Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής & Τηλεπικοινωνιών

Τεχνολογία Λογισμικού

8ο Εξάμηνο 2022-23

Πρότυπα σχεδίασης (1/2)

Δρ. Κώστας Σαΐδης (saiko@di.uoa.gr)

Πρότυπο σχεδίασης (Design pattern)

Μια βέλτιση πρακτική για την αντιμετώπιση ενός σχεδιαστικού προβλήματος στο λογισμικό.

- Ένας τρόπος δόμησης/διάρθωσης του κώδικα για την επίλυση ενός συγκεκριμένου προβλήματος (όχι ο κώδικας καθ' αυτός).
- Η υπερβολική χρήση των προτύπων μπορεί να επιφέρει περισσότερα προβλήματα απ' όσα λύνει (κάνει τον κώδικο πιο δυσνόητο).

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, Multi-tier, 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

Αρχιτεκτονικά πρότυπα

- Τα είδαμε στις αντίστοιχες διαλέξεις
- Ασχολούνται με τη διάρθρωση του κώδικα σε υψηλό αρχιτεκτονικό επίπεδο (υποσυστήματα, συστατικά, επικοινωνία μεταξύ τους).

Κατασκευαστικά, δομικά και πρότυπα συμπεριφοράς

- Θα τα δούμε στη συνέχεια
- Ασχολούνται με τη διάρθρωση του κώδικα στο χαμηλό επίπεδο των κλάσεων (κατασκευή, δομή και συμπεριφορά αντικειμένων).

Παράδειγμα σχεδιαστικού προτύπου

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);
}

abstract class TreeNode {
  TreeNode getLeft();
  TreeNode getRight();
  int getValue();
  void accept(TreeVisitor visitor) {
    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, ...)

}
```

Κατασκευαστικά πρότυπα σχεδίασης

Πρότυπα σχετικά με την κατασκευή αντικειμένων / στιγμιοτύπων

Περιεχόμενα

Singleton

Σκοπός: Να διαφυλάξει ότι μια κλάση έχει μόνο ένα στιγμιότυπο (και να προσφέρει την πρόσβαση σε αυτό)

Παράδειγμα

Eager initialization

```
public class Configuration {
  private static final Configuration c = new Configuration();
  private Configuration() {
    ...
  }
  public static Configuration getInstance() {
    return c;
  }
}
```

Από το repo του μαθήματος

Lazy initialization

```
public class Configuration {
  private static Configuration c;
  private Configuration() {
    ...
  }
  public static Configuration getInstance() {
    if (c == null) {
      synchronized(Configuration.class) {
        if (c == null) {
          c = new Configuration();
        }
      }
    }
    return c;
  }
}
```

Πραγματικό παράδειγμα (JVM Runtime)

http://docs.oracle.com/javase/8/docs/api/java/lang/Runtime.html#getRuntime--

Συζήτηση

- Το singleton πρότυπο εξυπηρετεί όταν θέλουμε ένας "ευαίσθητος" για την εφαρμογή πόρος να αρχικοποιείται μόνο μία φορά (π.χ. configuration file reader, data access handler, κλπ.)
- Προβληματική κληρονομικότητα (λόγω του private constructor)
- Απαιτείται προσεκτική και "φειδωλή" εφαρμογή του προτύπου

Factory Method & Abstract Factory

Σκοπός: Να αποκρύψουν τις λεπτομέρειες δημιουργίας "συγγενών" στιγμιοτύπων

Factory Method (aka Virtual constructor)

Ένα πρόσθετο επίπεδο ανακατεύθυνσης / αφαίρεσης (level of indirection) στην απευθείας εκτέλεση ενός κατασκευαστή (constructor).

Η κλάση - χρήστης βασίζεται στον τύπο των στιγμιοτύπων και όχι σε συγκεκριμένη κλάση (concrete class).

Παράδειγμα

```
abstract class MVCResource { //controller
  abstract User createUser(HttpRequest req)
  void doService(HttpRequest req, HttpResponse res) {
    User user = createUser(req)
    Template tpl = Helper.getTemplateFrom(req)  //view
    Context ctx = Helper.createContextFrom(req) //model
    tpl.render(user, ctx, res)
  }
}
```

```
class PublicResource extends MVCResource {
  User createUser(req) {
    String id = req.getCookie().get("session-id")
    if (id) {
      User user = SessionDB.getUserFrom(id)
      if (user) {
        return user //for logging purposes
      }
    }
    return User.ANONYMOUS;
  }
}
```

```
class ProtectedResource extends MVCResource {
  User createUser(req) {
    String id = req.getCookie().get("session-id")
    if (id) {
      User user = SessionDB.getUserFrom(id)
      if (user && user.isAuthorizedFor(this)) {
        return user
      }
    }
    throw new AuthorizationException("Not authorized")
  }
}
```

Συζήτηση

Χρήση του προτύπου:

- όταν μια κλάση δεν μπορεί να "ξέρει πλήρως" τι στιγμιότυπα θα δημιουργηθούν
- όταν μια κλάση "θέλει" να μεταθέσει την αρμοδιότητα δημιουργίας στιγμιοτύπων σε υποκλάσεις της
- όταν μπορεί να προκύψει σφάλμα (exception) εντός των κατασκευαστών (constructors) ώστε να αποφευχθούν "ασυνεπή" στιγμιότυπα

Abstract Factory

Ένα interface δημιουργίας "συγγενών" στιγμιοτύπων

Παράδειγμα

```
interface Product {
   ...
}

class Tablet implements Product {
   ...
}

class Mobile implements Product {
   ...
}
```

```
interface ProductFactory {
  Product newProduct(HttpRequest req)
}

class TabletFactory implements ProductFactory {
  Product newProduct(HttpRequest req) {
    //process req
    return new Tablet(...)
  }
}

class MobileFactory implements ProductFactory {
  Product newProduct(HttpRequest req) {
    //process req
    return new Mobile(...)
  }
}
```

```
class ShopingCart {

  private final List<Product> = new ArrayList()

  static Map<String, ProductFactory> factories = [
    "Tablet": new TabletFactory(),
    "Mobile": new MobileFactory()
  ]

  void addProduct(HttpRequest req) {
    String family = req.getParameter("family")
    ProductFactory factory = factories.get(family)
    if (factory) {
      products.add(factory.newProduct(req))
    }
    else {
      throw new BadRequestException("Invalid product family")
    }
  }
}
```

Συζήτηση

Χρήση του προτύπου:

- όταν μια κλάση είναι ανεξάρτητη του πώς δημιουργούνται ή αναπαρίστανται τα "στοιχεία" που συνθέτει
- όταν μια κλάση θα "αποφασίσει" στο runtime τι στιγμιότυπα πρέπει να δημιουργηθούν

Πραγματικό παράδειγμα (XML Parsing)

https://docs.oracle.com/javase/8/docs/api/javax/xml/parsers/DocumentBuilderFactory.html#newInstance--

Builder

Σκοπός: Να απλοποιήσει τη δημιουργία σύνθετων αντικειμένων (που απαιτούν πολλές παραμέτρους)

Παράδειγμα

```
Something s = new BuilderOfSomething().
                  setAnOption(1).
                  setAnother("x").
                  setAFlag(false).
                  build();
```

Η υλοποίηση

```
class BuilderOfSomething {
  private Something s;

  BuilderOfSomething() {
    s = new Something();
  }

  BuilderOfSomething setAnOption(int option) {
    s.setInt(option);
    return this; //allows chaining of calls
  }

  BuilderOfSomething setAnother(String option) {
    s.setString(option);
    return this;
  }

  ...

  Something build() {
    return s;
  }
}
```

Πραγματικό Παράδειγμα (Guava Cache)

```
Cache<String, X> cache = CacheBuilder.
    newBuilder().
    maximumSize(100).
    expireAfterWrite(1, TimeUnit.MINUTES).
    removalListener(
    new RemovalListener<String, X>() {
      @Override
      void onRemoval(RemovalNotification<String, X> n) {
        ...
      }
    }).
    build()
```

Πραγματικό Παράδειγμα (Guava Cache)

```
Cache<String, X> cache = CacheBuilder.
    newBuilder().
    maximumSize(100).
    expireAfterWrite(1, TimeUnit.MINUTES).
    removalListener( (RemovalNotification<String, X> n) -> {
      ...
    }).
    build()
```

> If you have a procedure with ten parameters, you probably missed some.

Alan Perlis

Συζήτηση

- Αποφυγή κατασκευαστών με πολλές παραμέτρους (>6)
- Επιλογή του χρήστη ποια παράμετρο θα θέσει ρητά (οι υπόλοιπες θα πάρουν κάποια default τιμή)
- Η "κατασκευαστική" διαδικασία πρέπει να υποστηρίζει διαφορετικές "αναπαραστάσεις" για τα αντικείμενα υπό κατασκευή

Object Pool

Σκοπός: Να διατηρήσει "ζωντανά" σε μια μορφή ενδιάμεσης μνήμης (cache) αντικείμενα των οποίων η κατασκευή είναι "ακριβή"

Συζήτηση

- Thread Pools, Connection Pools, κτλ.
- Μοτίβο borrow & return.
- Εκ των προτέρων κατασκευή συγκεριμένου πλήθους αντικειμένων που επαναχρησιμοποιούνται αντί να κατασκευάζονται και να καταστρέφονται σε κάθε χρήση.
- Άνω όρια στη δέσμευση πόρων (αποφυγή επιθέσεων τύπου Denial-Of-Service).
- Μικρά όρια μπορεί να μειώσουν την απόδοση.

Παράδειγμα

Database Connection Pool στο repo του μαθήματος

Thread Pools (Java 5+ Executors)

java.util.concurrent.Executors

```
static ExecutorService newCachedThreadPool()

static ExecutorService newFixedThreadPool()
```

Από το Javadoc

newFixedThreadPool()

Creates a thread pool that reuses a fixed number of threads operating off a shared unbounded queue. At any point, **at most n threads will be active** processing tasks. If additional tasks are submitted when all threads are active, they will wait in the queue until a thread is available. If any thread terminates due to a failure during execution prior to shutdown, a new one will take its place if needed to execute subsequent tasks. The threads in the pool will exist until it is explicitly shutdown.

newCachedThreadPool()

Creates a thread pool that creates new threads as needed, but will reuse previously constructed threads when they are available. These pools will typically improve the performance of programs that **execute many short-lived asynchronous tasks**. Calls to execute will reuse previously constructed threads if available. If no existing thread is available, a new thread will be created and added to the pool. Threads that have not been used for sixty seconds are terminated and removed from the cache. Thus, a pool that remains idle for long enough will not consume any resources.

Prototype

Σκοπός: Δημιουργία νέου στιγμιοτύπου από την "αντιγραφή" ενός υπάρχοντος

Java Cloneable

http://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#clone()

Παράδειγμα

Shallow cloning vs deep cloning

https://programmingmitra.blogspot.gr/2016/11/Java-Cloning-Types-of-Cloning-Shallow-Deep-in-Details-with-Example.html

Copy Constructor pattern

Εναλλακτική στο cloning

```
public Person(Person original) {
    this.id = original.id + 1;
    this.name = new String(original.name);
    this.city = new City(original.city);
}
```