



Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών  
Τμήμα Πληροφορικής & Τηλεπικοινωνιών

# Τεχνολογία Λογισμικού

8ο Εξάμηνο 2019-20

**Αντικειμενοστρεφής προγραμματισμός (επανάληψη)**

Δρ. Κώστας Σαΐδης ([saiko@di.uoa.gr](mailto:saiko@di.uoa.gr))

# Αντικειμενοστρεφής προγραμματισμός

- Επανάληψη
- Παραδείγματα σε Java και Javascript

# Αντικείμενα

- Ενθυλακώνουν (encapsulate)
  - Κατάσταση (state)
    - δεδομένα που τηρούνται σε πεδία
  - Συμπεριφορά (behavior)
    - λειτουργίες που τηρούνται σε μεθόδους (behavior)
- Στιγμιοτύπιση (instantiation)
  - Μέσω κατασκευαστών (constructors)
- Αυτο-αναφορά (this, self)
- Ανταλλαγή μηνυμάτων (message passing)

# Βασικές έννοιες

- Συνάθροιση (aggregation)
- Σύνθεση (composition)
- Κληρονομικότητα (inheritance)
  - Με βάση κλάσεις / διεπαφές (classes / interfaces)
  - Με βάση πρωτότυπα (prototypes)
- Δυναμική αποστολή μηνυμάτων (dynamic method dispatch)
- Αργή δέσμευση (late binding)
- Πολυμορφισμός

# Βασικές αρχές αφαίρεσης (abstraction principles)

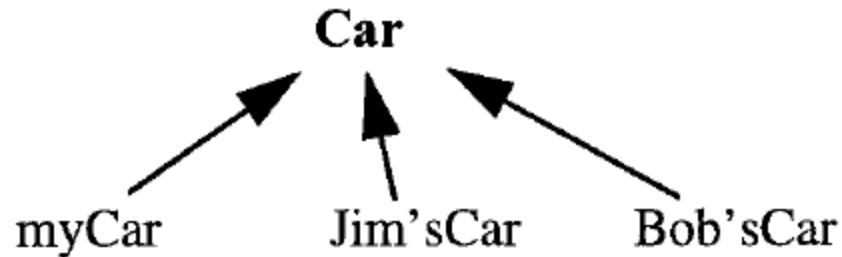
Το σήμειο που το --αντικειμενοστρεφές μοντέλο--  
συνδέεται με την εννοιολογική μοντελοποίηση (conceptual  
modeling) και την αναπαράσταση γνώσης (knowledge  
representation)

# Λίστα αναγνωσμάτων

Antero Taivalsaari, "On the notion of inheritance", ACM Computing Surveys, Vol. 28, No 3, September 1996.

# I. Classification - Instantiation

## Classification vs. instantiation



# Classification - Instantiation

- Σχέση του αντικειμένου με την κλάση του και αντίστροφα.
- Όλα τα αντικείμενα / στιγμιότυπα μιας κλάσης μοιράζονται κοινά και ομοιόμορφα χαρακτηριστικά.
- Η κλάση είναι το intensional abstraction όλων των δυνατών της αντικειμένων.



# Παράδειγμα (Java)

```
class Point {
    private int x;
    private int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public int getX() { return x; }

    public int getY() { return y; }

    public void setX(int x) { this.x = x; }

    public void setY(int y) { this.y = y; }

    public void addToX(int num) { this.x += num; }

    public void addToY(int num) { this.y += num; }
}
```

# Χρήση

```
Point p1 = new Point(0, 0); //στιγμιότυπιση  
p1.addToX(10); //αποστολή μηνύματος (επίκληση μεθόδου)  
Point p2 = new Point(10, 0);  
assert(p1.getX() == p2.getX());  
assert(p1 instanceof Point);  
assert(p2 instanceof Point);
```

# Παράδειγμα (Javascript < ES6)

```
var Point = function(x, y) { //constructor
  this.x = x;
  this.y = y;
  this.getX = function() { return this.x; }
  this.getY = function() { return this.y; }
  this.setX = function(x) { this.x = x; }
  this.setY = function(y) { this.y = y; }
  this.addToX = function(num) { this.x += num; }
  this.addToY = function(num) { this.y += num; }
}
```

# Καλύτερο Παράδειγμα (Javascript < ES6)

```
var Point = function(x, y) { //constructor
  this.x = x;
  this.y = y;
}

Point.prototype.getX = function() { return this.x; }
Point.prototype.getY = function() { return this.y; }
Point.prototype.setX = function(x) { this.x = x; }
Point.prototype.setY = function(y) { this.y = y; }
Point.prototype.addToX = function(num) { this.x += num; }
Point.prototype.addToY = function(num) { this.y += num; }
```

# Παράδειγμα (Javascript $\geq$ ES6)

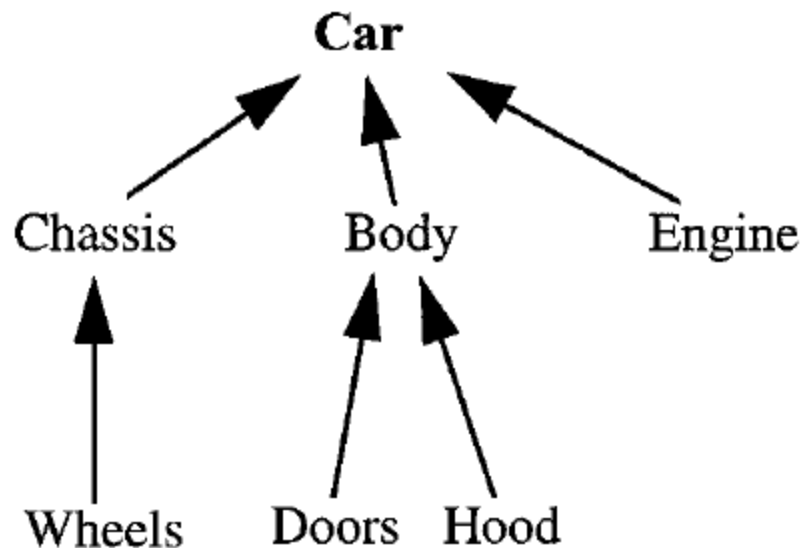
```
class Point {  
  
  constructor(x, y) {  
    this.x = x;  
    this.y = y;  
  }  
  
  get x() { return this.x; }  
  
  get y() { return this.y; }  
  
  set x(x) { this.x = x; }  
  
  set y(y) { this.y = y; }  
  
  addToX(num) { this.x += num; }  
  
  addToY(num) { this.y += num; }  
}
```

# Χρήση

```
var p1 = new Point(0, 0);  
p1.addToX(10);  
var p2 = new Point(10, 0);  
assert p1.getX() == p2.getX();  
assert (p1 instanceof Point);  
assert (p2 instanceof Point);
```

## 2. Aggregation - Decomposition

### Aggregation vs. decomposition



# Aggregation - Decomposition

- Συνάθροιση (ή σύνθεση) επιμέρους εννοιών για τη σύσταση μιας νέας ξεχωριστής έννοιας.
- Σχέσεις μέρους-όλου (part-of).



# Παράδειγμα (Java)

Ένα αντικείμενο περιέχει (συντίθεται) από άλλο αντικείμενα

```
class Line {  
    private Point first;  
    private Point second;  
  
    public Line(Point first, Point second) {  
        this.first = first;  
        this.second = second;  
    }  
  
    Point getFirstPoint() { return first; }  
  
    Point getSecondPoint() { return second; }  
}
```

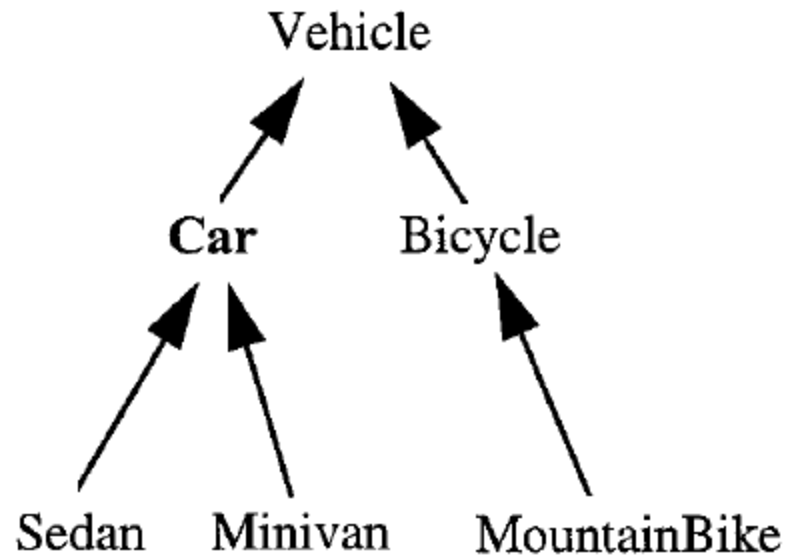
# Παράδειγμα (ES6)

Ένα αντικείμενο περιέχει (συντίθεται) από άλλο αντικείμενα

```
class Line {  
  
  constructor(first, second) { //Points  
    this.first = first;  
    this.second = second;  
  }  
  
  getFirstPoint() { return this.first; }  
  
  getSecondPoint() { return this.second; }  
}
```

# 3. Generalization - Specialization

## Generalization vs. specialization



# Generalization - Specialization

- Σχέση μεταξύ κλάσεων.
- Η γενική κλάση συγκεντρώνει τα κοινά στοιχεία όλων των εξειδικεύσεών της.
- Η κληρονομικότητα είναι ο κατεξοχήν μηχανισμός υλοποίησης της εξειδίκευσης.

# Κληρονομικότητα

- Ένα αντικείμενο κληρονομεί τα πεδία ή/και τις μεθόδους των "προγόνων" του
- Στην πράξη έχουμε:
  - Κληρονομικότητα για εξειδίκευση (specialization)
  - Κληρονομικότητα για επαναχρησιμοποίηση (reuse)

# Παράδειγμα (Java)

```
class Arrow extends Line {  
  
    enum Direction{FIRST_TO_SECOND, SECOND_TO_FIRST};  
  
    private Direction d;  
  
    public Arrow(Point p1, Point p2, Direction d) {  
        super(p1, p2);  
        this.d = d;  
    }  
  
    public void toggleDirection() {  
        if (d == Direction.FIRST_TO_SECOND)  
            d = Direction.SECOND_TO_FIRST;  
        else  
            d = Direction.FIRST_TO_SECOND;  
    }  
}
```

# Χρήση

```
Point p1 = new Point(0, 0);  
Point p2 = new Point(10, 10);  
Arrow a = new Arrow(p1, p2, Arrow.Direction.FIRST_TO_SECOND);  
assert(a instanceof Arrow); //Προφανώς  
assert(a instanceof Line); //Επίσης -- πολυμορφισμός  
assert(a.getFirstPoint().getX() == 0); //Κληρονομικότητα
```

# Παράδειγμα (Javascript < ES6)

```
var Line = function() { //Line constructor
  ...
}

var Direction = {
  FIRST_TO_SECOND: 1,
  SECOND_TO_FIRST: 2
}

var Arrow = function(p1, p2, d) {
  Line.call(this, p1, p2); //call the Line constructor
  this.d = d;
}

Arrow.prototype = new Line(); // "Inherit" from Line
Arrow.prototype.constructor = Arrow; // Just to make sure
Arrow.prototype.toggleDirection = function() {
  if (this.d == Direction.FIRST_TO_SECOND)
    this.d = Direction.SECOND_TO_FIRST;
  else
    this.d = Direction.FIRST_TO_SECOND;
}
```



# Κληρονομικότητα στη Javascript

- Δεν υπάρχουν κλάσεις, υπάρχουν μόνο αντικείμενα
- Τα οποία συνδέονται μεταξύ τους μέσω "αλυσίδας πρωτοτύπων" (prototypes chain)
- Η γλώσσα για να "βρει" ένα χαρακτηριστικό ενός αντικειμένου:
  - "κοιτάζει" στο ίδιο το αντικείμενο
  - αν δεν το βρεί, "κοιτάζει" στο αντικείμενο/πρωτότυπο αυτού
  - συνεχίζει αναδρομικά μέχρι είτε να το βρει είτε να καταλήξει σε αντικείμενο με `null` πρωτότυπο

# Κλάσεις στη Javascript $\geq$ ES6

```
class Line {
  ...
}

var Direction = {
  FIRST_TO_SECOND: 1,
  SECOND_TO_FIRST: 2
}

class Arrow extends Line {

  constructor(p1, p2, d) {
    super(p1, p2);
    this.d = d;
  }

  toggleDirection() {
    if (this.d == Direction.FIRST_TO_SECOND)
      this.d = Direction.SECOND_TO_FIRST;
    else
      this.d = Direction.FIRST_TO_SECOND;
  }
}
```

Προσοχή: δεν είναι "πραγματικές" κλάσεις, αλλά μια συντακτική ευκολία για πιο εύκολη χρήση της αλυσίδας πρωτοτύπων

# Χρήση

```
var p1 = new Point(0, 0);  
var p2 = new Point(10, 10);  
var a = new Arrow(p1, p2, Direction.FIRST_TO_SECOND);  
assert(a instanceof Arrow);  
assert(a instanceof Line);  
assert(a.getFirstPoint().getX() == 0);
```

# Private state/behavior

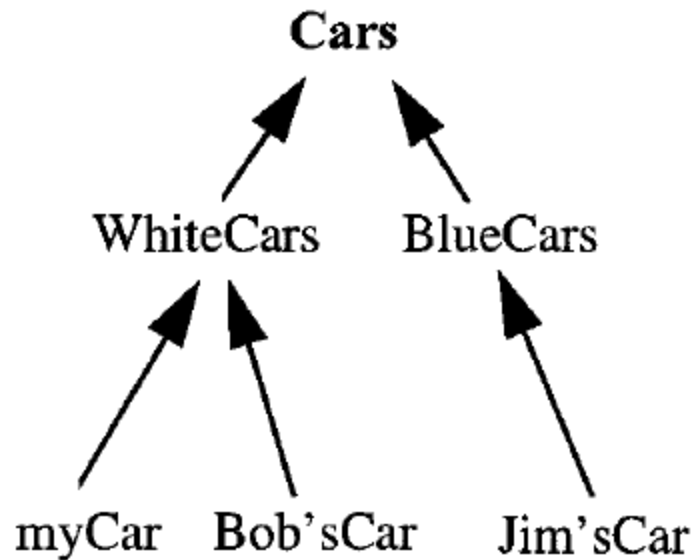
- Στο (απλό) παράδειγμά μας:
  - Η κλάση Point ενθυλακώνει δύο ακεραίους
  - Η κλάση Line ενθυλακώνει δύο Point αντικείμενα
  - Η κλάση Arrow ενθυλακώνει δύο Point αντικείμενα και μια διεύθυνση
- Μηχανισμός απόκρυψης πληροφορίας (information hiding)

# Private state σε Javascript

```
var Line = function(x, y) {  
  var state = {  
    x:x,  
    y:y  
  };  
  
  this.getX = function() {  
    return state.x  
  }  
  
  ...  
};
```

# 4. Grouping - Individualization

## Grouping vs. individualization



# Ομαδοποίηση - Διαχωρισμός

- Ομαδοποίηση αντικειμένων με βάση κάποιο extensional και όχι intensional χαρακτηριστικό.
- Παραδείγματα:
  - Αγαπημένα του χρήστη (user favorites)
  - Πιο πρόσφατα / πιο δημοφιλή
  - Αποτελέσματα μιας αναζήτησης

# Πολυμορφισμός

Παροχή μιας κοινής διεπαφής για αντικείμενα διαφορετικών τύπων

ή

Ένα αντικείμενο μπορεί να έχει πολλούς τύπους (πολλές συμπεριφορές)



# Είδη πολυμορφισμού

## Universal polymorphism

- Parametric
- Inclusion

## Ad-hoc

- Overloading
- Coercion

# Λίστα αναγνωσμάτων

Luca Cardelli, Peter Wegner, "On Understanding Types, Data Abstraction, and Polymorphism", ACM Computing Surveys, Vol 17 n. 4, pp 471-522, December 1985.

# Ad-hoc πολυμορφισμός (Overloading)

## Operator overloading

```
int x = 3 + 5;  
String s = name + " " + surname;
```

## Method overloading

```
class Foo {  
    void doSomething(A a) { ... }  
    void doSomething(A a, B b) { ... }  
}  
  
class Bar extends Foo {  
    void doSomething(C c, D d) { ... }  
}
```

# Ad-hoc πολυμορφισμός (Coercion)

Type coercion

```
double x = 1;
//The int constant is converted to double automatically

double avg, sum;
int count;
...
avg = sum / count;
//The int count is converted to double automatically
//before applying the division
```

# Παραμετρικός πολυμορφισμός

## Generics

```
class Cache<K, V> {  
    private final Map<K, V> cache = new HashMap<>();  
    public synchronized void put(K key, V value) {  
        cache.put(key, value);  
    }  
  
    public synchronized V get(K key) {  
        return cache.get(key);  
    }  
}
```

```
Cache<String, Line> lineLabels = new Cache<>();  
cache.put("First line", someLine);  
cache.put("Second line", someOtherLine);
```

# Πολυμορφισμός υπο-τύπων (inclusion polymorphism)

```
interface Shape {
    double getArea();
}

class Rectangle implements Shape {
    private double width;
    private double height;
    public double getArea() {
        return width * height;
    }
}

class Circle implements Shape {
    private double radius;
    public double getArea() {
        return Math.PI * radius * radius;
    }
}
```

```
class AreaPrinter {  
    public static void print(Shape s) {  
        System.out.println(s.getArea());  
    }  
}
```

```
Rectangle r = new Rectangle(2.0, 3.0);  
Circle c = new Circle(1.0);
```

```
AreaPrinter.print(r); // 6.0  
AreaPrinter.print(c); // 3.14
```



# Δυναμική αποστολή μηνυμάτων και αργή δέσμευση

```
interface Computation {
    String getName();
    void compute();
}

abstract class ComputationBase implements Computation {
    public String getName() {
        return getClass().getName();
    }
}
```

```
class SimpleComputation extends ComputationBase {
    public void compute() {
        System.out.println("Done computing");
    }
}

class SimpleComputation2 extends SimpleComputation {
    public String getName() {
        return "Simple2"
    }
}
```

```
class ListOfComputations extends ComputationBase {
    private final List<Computation> computations;

    public ListOfComputations(Computation... computations) {
        this.computations = Arrays.asList(computations);
    }

    public void compute() {
        for(Computation c: computations) {
            System.out.println("Computing: " + c.getName());
            c.compute();
        }
    }
}
```

```
Computation c1 = new SimpleComputation();  
Computation c2 = new SimpleComputation2();  
Computation list = new ListOfComputations(c1, c2);  
System.out.println(list.getName());  
list.compute();
```

```
//Output
```

```
ListOfComputations
```

```
Computing: SimpleComputation
```

```
Done computing
```

```
Computing: Simple2
```

```
Done computing
```