14η Διάλεξη

Προχωρημένα θέματα σχεδίασης

Υποέργο 1 «Αυτεπιστασία του τμήματος Πληροφορικής και Επικοινωνιών»,
«Ενίσχυση Σπουδών Πληροφορικής στο ΤΕΙ Σερρών» Μέτρο 2.2, Γ' ΚΠΣ

ΤΕΙ ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΑΝΟΙΚΤΑ ΑΚΑΔΗΜΑΙΚΑ ΜΑΘΗΜΑΤΑ

ΤΕΧΝΟΛΟΓΙΑ ΛΟΓΙΣΜΙΚΟΥ Ι

κ. ΠΕΤΑΛΙΔΗΣ

ΤΜΗΜΑ **ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΤΕ**

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

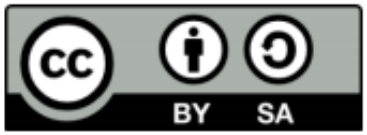
ΕΣΠΑ 2007-2013
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.

- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο ΤΕΙ Κεντρικής Μακεδονίας» 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.

Μοτίβα σχεδίασης

- Οι σχεδιαστές έχουν διαπιστώσει ότι συγκεκριμένα μοτίβα (patterns) εμφανίζονται διαρκώς κατά τη σχεδίαση προγραμμάτων
- Για αυτό έχουν αναπτύξει συγκεκριμένες λύσεις οι οποίες μπορούν να εφαρμοστούν όποτε αναγνωρίζεται ότι το πρόβλημα εντάσσεται σε ένα από τα γνωστά μοτίβα ανάπτυξης

Τι είναι τα μοτίβα σχεδίασης;

- Προσφέρουν λύσεις σε προβλήματα που συναντιούνται συχνά
- Είναι γενικά και μπορούν να εφαρμοσθούν σε πολλές περιπτώσεις
- Περιγράφουν τη μορφή του κώδικα και όχι τις λεπτομέρειες

Υποέργο 1 «Αυτεπιστασία του τμήματος Πληροφορικής και Επικοινωνιών»,
«Ενίσχυση Σπουδών Πληροφορικής στο ΤΕΙ Σερρών» Μέτρο 2.2, Γ' ΚΠΣ

Μοτίβα Σχεδίασης

- Υπάρχουν πολλά μοτίβα σχεδίασης αλλά αρχικά ορίσθηκαν 23 βασικά
- Κάθε μοτίβο έχει τουλάχιστον
  - Ένα όνομα
  - Ένα σκοπό για τον οποίο δημιουργήθηκε
  - Μια περιγραφή του προβλήματος το οποίο προσπαθεί να λύσει

Κατηγορίες μοτίβων σχεδίασης

- Δημιουργικά μοτίβα (Creational)
  - Ποιος είναι ο καλύτερος τρόπος να φτιάχνεις αντικείμενα
    - Abstract Factory, **Factory,** Prototype, Singleton
- Μοτίβα συμπεριφοράς (Behavioural)
  - Επικοινωνία μεταξύ αντικειμένων
    - Iterator, Visitor
- Δομικά μοτίβα (Structural)
  - Συνδυασμός ομάδων αντικειμένων
    - Composite, Adaptor
- Παρακάτω θα παρουσιαστεί το Factory Pattern ως παράδειγμα μοτίβου

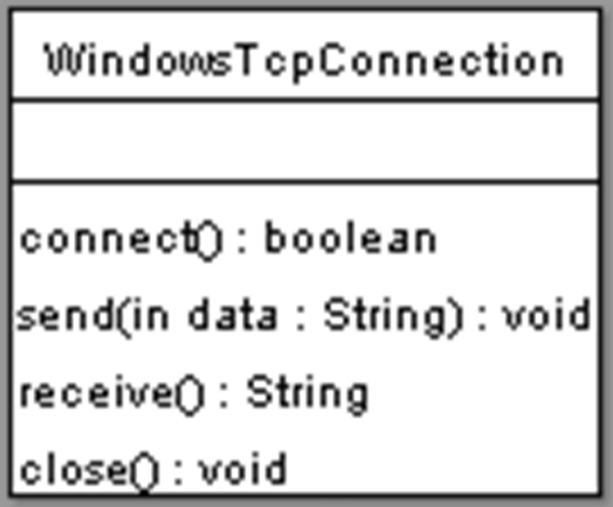
Υποέργο 1 «Αυτεπιστασία του τμήματος Πληροφορικής και Επικοινωνιών»,
«Ενίσχυση Σπουδών Πληροφορικής στο ΤΕΙ Σερρών» Μέτρο 2.2, Γ' ΚΠΣ

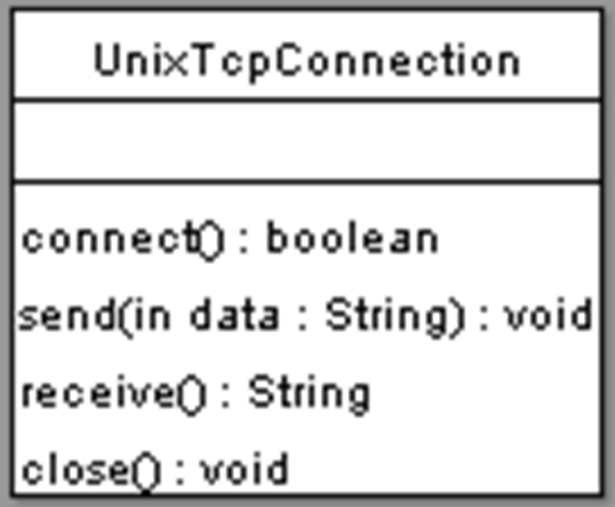
Παράδειγμα

- Γράφετε κώδικα και για Windows και για Unix.
- Πρέπει να γράψετε μια κλάση η οποία είναι υπεύθυνη για τη δημιουργία μιας σύνδεσης TCP
- Όμως η κλάση αυτή πρέπει να υλοποιεί διαφορετικά τη σύνδεση, αν πρόκειται για την υλοποίηση σε Windows και διαφορετικά αν πρόκειται για την υλοποίηση σε Unix

Χωρίς το Factory Pattern

- Ορίζετε δύο κλάσεις
  - WindowsTcpConnection
  - UnixTcpConnection





Υποέργο 1 «Αυτεπιστασία του τμήματος Πληροφορικής και Επικοινωνιών»,
«Ενίσχυση Σπουδών Πληροφορικής στο ΤΕΙ Σερρών» Μέτρο 2.2, Γ' ΚΠΣ

Χωρίς το Factory Pattern

- Και μέσα στον κώδικά σας, **όποτε** χρειάζεται να χρησιμοποιήσετε αυτήν την κλάση έχετε:

```
if (platform == "Unix") {
     UnixTcpConnection connection;
      connection.connect();
} else if (platform == "Windows") {
     WindowsTcpConnection connection;
   connection.connect();
}

     // κ.τ.λ.
```

Χωρίς το Factory Pattern

- Αν αργότερα αποφασίσετε να προσθέσετε και μια υλοποίηση για Mac, τότε θα προσθέσετε μια κλάση MacTcpConnection και παντού θα πρέπει να προσθέσετε τον αντίστοιχο κώδικα

Χωρίς το Factory Pattern

```
if (platform == "Unix") {
  UnixTcpConnection connection;
connection.connect();
} else if (platform == "Windows") {
  WindowsTcpConnection connection;
connection.connect();
} else if (platform == "Mac") {
  MacTcpConnection connection;
connection.connect();
}

  // κ.τ.λ.
```


Υποέργο 1 «Αυτεπιστασία του τμήματος Πληροφορικής και Επικοινωνιών»,
«Ενίσχυση Σπουδών Πληροφορικής στο ΤΕΙ Σερρών» Μέτρο 2.2, Γ' ΚΠΣ

Χωρίς το Factory Pattern

- Τα πράγματα είναι ακόμα πιο δύσκολα αν αποφασίσετε να κρατάτε και μία λίστα με όλες τις ανοιχτές συνδέσεις σας. Σε αυτήν την περίπτωση θα πρέπει να γράψετε τον ακόλουθο κώδικα

Υποέργο 1 «Αυτεπιστασία του τμήματος Πληροφορικής και Επικοινωνιών»,
«Ενίσχυση Σπουδών Πληροφορικής στο ΤΕΙ Σερρών» Μέτρο 2.2, Γ' ΚΠΣ

```
vector<UnixTcpConnection> unixTcpConnections;
vector<WindowsTcpConnection> windowsTcpConnections;

if (platform == "Unix") {
   UnixTcpConnection connection;
   unixTcpConnections.add(connection);
    connection.connect();
} else if (platform == "Windows") {
   WindowsTcpConnection connection;
   windowsTcpConnections.add(connection);
   connection.connect();
}
   // κ.τ.λ.
```

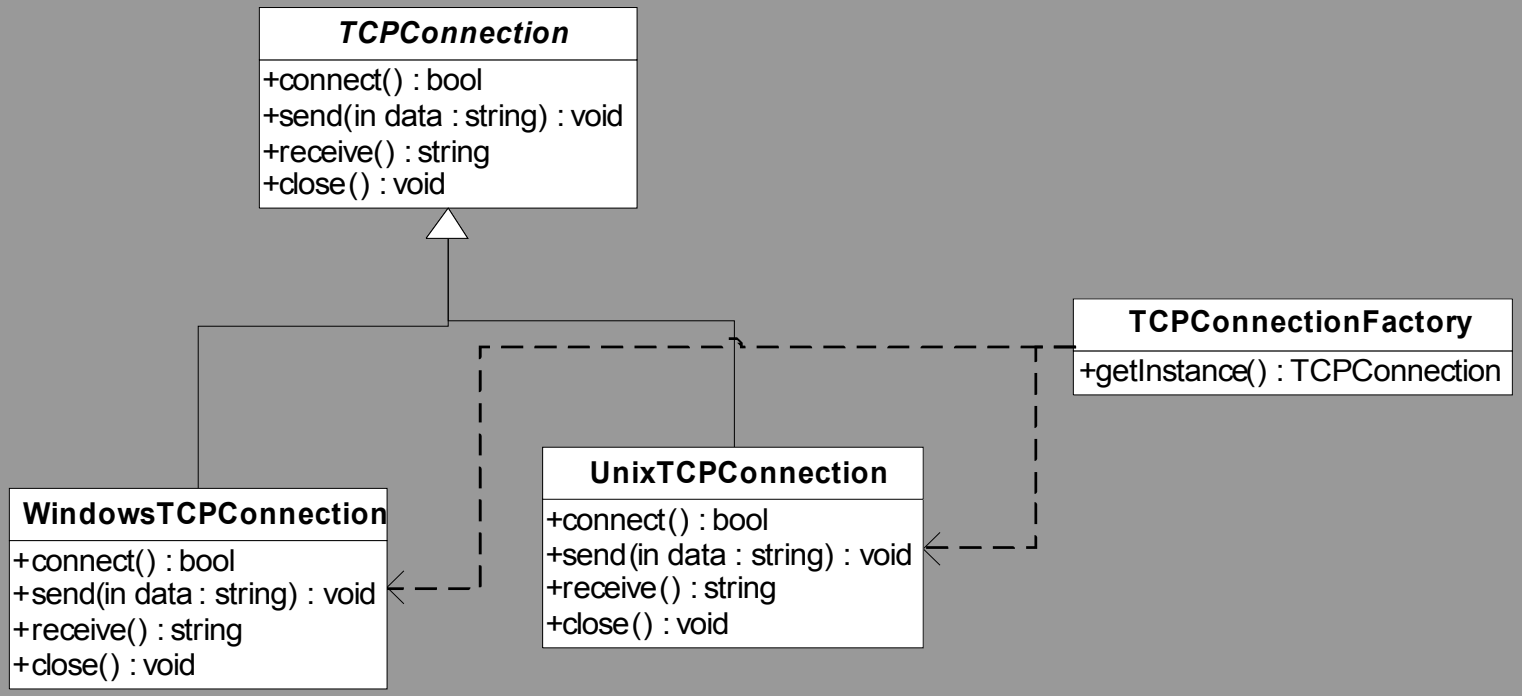
Προβλήματα

- Το κύριο πρόβλημα είναι ότι πρέπει να γράφετε τον κώδικά σας 2 φορές, μία για κάθε υλοποίηση.
- Αυτό τον κάνει πολύπλοκο, δυσανάγνωστο ενώ είναι εύκολο να γίνουν και λάθη

Με το Factory Pattern

- Χρησιμοποιώντας την κληρονομικότητα και τον πολυμορφισμό μπορούμε να καταφύγουμε σε μια λύση η οποία είναι αρκετά πιο κομψή
- Ορίζετε κλάσεις όπως στο ακόλουθο διάγραμμα

Με το Factory Pattern



Με το FactoryPattern

- Η κλάση `TcpConnection` είναι abstract, δεν περιέχει υλοποίηση δηλαδή των μεθόδων `connect(), send(), receive(), close().`
- Ορίζει ουσιαστικά τη διεπαφή επικοινωνίας με τις κλάσεις

`WindowsTcpConnection,`

`UnixTcpConnection`

Με το Factory Pattern

- Η κλάση TcpConnectionFactory περιέχει μια μέθοδο getInstance η οποία μπορεί να οριστεί όπως παρακάτω:

```
class TcpConnectionFactory {
 public:
      static TcpConnection &getInstance(string type) {
       if (type=="Windows") {
          WindowsTcpConnection *connection = new
      WindowsTcpConnection();
          return connection;
       } else {
          UnixTcpConnection *connection = new
      UnixTcpConnection();
         return connection;
       }
};
```

Με το FactoryPattern

- Και πώς χρησιμοποιείται; Απλά:

```
TcpConnection connection =
    TcpConnectionFactory::getInstance("Windows");

connection.connect();
```


Υποέργο 1 «Αυτεπιστασία του τμήματος Πληροφορικής και Επικοινωνιών»,
«Ενίσχυση Σπουδών Πληροφορικής στο ΤΕΙ Σερρών» Μέτρο 2.2, Γ' ΚΠΣ

Με το Factory Pattern

- Αν θέλετε να κρατάτε και μια λίστα με όλες τις συνδέσεις σας;

```
vector<TcpConnection> connections;
TcpConnection connection =
    TcpConnectionFactory.getInstance("Windows
    ");
connections.add(connection);
connection.connect();
```

Factory Pattern

- Σκοπός
  - Ορίζει μια διεπαφή για τη δημιουργία αντικειμένων αλλά αφήνει τις υποκλάσεις να αποφασίσουν τι είδους αντικείμενο θα φτιάξουν
- Που μπορεί να χρησιμοποιηθεί
  - Όταν χρειάζεται να δημιουργήσετε αντικείμενα κλάσεων που έχουν την ίδια διεπαφή (αλλά διαφορετική υλοποίηση) και δε γνωρίζετε τη συγκεκριμένη στιγμή τι είδους αντικείμενα θα χρειαστείτε

Υποέργο 1 «Αυτεπιστασία του τμήματος Πληροφορικής και Επικοινωνιών»,
«Ενίσχυση Σπουδών Πληροφορικής στο ΤΕΙ Σερρών» Μέτρο 2.2, Γ' ΚΠΣ

Μοτίβα σχεδίασης

- Τα περισσότερα μοτίβα σχεδίασης στηρίζονται σε έξυπνη χρήση
  - Της κληρονομικότητας
  - Του πολυμορφισμού
  - Της περιεκτικότητας

Υποέργο 1 «Αυτεπιστασία του τμήματος Πληροφορικής και Επικοινωνιών»,
«Ενίσχυση Σπουδών Πληροφορικής στο ΤΕΙ Σερρών» Μέτρο 2.2, Γ' ΚΠΣ