



Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής & Τηλεπικοινωνιών

Προηγμένες Μέθοδοι Προγραμματισμού

ΠΜΣ (M135.CS1E, M135.CS23B, M135.IC1E, παλαιό: M117)

Πρότυπα σχεδίασης (2)

Δρ. Κώστας Σαΐδης (saiko@di.uoa.gr)

Κατασκευαστικά πρότυπα σχεδίασης

Πρότυπα σχετικά με την κατασκευή αντικειμένων /
στιγμιότυπων

Περιεχόμενα

1. Singleton
2. Factory Method
3. Abstract Factory
4. Builder
5. Object Pool
6. Prototype

Singleton

Σκοπός: Να διαφυλάξει ότι ένας τύπος/κλάση έχει μόνο ένα στιγμιοτύπο (και να προσφέρει την πρόσβαση σε αυτό)

Πώς το υλοποιούμε;

- Στη Java -συνήθως- βασιζόμαστε στον class loading μηχανισμό του JVM, όπου μια κλάση φορτώνεται και αρχικοποιείται μία -και μόνο μία- φορά
- Τουλάχιστον, αν η κλάση δεν γίνει GCed (που στις περισσότερες εφαρμογές δεν συμβαίνει)

Πώς το υλοποιούμε;

- Στη Javascript είναι πιο απλό, βασιζόμαστε στα scopes (θα το δούμε σε άλλη διάλεξη)

Παράδειγμα

Eager initialization

```
public class Configuration {  
    // The class holds an instance of itself as a static member  
    private static final Configuration c = new Configuration();  
    // Private constructor (cannot be invoked by any other class)  
    private Configuration() {  
        ...  
    }  
    // Public getter of the singleton instance  
    public static Configuration getInstance() {  
        return c;  
    }  
}
```

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(HttpServletRequest req)
    void doService(HttpServletRequest req, HttpServletResponse 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(HttpServletRequest req)  
}  
  
class TabletFactory implements ProductFactory {  
    Product newProduct(HttpServletRequest req) {  
        //process req  
        return new Tablet(...)  
    }  
}  
  
class MobileFactory implements ProductFactory {  
    Product newProduct(HttpServletRequest req) {  
        //process req  
        return new Mobile(...)  
    }  
}
```

```
class ShoppingCart {  
  
    private final List<Product> = new ArrayList()  
  
    static Map<String, ProductFactory> factories = [  
        "Tablet": new TabletFactory(),  
        "Mobile": new MobileFactory()  
    ]  
  
    void addProduct(HttpServletRequest 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().  
        maxSize(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().  
        maxSize(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 συγκεκριμένου 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

Remote connection Pool (e.g. http connections)

Database Connection Pool

Java Thread Pools (θα τα δούμε λεπτομερώς σε άλλη διάλεξη)

Παράδειγμα `ewFixedThreadPool(n)`

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.

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\(\)`](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);  
}
```