**ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ**

Χρόνης Βελέντζας

**ΠΡΟΤΥΠΑ ΣΧΕΔΙΑΣΗΣ**

Στόχος αυτής της ενότητας είναι η γνωριμία με επιπλέον αντιπροσωπευτικά **πρότυπα σχεδίασης** (design patterns), αξιοποιώντας γνώσεις των προηγούμενων ενοτήτων.

# Πρότυπα σχεδίασης

Τα **πρότυπα σχεδίασης** (design patterns) είναι λύσεις σε γενικά προβλήματα που έχουν αντιμετωπίσει διάφοροι προγραμματιστές ανά τον κόσμο κατά την ανάπτυξη λογισμικού. Οι λύσεις αυτές έχουν προκύψει έπειτα από σχετικά μακροχρόνιες διαδικασίες δοκιμών και αποτελούν χαρακτηριστικά παραδείγματα στα οποία η αφαίρεση μπορεί να προσφέρει αποτελεσματικές λύσεις.

Υπάρχουν κυρίως 3 διαφορετικοί τύποι προτύπων σχεδίασης:

* + - **Δημιουργικά πρότυπα** (creational patterns) για τη δημιουργία νέων αντικειμένων αποκρύπτοντας λεπτομέρειες που μπορεί να αφορούν ελέγχους στη διαδικασία αυτή. Στην κατηγορία αυτή ανήκει και το **πρότυπο του εργοστασίου** που μελετήσαμε.
    - **Δομικά πρότυπα** (structural patterns) για τον προσδιορισμό του τρόπου δόμησης των κλάσεων ώστε να μπορούν να αξιοποιηθούν ευκολότερα τα αντικείμενά τους ή να επεκταθούν ευκολότερα με νέες λειτουργίες.
    - **Πρότυπα συμπεριφοράς** (behavioral patterns) που αποσκοπούν στην επικοινωνία μεταξύ αντικειμένων.

Παρακάτω παρουσιάζονται αντιπροσωπευτικά παραδείγματα των παραπάνω κατηγοριών προτύπων σχεδίασης.

# Δημιουργικό μονήρες πρότυπο

Το **μονήρες πρότυπο** (singleton pattern) αποσκοπεί στη δημιουργία μόνο ενός αντικειμένου από μια κλάση και όχι περισσότερων. Προκειμένου να μη μπορεί να δημιουργηθούν αυθαίρετα, περισσότερα αντικείμενα. Ένα παράδειγμα υλοποίησης του μονήρους προτύπου εικονίζεται στο Σχ. 1. Παρατηρείστε ότι η μέθοδος δημιουργός άνευ ορισμάτων δηλώνεται private, μη επιτρέποντας έτσι τη δημιουργία αντικειμένων με έκφραση όπως η ακόλουθη:

# SingleObject x = new SingleObject();

Στο Σχ.2 παρουσιάζεται ο τρόπος με τον οποίο μπορεί να κληθεί η μέθοδος

getInstance για τη δημιουργία ενός αντικειμένου της κλάσης SingleObject.

**1 public class SingleObject 2 {**

1. **private static SingleObject instance =**
2. **new SingleObject();**

**5**

**6 private SingleObject(){} 7**

**8 public static SingleObject getInstance() 9 {**

**10 return instance; 11 }**

**12 public void showMessage()**

**13 {**

**14 System.out.println("Hello World!"); 15 }**

**16 }**

**Σχήμα 1.** Κλάση για την παραγωγή μονήρων αντικειμένων.

**1** public class SingletonPatternDemo

**2** {

**3** public static void main(String[] args)

**4** {

1. SingleObject object =
2. SingleObject.getInstance();
3. object.showMessage();

**8**

**9** }

**10** }

**Σχήμα 2.** Διεπαφή που περιγράφει ένα υπόδειγμα σχήματος.

# Πρότυπο πρόσοψης

Το **πρότυπο πρόσοψης** (façade pattern) είναι ένα δομικό πρότυπο που προσφέρει μια διεπαφή μέσω της οποίας μπορούν να καλούνται εύκολα μέθοδοι διαφορετικών αντικειμένων που εξυπηρετούν παρόμοιες λειτουργικότητες. Ως αντιπροσωπευτικό παράδειγμα δίνεται η κλάση ShapeMaker του Σχ.6, η οποία αξιοποιείται στην κλάση του Σχ.7. Οι κώδικες των Σχ.3‐5 είναι βοηθητικοί αλλά απαραίτητοι για την υλοποίηση του παραδείγματος.

**1 public interface Shape 2 {**

**3 public void draw(); 4 }**

**Σχήμα 3.** Διεπαφή που περιγράφει ένα υπόδειγμα σχήματος.

**1 public class Circle implements Shape 2 {**

**3 public void draw()**

**4 {**

**5 System.out.println("This is a circle."); 6 }**

**7 }**

**Σχήμα 4.** Κλάση που αναπαριστά ένα κύκλο όπως περιγράφεται από τη διεπαφή

Shape.

**1 public class Rectangle implements Shape 2 {**

**3 public void draw()**

**4 {**

**5 System.out.println("This is a rectangle."); 6 }**

**7 }**

**Σχήμα 5.** Κλάση που αναπαριστά ένα ορθογώνιο όπως περιγράφεται από τη διεπαφή

Shape.

**1 public class ShapeMaker 2 {**

1. **private Shape circle;**
2. **private Shape rectangle; 5**

**6 public ShapeMaker()**

**7 {**

1. **circle = new Circle();**
2. **rectangle = new Rectangle(); 10 }**

**11**

**12 public void drawCircle()**

**13 {**

**15 circle.draw(); 16 }**

**17**

**18 public void drawRectangle()**

**19 {**

**20 rectangle.draw(); 21 }**

**22 }**

**Σχήμα 6.** Μια κλάση Façade.

**1 public class FacadePatternDemo 2 {**

**3 public static void main(String[] args) 4 {**

1. **ShapeMaker shapeMaker = new ShapeMaker();**
2. **shapeMaker.drawCircle();**
3. **shapeMaker.drawRectangle();**
4. **}**

**9 }**

**Σχήμα 7.** Παράδειγμα χρήσης της κλάσης του Σχ.6.

# Πρότυπο στρατηγικής

Το **πρότυπο στρατηγικής** (strategy pattern) είναι ένα πρότυπο συμπεριφοράς που προσφέρει τη δυνατότητα αλλαγής της συμπεριφοράς μιας κλάσης ή ενός αλγορίθμου σε χρόνο εκτέλεσης. Ένα τέτοιο παράδειγμα υλοποιούν οι παρακάτω κλάσεις. Οι διαφορετικές συμπεριφορές χαρακτηρίζονται ως στρατηγικές οι οποίες αναπαριστώνται με διαφορετικές κλάσεις (Σχ.9‐10). Οι στρατηγικές πρέπει να έχουν αντίστοιχες λειτουργίες, όπως περιγράφονται από μια κοινή διεπαφή Strategy (Σχ.8). Η επιλογή της εκάστοτε στρατηγικής γίνεται μέσω μιας ενδιάμεσης κλάσης που αναπαριστά ένα πλαίσιο καθορισμού της στρατηγικής (Σχ.11).

Στο Σχ.12 εικονίζεται ένα παράδειγμα χρήσης του προτύπου στρατηγικής για την αλλαγή της αριθμητικής πράξης που επιθυμούμε να υλοποιηθεί ανάμεσα σε δύο αριθμούς, π.χ. πρόσθεση ή πολλαπλασιασμό, σε χρόνο εκτέλεσης. Η μεθοδολογία αυτή προσφέρει ιδιαίτερη ευελιξία και επαναχρησιμοποιησιμότητα (reusability) στα προγράμματά μας.

**1 public interface Strategy 2 {**

**3 public int doOperation(int num1, int num2); 4 }**

**Σχήμα 8.** Μια διεπαφή που αναπαριστά τη στρατηγική.

**1 public class OperationAdd implements Strategy 2 {**

**3 public int doOperation(int num1, int num2) 4 {**

**5 return num1 + num2; 6 }**

**7 }**

**Σχήμα 9.** Μια κλάση που υλοποιεί ως στρατηγική την πράξη της πρόσθεσης.

**1 public class OperationMultiply implements Strategy 2 {**

**3 public int doOperation(int num1, int num2) 4 {**

**5 return num1 \* num2; 6 }**

**7 }**

**Σχήμα 10.** Μια κλάση που υλοποιεί ως στρατηγική την πράξη του πολλαπλασιασμού.

**1 public class Context 2 {**

**3 private Strategy strategy; 4**

**5 public Context(Strategy strategy)**

**6 {**

**7 this.strategy = strategy; 8 }**

**9**

**10 public int executeStrategy(int num1, int num2)**

**11 {**

**12**

**13 }**

**14 }**

**return strategy.doOperation(num1, num2);**

**Σχήμα 11.** Μια κλάση που αναπαριστά ένα πλαίσιο μέσω του οποίου καθορίζεται η εκάστοτε στρατηγική σε χρόνο εκτέλεσης.

**1 public class StrategyPatternDemo 2 {**

**3 public static void main(String[] args) 4 {**

1. **// Perform addition**
2. **Context context = new Context(new**
3. **OperationAdd());**
4. **System.out.println("10 + 5 = " +**
5. **context.executeStrategy(10, 5)); 10**
6. **// Perform multiplication**
7. **context = new Context(new OperationMultiply()); 13**
8. **System.out.println("10 \* 5 = " +**
9. **context.executeStrategy(10, 5)); 16 }**

**17 }**

**Σχήμα 12.** Παράδειγμα χρήσης της κλάσης του Σχ.11 για την υλοποίηση διαφορετικής πράξης (πρόσθεσης ή πολλαπλασιασμού) με τις ίδιες κλάσεις, σε χρόνο εκτέλεσης.

# Βιβλιογραφία

Περισσότερα πρότυπα σχεδίασης μπορείτε να βρείτε στις ακόλουθες πηγές:

1. <http://www.tutorialspoint.com/design_pattern/design_pattern_quick_guide.htm>
2. <http://www.avajava.com/tutorials/categories/design>‐patterns
3. S. Stelting, and O. Maassen, Applied Java Patterns, The Sun Microsystems Press, 2002.

# Ασκήσεις

1. Να υλοποιηθούν τα παραπάνω παραδείγματα.
2. Στο πλαίσιο του παραδείγματος του προτύπου στρατηγικής να προστεθεί η δυνατότητα, ανάμεσα στους δύο αριθμούς να εκτελείται η πράξη a\*a + b\*b.
3. Να κατασκευάσετε βάσει του προτύπου στρατηγικής ένα χαρακτήρα κόμικ, όπως είναι ο Ντόναλντ Ντακ, ο Γκούφη κλπ. Ο χαρακτήρας αυτός θα μπορεί να μιλάει. Όμως κάθε χαρακτήρας μιλά με το δικό του ιδιαίτερο τρόπο, π.χ. ο Ντόναλντ στην αρχή κάθε φράσης του θα λέει “quackquack…” ενώ ο Γκούφη θα λέει αντίστοιχα “hugh…”. Βάσει το Σχ. 12 να κατασκευάσετε ένα παράδειγμα στο οποίο οι χαρακτήρες θα μιλάνε μεταξύ τους χρησιμοποιώντας την ίδια μέθοδο.