Wakes up all threads that are waiting on this object's monitor.
String	toString() Returns a string representation of the object.
void	wait() Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.
void	wait(long timeout) Causes the current thread to wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
void	wait(long timeout, int nanos) Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

Κλάση Object (9...)

Modifier and Type	Method	Description
protected Object	clone()	Creates and returns a copy of this object.
boolean	equals(Object obj)	Indicates whether some other object is "equal to" this one.
protected void	finalize()	**Deprecated.** The finalization mechanism is inherently problematic.
Class<?>	getClass()	Returns the runtime class of this Object.
int	hashCode()	Returns a hash code value for the object.
void	notify()	Wakes up a single thread that is waiting on this object's monitor.
void	notifyAll()	Wakes up all threads that are waiting on this object's monitor.
String	toString()	Returns a string representation of the object.
void	wait()	Causes the current thread to wait until it is awakened, typically by being *notified* or *interrupted*.
void	wait(long timeout)	Causes the current thread to wait until it is awakened, typically by being *notified* or *interrupted*, or until a certain amount of real time has elapsed.
void	wait(long timeout, int nanos)	Causes the current thread to wait until it is awakened, typically by being *notified* or *interrupted*, or until a certain amount of real time has elapsed.

Κληρονομικότητα: Σύνταξη

- Η κληρονομικότητα είναι αναπόσπαστο μέρος της Java και του αντικειμενοστραφούς προγραμματισμού γενικότερα. Εάν δεν κληρονομείται άμεσα κάποια κλάση τότε κληρονομείται έμμεσα η κλάση **Object**.

```
class <υποκλάση> extends <υπερκλάση> {

        //σώμα κλάσης

}
```

- Στην Java, κάθε κλάση μπορεί να έχει μόνο μία υπερκλάση. Δεν υπάρχει πολλαπλή κληρονομικότητα κλάσεων.

Κληρονομικότητα: Ιδιότητες

- Μία κλάση που κληρονομείται (υπερκλάση) συνεχίζει να είναι αυτόνομη.
- Η παραγόμενη κλάση (υποκλάση) περιέχει αυτόματα (κληρονομεί) τις μεταβλητές και μεθόδους της προϋπάρχουσας (υπερκλάση). Η υποκλάση συμπεριλαμβάνει όλα τα μέλη της αντίστοιχης υπερ-κλάσης **εκτός** των private μελών και των δομητών.
- Ο προγραμματιστής μπορεί να προσθέσει στις υποκλάσεις νέες μεταβλητές και μεθόδους ή να μετατρέψει τις μεθόδους που κληρονομήθηκαν.
- Γενικά, αντικείμενα κάποιας υπερκλάσης δεν μπορούν να έχουν πρόσβαση στις υποκλάσεις της.

Κληρονομικότητα Παράδειγμα - 1

1

```
ΥΠΕΡΚΛΑΣΗ:
class Book{
    private int pages=1500;
    public void setPages(int pages){
          this.pages = pages;
    }
    public int getPages(){
          return pages;
    }
    public void pageMessage(){
          System.out.println("No of
                    pages: " + pages);
    }
} // class Book
```

Κληρονομικότητα Παράδειγμα - 1

2

```
ΥΠΟΚΛΑΣΗ:
class Dictionary extends Book{
    private int definitions=52500;
    public void defMessage() {
       System.out.println("No of
          definitions:" + definitions);
       System.out.println("Definitions per
          page" + definitions/getPages());
    }
} // class Dictionary
```

Κληρονομικότητα
Παράδειγμα - 1

3

```
class Words{
    public static void main(String[] args){
       Dictionary Oxford = new Dictionary();
       Oxford.pageMessage();
       Oxford.defMessage();
    }
} // class Words
```

```
General Output
--------------------Configuration: <Default>----------
No of pages: 1500
No of definitions: 52500
Definitions per page: 35

Process completed.
```

Κληρονομικότητα
Παράδειγμα - 2

1

```
class Cleanser {
   private String s = new String("Cleanser");
   public void append(String a) { s += a; }
   public void dilute() { append(" dilute()"); }
   public void apply() { append(" apply()"); }
   public void scrub() { append(" scrub()"); }
   public void print() { System.out.println(s); }
   public static void main(String[] args) {
       Cleanser x = new Cleanser();
       x.dilute(); x.apply(); x.scrub();
       x.print();
   }
}
```

Κληρονομικότητα
Παράδειγμα - 2

2

```
public class Detergent extends Cleanser {
  // Change a method:
  public void scrub() {
     append(" Detergent.scrub()");
     super.scrub(); // Call base-class version
  }
  // Add methods to the interface:
  public void foam() { append(" foam()"); }
  // Test the new class:
  public static void main(String[] args) {
     Detergent x = new Detergent();
     x.dilute();
     x.apply();
     x.scrub();
     x.foam();
     x.print();
     System.out.println("Testing base class:");
     Cleanser.main(args);
  }
}
```

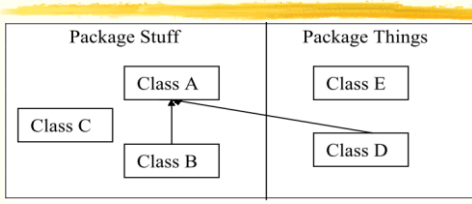
Κληρονομικότητα
Παράδειγμα - 2: Έξοδος

3

```
C:\Progs>java Cleanser
Cleanser dilute() apply() scrub()
```

```
C:\Progs>java Detergent
Cleanser dilute() apply() Detergent.scrub() scrub() foam()
Testing base class:
Cleanser dilute() apply() scrub()
```

Τροποποιητές Πρόσβασης (Modifiers)
Πρόσβαση σε μέλη κλάσεων



Πρόσβαση στα μέλη της κλάσης A

Στα	public	από	A,B,C,D,E
	private		A
	protected		A,B,C,D
	default		A,B,C

Πρόσβαση σε μέλη κλάσεων

	Ίδια κλάση	Ίδιο Πακέτο	Υποκλάση	Άλλη public κλάση
private	ΝΑΙ			
Default	ΝΑΙ	ΝΑΙ		
protected	ΝΑΙ	ΝΑΙ	ΝΑΙ	
public	ΝΑΙ	ΝΑΙ	ΝΑΙ	ΝΑΙ

Κληρονομικότητα Παράδειγμα - 3 1

```
class Point{
    int x,y;
    Point(){}     <----- Είναι απαραίτητη;
    Point(int x, int y){
        this.x = x;
        this.y = y;
    }
    double distance(int x,int y){
       int dx = this.x - x;
       int dy = this.y - y;
       return Math.sqrt(dx*dx+dy*dy);
    }
    double distance (Point p){
       return distance(p.x, p.y);
    }
 }
```

Κληρονομικότητα Παράδειγμα - 3 2

```
class Point3D extends Point{
    private int z;
    public Point3D(int x,int y,int z){
        super(x,y); // καλεί τον δομητή
                    // Point(x,y)
        this.z=z;
    }
    public Point3D(){
        this(-1,-1,-1);
    }
}
```

- Οι δομητές δεν κληρονομούνται, αν και είναι public. Η χρήση τους κατά την κληρονομικότητα γίνεται με το keyword **super**.

Κληρονομικότητα Παράδειγμα - 3 3

```
class PointDistance{
  public static void main(String args[]){
     Point p1=new Point(0,0);
     Point p2=new Point(30,40);
     System.out.println("p1="+p1.x+","+p1.y);
     System.out.println("p2="+p2.x+","+p2.y);
     System.out.println("p1.distance(p2)= " +
                          p1.distance(p2));
     System.out.println("p1.distance(60,80)= "
                        + p1.distance(60,80));
   }
} // class PointDist
```

Κληρονομικότητα Παράδειγμα - 4 1

```
class Mammal {
    private int age;
    private int weight;
    public Mammal() {}
    public Mammal(int a, int w) { age=a; weight=w; }
    public void speak() {
        System.out.println ("Mammal sound");
    }
    public void sleep() {
        System.out.println ("zzzZZZ");
    }
    public int getAge() { return age; }
    public void setAge(int age) { this.age = age; }
    public int getWeight() { return weight; }
    public void setWeight(int w) { weight = w; }
}
```

Κληρονομικότητα Παράδειγμα - 4 2

```
class Cat extends Mammal {
    public void purr() {
        System.out.println ("Purrrrr...");
    }
}
```

Κληρονομικότητα Παράδειγμα - 4 3

```
class Dog extends Mammal {
    enum Breed { Yorkshire, Bulldog,  Labrador,
                                 Shetland, Doberman };
    private Breed breed;
    public Breed getBreed() { return breed;}
    public void setBreed (Breed b) { breed = b; }
    public Dog() {}
    public Dog (int age, int weight, Breed breed){
        super(age,weight);
        this.breed = breed;
    }
    public void wagTail() {
        System.out.println ("Tail wagging...");
    }
    public void askForFood() {
        System.out.println ("Asking for food...");
    }
}
```

Κληρονομικότητα Παράδειγμα - 4 4

```
class Example_4 {
   public static void main(String[] args) {
      Dog rex = new Dog(3, 15, Dog.Breed.Labrador);
      //rex.breed = Dog.Breed.Labrador;
      System.out.println("Rex is "+ rex.getAge() +
                                    " years old");
     System.out.println("Rex is a " + rex.getBreed());
      rex.speak();
      rex.wagTail();

      Cat fluffy = new Cat();
      System.out.println("Flufy weighs "+
                     fluffy.getWeight() + " kilos");
      fluffy.sleep();
      fluffy.purr();
    }
}
```

Κληρονομικότητα Παράδειγμα - 4 5

ΕΞΟΔΟΣ

```
Rex is 3 years old
Rex is a Labrador
Mammal sound
Tail wagging...
Flufy weighs 0kilos
zzzZZZ
Purrrrr...
```

Αρχικοποίηση Υπερκλάσης

- Στην κληρονομικότητα δεν αντιγράφεται απλώς η διασύνδεση/συμπεριφορά της υπερκλάσης. Όταν δημιουργείται ένα αντικείμενο της παραγόμενης κλάσης, περιέχει ένα «υπο-αντικείμενο» της βασικής κλάσης. Αυτό το «υπο-αντικείμενο» είναι το ίδιο όπως και αν είχε δημιουργηθεί από την υπερκλάση.
- Η σωστή αρχικοποίηση του υπο-αντικειμένου αυτού είναι ουσιώδης. Η σωστή αρχικοποίηση γίνεται στον δομητή της υποκλάσης καλώντας τον δομητή της υπερκλάσης, ο οποίος έχει την κατάλληλη γνώση και το δικαίωμα αρχικοποίησης της υπερκλάσης.
- Η Java αυτόματα καλεί τον δομητή της υπερκλάσης κατά την κλήση του δομητή της υποκλάσης.

Αρχικοποίηση Υπερκλάσης: Παράδειγμα 1

```
class Art {
  Art() {System.out.println("Art constructor"); }
}
class Drawing extends Art {
  Drawing() {
    System.out.println("Drawing constructor");
  }
}
public class Cartoon extends Drawing {
  Cartoon() {
    System.out.println("Cartoon constructor");
  }
  public static void main(String[] args) {
    Cartoon x = new Cartoon();
  }
}
```

Αρχικοποίηση Υπερκλάσης: Παράδειγμα Έξοδος 2

```
C:\Panos\Progs>java Cartoon
Art constructor
Drawing constructor
Cartoon constructor
```

Δομητές με ορίσματα

- Το προηγούμενο παράδειγμα χρησιμοποιεί τους προκαθορισμένους (default) δομητές, δηλ. δομητές χωρίς ορίσματα. Ο μεταγλωττιστής μπορεί εύκολα να τους καλέσει χωρίς να υπάρχει αμφισβήτηση για τα ορίσματα που θα χρησιμοποιήσει.
- Εάν θέλετε να καλέσετε ένα δομητή της υπερκλάσης που έχει ορίσματα, τότε πρέπει να χρησιμοποιηθεί η λέξη κλειδί **super** με τα κατάλληλα ορίσματα και αυτό πρέπει να είναι η πρώτη εντολή του δομητή της υποκλάσης.

Δομητές με ορίσματα: Παράδειγμα 1

```
class Game {
    Game(int i) {
        System.out.println("Game constructor"+i);
    }
}

class BoardGame extends Game {
   BoardGame(int i) {
      super(i);
      System.out.println("BoardGame constructor");
   }
}
```

συνεχίζεται

Δομητές με ορίσματα: Παράδειγμα 2

```
public class Chess extends BoardGame {
   Chess() {
      super(11);
      System.out.println("Chess constructor");
   }
   public static void main(String[] args) {
     Chess x = new Chess();
   }
}
```

```
C:\Panos\Progs>java Chess
Game constructor
BoardGame constructor
Chess constructor
```

Δομητές με ορίσματα: Παράδειγμα - 2 1

```
class TwoDShape {
  private double width;
  private double height;
  TwoDShape() { width=height=0.0; }
  TwoDShape(double w, double h) {
     width=w;      height=h;   }
  TwoDShape(double x) { width=height=x; }
  //Accessor methods
  double getWidth() { return width; }
  double getHeight() {return height; }
  void setWidth(double w) { width=w; }
  void setHeight(double h) { height=h; }
  void showDim() {
     System.out.println("Width and height are: "+width+
                        " and "+height);
  }
}
```

συνεχίζεται

Δομητές με ορίσματα: Παράδειγμα - 2 2

```
class Triangle extends TwoDShape {
    private String style;
    Triangle() {
        super();        style="null";
    }
    Triangle(String s, double w, double h) {
        super(w,h);      style=s;
    }
    Triangle(double x) {
        super(x);      style="isosceles";
    }
    double area() {
        return getWidth()*getHeight()/2;
    }
    void showStyle() {
        System.out.println("Triangle is "+style);
    }
}
```

συνεχίζεται

Δομητές με ορίσματα: Παράδειγμα - 2 3

```
class Shapes5 {
   public static void main(String[] args) {
       Triangle t1 = new Triangle();
       Triangle t2 = new Triangle("right", 8.0, 12.0);
       Triangle t3 = new Triangle(4.0);
       System.out.println("Info for t1");
       t1.showStyle();
       t1.showDim();
       System.out.println("Area is "+t1.area());
       System.out.println("Info for t2");
       t2.showStyle();
       t2.showDim();
       System.out.println("Area is "+t2.area());
       System.out.println("Info for t3");
       t3.showStyle();
       t3.showDim();
       System.out.println("Area is "+t3.area());
   }
}
```

Δομητές με ορίσματα: Παράδειγμα - 2 Έξοδος 4

```
C:\Progs>java Shapes5
Info for t1
Triangle is null
Width and height are: 0.0 and 0.0
Area is 0.0

Info for t2
Triangle is right
Width and height are: 8.0 and 12.0
Area is 48.0

Info for t3
Triangle is isosceles
Width and height are: 4.0 and 4.0
Area is 8.0
```

Πρόσβαση σε μέλη υπερκλάσης

- Μία άλλη μορφή της super μας επιτρέπει να έχουμε πρόσβαση σε μέλη της υπερκλάσης.
- Σύνταξη:

```
super.μέλος
```

όπου `μέλος` μπορεί να είναι είτε μία μέθοδος είτε μία μεταβλητή στιγμιότυπου.

- Η μορφή αυτή της super χρησιμοποιείται συνήθως στις περιπτώσεις κατά τις οποίες τα ονόματα των μελών μιας υποκλάσης αποκρύπτουν τα μέλη που έχουν το ίδιο όνομα με την υπερκλάση.

Πρόσβαση σε μέλη υπερκλάσης: Παράδειγμα

```
class A { int i; }
class B extends A {
    int i;
    B(int a, int b) {
        super.i=a;
        i=b;
    }
    void show() {
        System.out.println("i in superclass: "+super.i);
        System.out.println("i in subclass: "+i);
    }
}
class UseSuper {
    public static void main (String[] args) {
        B subOb = new B(1,2);
        subOb.show();
    }
}
```

```
run:
i in superclass: 1
i in subclass: 2
```

Ερωτήσεις;