04 Αντικειμενοστραφής ανάλυση και σχεδιασμός

Τεχνολογία Λογισμικού

Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο

Χειμερινό εξάμηνο 2017-18

Δρ. Κώστας Σαΐδης (saiko@di.uoa.gr)

Περιεχόμενα

1. Αντικειμενοστράφεια (με παραδείγματα σε Java, Javascript)
2. Εισαγωγή στη UML*
3. Εισαγωγή στα πρότυπα σχεδίασης*

*** Θα επανέλθουμε αναλυτικότερα σε επόμενες διαλέξεις**

1. Αντικειμενοστράφεια

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

- Ενθυλάκωση (encapsulation)
- Στυγμιοτύπιση (instantiation)
- Κληρονομικότητα (inheritance)
  - Με βάση κλάσεις / διεπαφές (classes / interfaces)
  - Με βάση πρωτότυπα (prototypes)
- Δυναμική αποστολή μηνυμάτων (dynamic method dispatch)
- Αργή δέσμευση (late binding)
- Σύνθεση
- Πολυμορφισμός

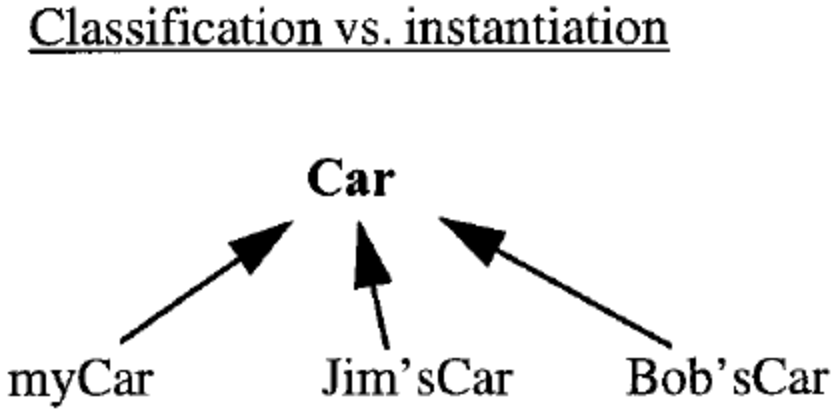
Βασικές αρχές αφαίρεσης (abstraction principles)

Το σήμειο που το **αντικειμενοστραφές μοντέλο** συνδέεται με την εννοιολογική μοντελοποίηση (conceptual modeling) και την αναπαράσταση γνώσης (knowledge represenation)

Λίστα αναγνωσμάτων

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

I. Classification - Instantiation



A. Taivalsaari

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); //στιγμιοτύπιση
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) { //constuctor
  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) { //constuctor
  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; }
```

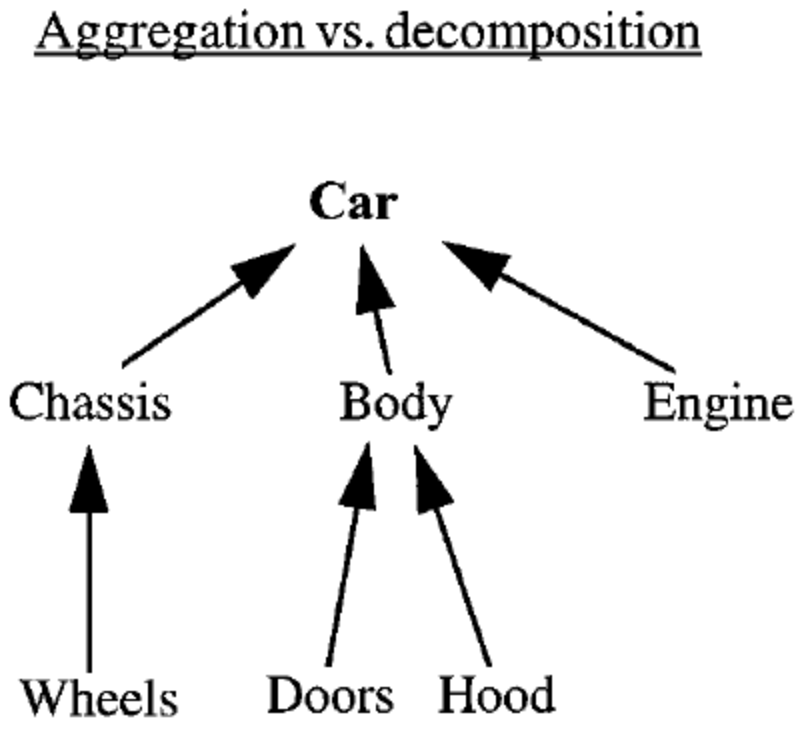
Παράδειγμα (Javascript >= ES6)

```
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



A. Taivalsaari

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

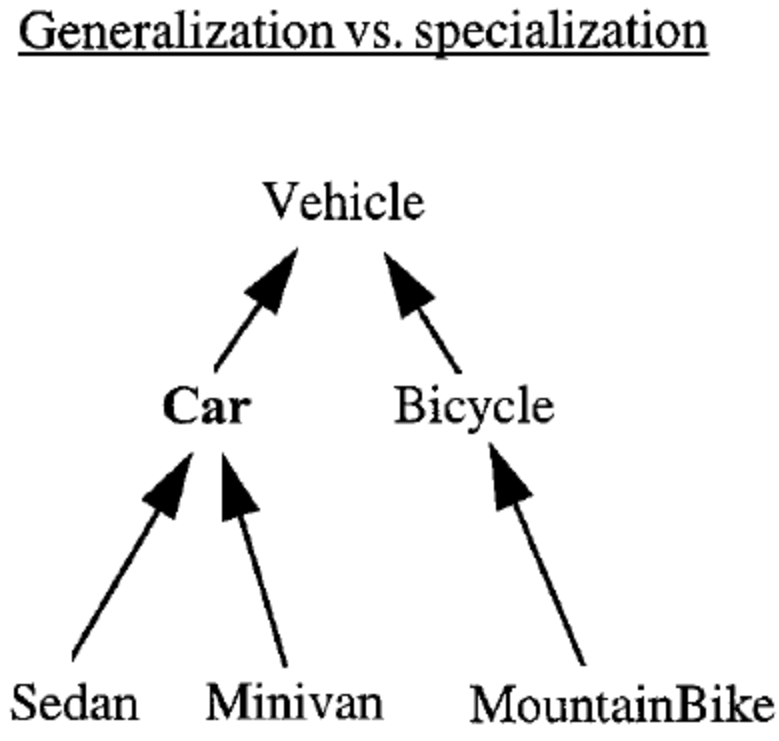
```
class Line {
  private Point first;
  private Point second;

  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



A. Taivalsaari

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;
}
```

Παράδειγμα (Javascript >= 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;
  }
```

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);
```

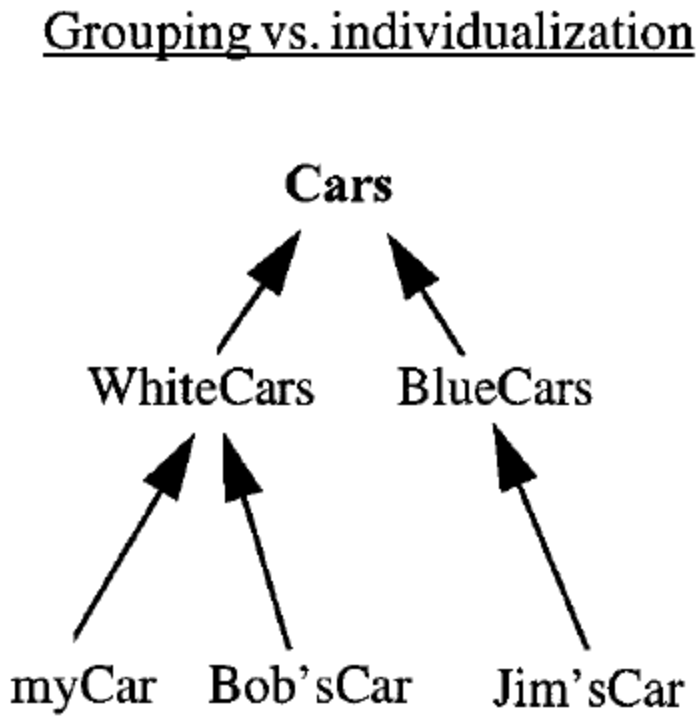
Private state/behavior

- 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



A. Taivalsaari

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 coverted 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
```

2. Εισαγωγή στη γλώσσα UML

Unified Modeling Language (UML)

- Γλώσσα μοντελοποίησης γενικής χρήσης που στοχεύει στην παροχή ενός καθιερωμένου τρόπου οπτικοποίησης και επικοινωνίας του σχεδιασμού ενός συστήματος.
- ISO standard 19501:2005
- Ενοποιημένη μοντελοποίηση τόσο των απαιτήσεων όσο και του σχεδιασμού
- Τελευτία έκδοση 2.5 (2015)

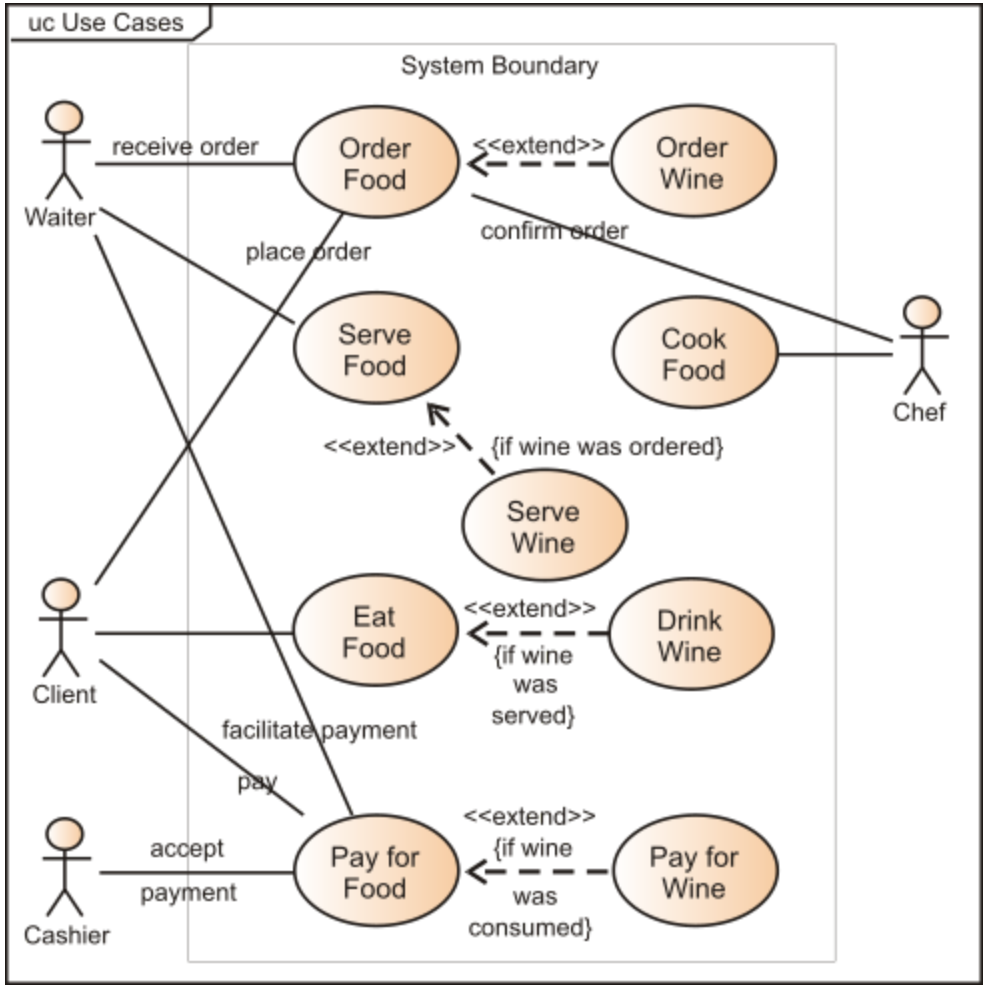
Μοντελοποίηση συμπεριφοράς

- Διαγράμματα περιπτώσεων χρήσης (Use-case diagrams)
- Διαγράμματα δραστηριότητας (Activity diagrams)
- Διαγράμματα μηχανών κατάστασης (State machine diagrams)
- Διαγράμματα ακολουθίας (Sequence diagrams)
- Διαγράμματα επικοινωνίας (Communication diagrams)
- Διαγράμματα χρονισμού (Timing diagrams)

Μοντελοποίηση δομής

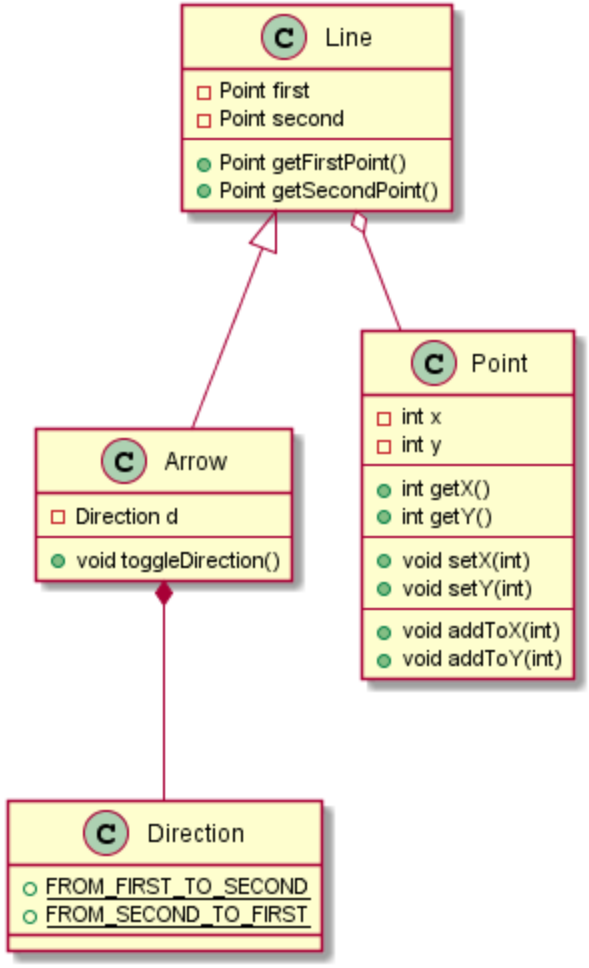
- Διαγράμματα κλάσεων (Class diagrams)
- Διαγράμματα συστατικών (Component diagrams)
- Διαγράμματα πακέτων (Package diagrams)
- Διαγράμματα "παράταξης / εγκατάστασης" (Deployment diagrams)
- Διαγράμματα σύνθετης δομής (Composite structure diagrams)

Διάγραμμα περίπτωσης χρήσης

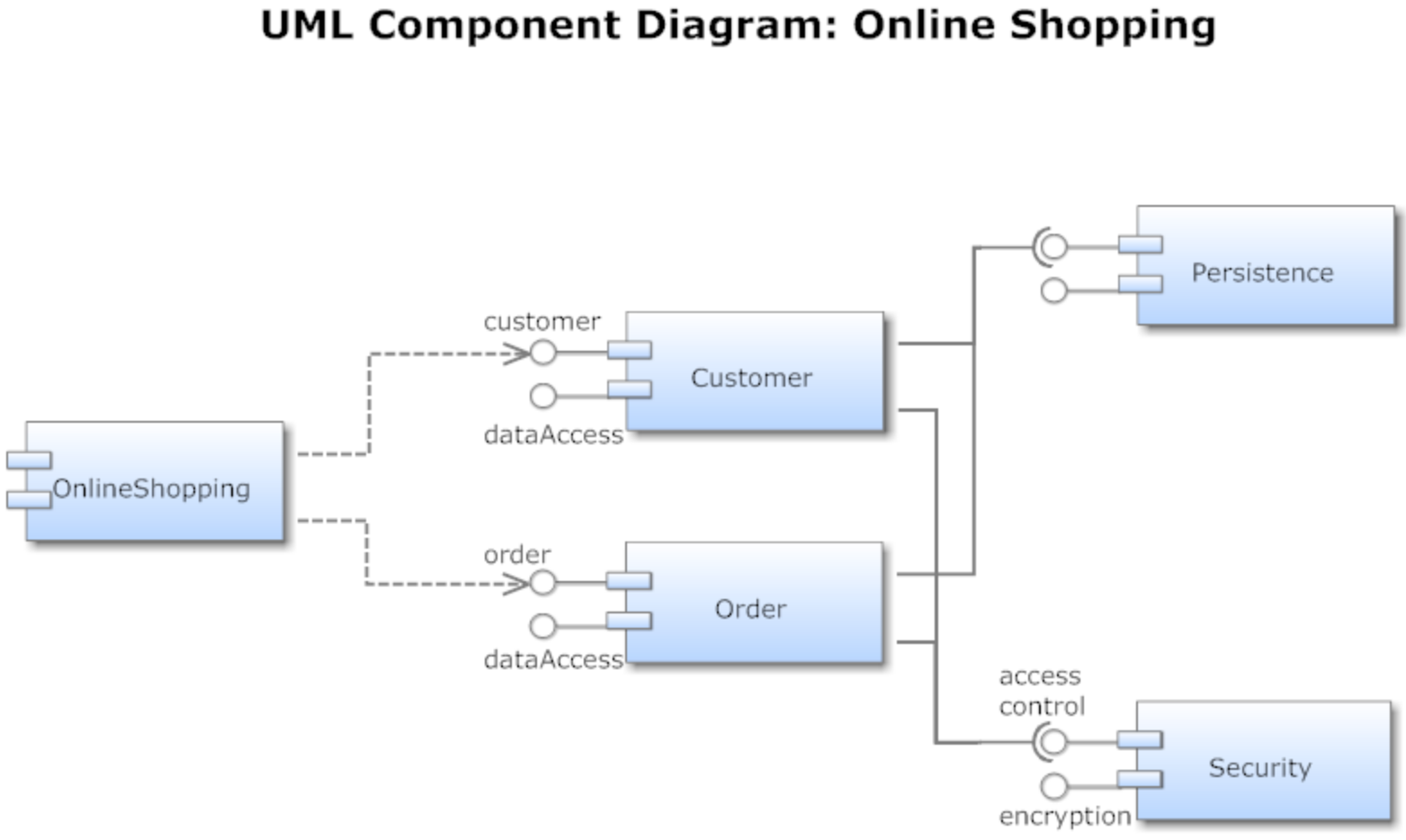


By Kishorekumar 62, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=7880320

Διάγραμμα κλάσεων

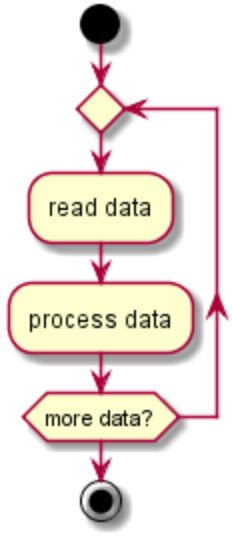
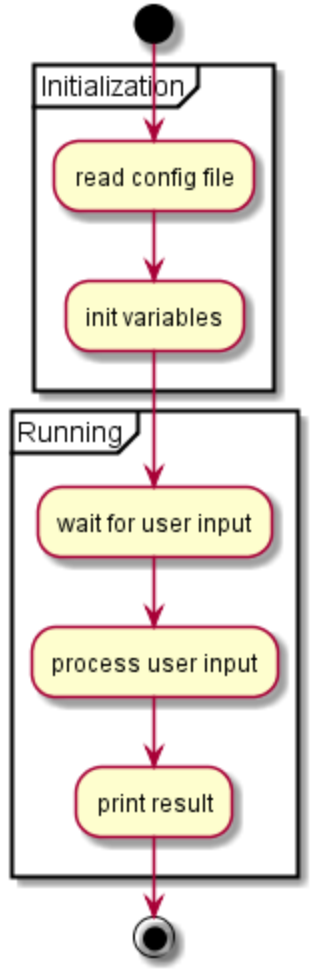


Διάγραμμα συστατικών

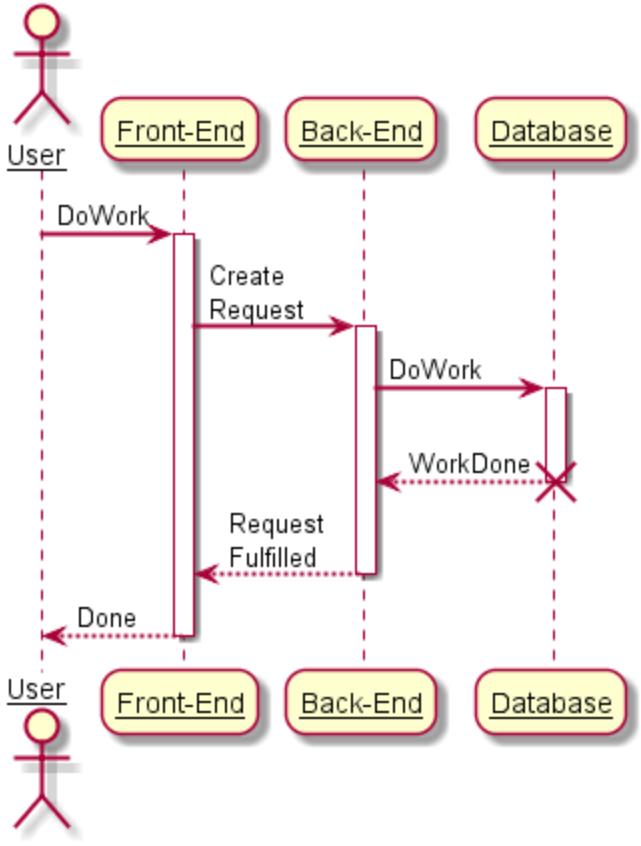


smartdraw.com

Διάγραμμα δραστηριότητας

1ο Παράδειγμα	2ο Παράδειγμα
	

Διάγραμμα ακολουθίας



Θα επανέλθουμε σε επόμενη διάλεξη…

3. Εισαγωγή στα πρότυπα σχεδίασης

Πρότυπο σχεδίασης (Design pattern)

- Μια καλή πρακτική για την αντιμετώπιση ενός σχεδιαστικού προβλήματος στο λογισμικό.

Προσοχή

- Ένας τρόπος δόμησης/διάρθωσης του κώδικα για την επίλυση ενός συγκεκριμένου προβλήματος (όχι ο κώδικας καθ' αυτός).
- Η υπερβολική χρήση των προτύπων επιφέρει περισσότερα προβλήματα απ' όσα λύνει (αύξηση πολυπλοκότητας).

Λίστα αναγνωσμάτων

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison Wesley, 1994, 0-201-63361-2.

Κύρια είδη προτύπων

- Αρχιτεκτονικά (architectural)
  - MVC, MVVM, κ.ά
- Κατασκευαστικά (creational)
  - Factory, Builder, Object Pool, Singleton, κ.ά
- Δομικά (structural)
  - Adapter/Wrapper, Bridge, Decorator, Facade, Proxy, κ.ά
- Συμπεριφοράς (behavioral)
  - Chain of responsibility, Command, Iterator, Mediator, Observer, Strategy, Template method, Visitor, κ.ά

Παράδειγμα

Visitor

- Διαχωρισμός του αλγορίθμου από τη δομή δεδομένων.
- Η "λογική" της επίσκεψης στα στοιχεία της δομής (traversal) διαχωρίζεται από τη δομή αυτή καθ' αυτή.

Παράδειγμα

```
interface TreeVisitor {
  void visit(TreeNode node);
}

interface TreeNode {
  TreeNode getLeft();
  TreeNode getRight();
  int getValue();
  void accept(TreeVisitor visitor);
}

class SimpleTreeNode implements TreeNode {
  ...
  public void accept(TreeVisitor vistor) {
    visitor.visit(this);
  }
}
```

Υλοποίση

```
class PreOrderTreeVisitor implements TreeVisitor {

  public void visit(TreeNode node) {

    int value = node.getValue();
    //do something with the node's value
    ...

    //then traverse from left to right (pre-order)
    TreeNode left = node.getLeft();
    if (left != null) left.accept(this);

    TreeNode right = node.getRight();
    if (right != null) right.accept(this);
  }
}
```

Πραγματικό Παράδειγμα (Java 7+ FileVisitor)

java.nio.Files

```
static Path walkFileTree(
  Path start,
  FileVisitor<? super Path> visitor
) throws IOException
```

java.nio.file.FileVisitor

```
interface FileVisitor<T> {

  FileVisitResult postVisitDirectory(T dir, ...)

  FileVisitResult preVisitDirectory(T dir, ...)

  FileVisitResult visitFile(T file, ...)

  FileVisitResult visitFileFailed(T file, ...)

}
```

Θα επανέλθουμε σε επόμενη διάλεξη…