UNIVERSITAT DE LLEIDA
Escola Politècnica Superior
Grau en Enginyeria Informàtica
Ampliació de BBDD i ES

# Pràctica de Patrons de Disseny (v1)

Sergi Simón Balcells, Ian Palacin Aliana, Joaquim Picó Mora
PraLab2

Professorat : J.M.Gimeno
Data : Dilluns 11 de Maig

# Contents

# 1 Introducció

En aquesta pràctica es crea una petita infraestructura per facilitar la interconnexió de les instàncies de diferents classes fent ús del patró injector. Les principals decisions d'implementació i dificultats trobades queden reflectides en aquest document, així com els tests emprats per facilitar el desenvolupament.

# 2 Detalls implementació

## 2.1 Map

Es va decidir utilitzar un Map, ja que els mètodes de la mateixa interfície d'injector portaven a pensar amb un Map. En aquest, existeix el mètode de "get" per una clau amb el mètode `getObject`, i tres mètodes per a afegir valors amb `registerConstant, registerFactory` i `registerSingleton`.

### 2.1.1 Value

El problema que resta és el tipus de valor que es guardarà en el diccionari. No es podia guardar instàncies d'objecte, ja que podria ser que un `Factory` necessites tornar sempre elements diferents per un camp intern, com per exemple, una classe `Factory` que crea noves identificacions pels clients que la utilitzen.

Descartant la possibilitat de guardar valors instanciats només quedava guardar funcions. La idea és utilitzar una funció del tipus `() ->;;; a` (en notació similar a Haskell) per a extreure la idea de constant, `Singleton` i `Factory`.

A més a més, s'ha optat per fer una altra interfície per a fer-ho, que s'ha anomenat `FunctionUnitToObject`. Realment es podia haver utilitzat la interfície de `Factory` donada, però s'ha cregut que era més costós d'entendre que guardava el Map, pel que es va optar per fer una interfície a part.

## 2.2 Throwables

S'ha fet ús de l'excepció DependencyException pels casos mencionats a l'enunciat de la pràctica. El que val més la pena mencionar és el cas que preveu crear un objecte que forma un cicle de dependències.

Per trobar el cicle de dependències s'utilitzarà un set, on s'afegiran les claus que s'han demanat quan es fa la crida a getObject. Si en un moment donat es demana una clau que ja està al set voldrà dir que s'ha creat un cicle de dependències, i es llençarà l'excepció.

## 2.3 Implementacions

Per objectiu propi, es va voler que les implementacions fossin mandroses. D'aquesta forma, sigui l'ordre que sigui quan es registrin les dependències, quan es resolgui el valor es retornarà el valor sempre que es pugui. Per a fer-ho es va crear el mètode `getObjects`, el qual tradueix en les crides a les classes les dades.

### 2.3.1 registerSingelton()

Els mètodes de constant i `Factory` són trivials donat aquest disseny, però el mètode de `Singleton` s'ha hagut de realitzar de forma diferent. L'objectiu en el qual es volia arribar era que en el primer `getObject` es crea per primer cop la instància, però la resta de vegades no. Per a guardar aquest estat s'ha utilitzat una classe anònima, on existeix un objecte primerament instanciat a null. Quan es cridi per primer cop en el "get", es crearà l'objecte, la resta de vegades, es retornarà el mateix.

### 2.4 Dificultats

## 3 Tests

### 3.1 Mocks

Per tal de poder testejar el funcionament correcte de l'injector s'han implementat quatre mocks diferents. Cada un d'aquests compta amb la seva respectiva interfície i implementació i també amb les seves Factories, tant la simple com la complexa.

### 3.2 Tests Especifics

La raó per la qual s'han decidit implementar testos específics és per seguir les bones pràctiques, donatel el cas que sorgeixin errors específics en el codi, es puguin afegir els testos ràpidament en una classe. Es tracta de les classes de test ContainerConstantTest, ContainerFactoryTest i ContainerSingeltonTest.

### 3.3 Tests Generals

Per altra banda, també s'ha fet una classe de test general per provar el funcionament de l'injector. Aquest ens permet veure que amb múltiples objectes registrats el programa es comporta correctament. Es tracta de la classe ContainerTest

## 4 Conclusions

Fer ús del patró injector és una bona forma de tenir les dependències localitzades i ordenades. Fer una versió senzilla d'un injector és relativament ràpid i a la vegada fa entendre amb més claredat aquest patró.