

# Pràctica introductòria #6 - Arquitectura neta

Projecte Integrat de Software (2024/25)  
Facultat de Matemàtiques i Informàtica  
Universitat de Barcelona

## 1 Introducció

L'arquitectura neta (*clean architecture*) és un enfocament de disseny de programari que organitza els components en capes concèntriques segons el seu nivell d'abstracció, seguint la *regla de dependència*. Aquesta regla estableix que les capes interiors, més estables i independents dels detalls concrets, no han de dependre de les capes exteriors, que són més volàtils i específiques.

Per exemple, la interfície d'usuari (UI) i els mecanismes de persistència són elements que poden canviar amb freqüència, per la qual cosa es col·loquen a les capes més externes. En canvi, els objectes del domini i els casos d'ús, que encapsulen la lògica fonamental del sistema, es mantenen a les capes interiors, protegits de modificacions externes. Si un component d'una capa interior ha de comunicar-se amb un de l'exterior, ho fa mitjançant una abstracció que la capa exterior implementa.

Aquesta guia estructurarà el codi en mòduls i paquets que reflecteixin aquesta organització, assegurant un compliment estricte de la regla de dependència.

## Requeriments

Haver fet les pràctiques introductòries anteriors (PI1 - PI5).

## Objectius i temps

1. Repassar els conceptes claus de l'arquitectura neta.
2. Saber aplicar l'arquitectura neta en context d'una aplicació Android.
3. Saber fer injecció de dependències (manual) en una aplicació Android.
4. Veure una manera de propagar errors a través de l'arquitectura neta sense vulnerar la regla de dependència.

## Objectius i temps

- `CleanArchitectureExample.zip`: projecte base d'Android Studio descarregable del campus virtual, per fer el seguiment d'aquest guió. Està basat en el codi de la solució de la PI5 (`FirebaseRepositoryExample.zip`) per a fer-ho incremental.

## 2 Arquitectura Neta

Proposada per Robert C. Martin, l'**arquitectura neta** [2] estableix una organització clara dels components en capes circulars i concèntriques, seguint una estructura similar a una ceba. Aquesta arquitectura es compon de quatre capes principals, ordenades de l'exterior cap a l'interior:

1. Regles del negoci de l'empresa.
2. Regles de negoci de l'aplicació.
3. Adaptadors d'interfície.
4. *Frameworks* i connectors.

Vegeu la Fig. 1.

### 2.1 Nivell d'abstracció i estabilitat

Les capes exteriors, amb un nivell d'abstracció més baix, soLEN ser més inestables que les capes interiors, que tenen un nivell d'abstracció més alt. Un exemple clar de component inestable és la Interfície d'Usuari (UI). Les modificacions a la UI són habituals i necessàries, però han de poder realitzar-se sense afectar la part estable de l'aplicació, com la lògica del negoci. Un altre exemple de component inestable és la capa de persistència, que depèn d'infraestructura externa (per exemple, la llibreria *Firestore*), la qual no controlem com a desenvolupadors.

### 2.2 Regla de dependència i DIP

L'arquitectura neta es basa en la **regla de dependència**, que estableix que *les dependències que travessen fronteres entre capes han de fer-ho seguint el sentit de l'estabilitat*. Això implica que els components de les capes exteriors podEN dependre dels components de les capes interiors, però no a l'inrevés.

Tot i això, les capes interiors podRIEN necessitar comunicar-se amb les exteriors. És a dir, es vol permetre el flux de control de dins cap a fora, però mantenint la dependència en sentit contrari, per no vulnerar la regla de dependència. Per aconseguir-ho, apliquem el **d'inversió de dependències (DIP)**: la capa interior defineix una abstracció (com una interfície en Java) que la capa exterior ha d'implementar. Així, el component de la capa interior pot interactuar amb

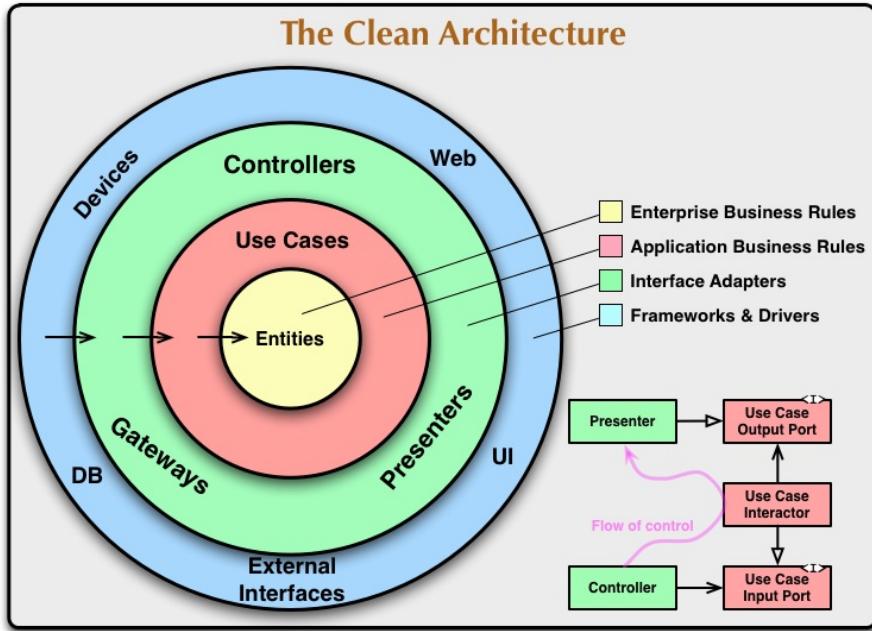


Figure 1: Divisió en capes circulars concèntriques de l'arquitectura neta. Les fleixes indiquen el sentit de les dependències.

l'abstracció sense dependre directament de la seva implementació a la capa exterior.

A continuació, veiem com aplicar aquests conceptes a una aplicació pràctica amb una UI basada en *Android* i una capa de persistència amb *Firebase*.

### 3 Aplicant l'arquitectura neta

Per tenir més control del sentit de les dependències entre capes, implementarem les capes mitjançant la definició de mòduls. A continuació, veurem quins mòduls tindrem i com podem crear-los. Més tard, concretarem quins components formaran part de cada mòdul.

### 3.1 Estructuració del projecte Android en mòduls

Els mòduls ens permetran una més clara separació i control sobre les dependències entre capes. Dins de cada mòdul, podrem – obviament – tenir-hi paquets per separar els components de la mateixa capa.

Tal com mostrem a la Fig. 2, estructurarem el codi en els mòduls `app` (o

presentation), **features**, **domain** i **data**. Seguim la codificació de colors de l'arquitectura neta per aclarir a quina capa de l'arquitectura correspon cada component de cadascun dels mòduls:

- **presentation** i **data** són frameworks (és a dir, llibreries) i connectors.
- **features** són regles de negoci de l'aplicació.
- **domain** són regles de negoci de l'empresa.

Amb les fletxes de la mateixa figura, ja veiem el sentit de les dependències i com es respecta la regla de dependència.

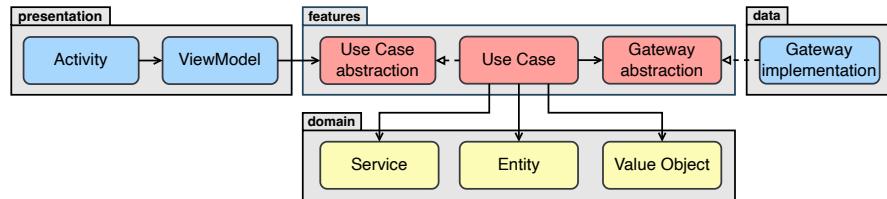


Figure 2: Una aplicació d'Android amb mòduls: **presentation**, **features**, **domain** i **data**. La divisió per colors indica el nivell d'abstracció dels components continguts en els mòduls.

### 3.1.1 Creació de mòduls amb Android Studio

Per a crear un mòdul, fem `File > New > New Module...` per obrir la finestra de la Fig. 3. A l'esquerra hi veureu el tipus de mòdul. En funció del que hagi de contenir el mòdul, haurà de ser d'un tipus o d'un altre:

- **app** (reanomenat **presentation**) serà un mòdul de tipus "Phone & Tablet". El mòdul ja ve creat quan creem el projecte. Per canviar-li el nom, cliquem el botó dret a sobre del nom del mòdul i fent `Refactor > Rename`.
- **features** i **domain** seran mòduls de tipus **Java o Kotlin Library**, ja que, no contenen codi Android. El nom del mòdul s'especifica en el camp "Library name" (per defecte, "lib") i al "Package name" li incloem el nom de l'aplicació (p. ex. `edu.ub.pis2425.\textbf{nomdelprojecte}.features`).
- **data** és de tipus **Android Library** perquè, en el nostre cas, farà ús de *Firebase*, un *framework* que depèn de llibreries d'Android.

Cada mòdul tindrà el seu propi fitxer gradle. Veure la Fig. ???. Això permet afegir dependències als mòduls de manera individual.

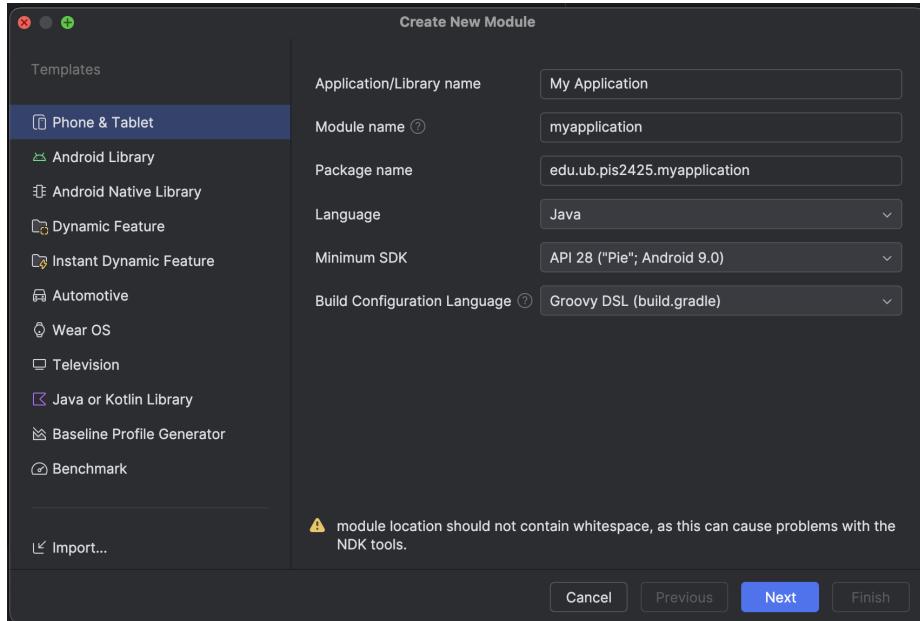


Figure 3: Finestra de creació de mòduls. Veure text pels detalls.

### 3.1.2 Dependències entre mòduls

A més de les dependències de llibreries externes, a cada mòdul li hem d'afegir les dependències dels altres mòduls. A la mateixa secció dependencies dels gradle a nivell de mòdul, hi podem afegir les dependències dels mòduls.

Per exemple, si al mòdul **features** li volguéssim afegir la dependència del mòdul **domain**, modificaríem la secció dependencies del fitxer `build.gradle` (Module :**features**) afegint-hi la línia:

```
dependencies {
    implementation project(:domain)

    ...
}
```

A continuació, llistem totes les dependències entre els mòduls per tal de poder muntar l'arquitectura que es mostrava a la Fig. 2:

- `domain` no depen de cap altre mòdul.
- `features` depèn de `domain`.
- `data` depèn de `domain`.

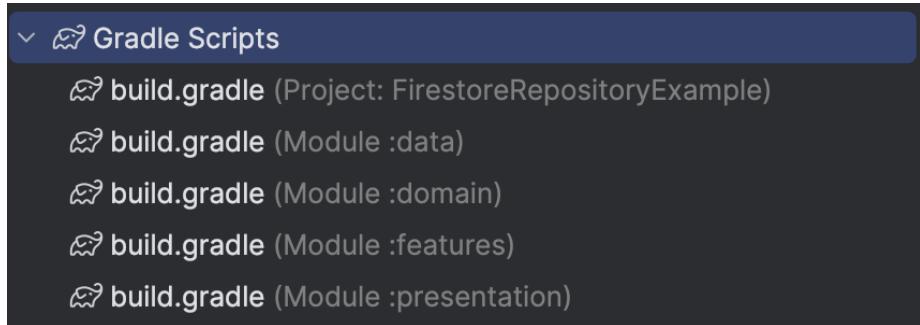


Figure 4: Fitxers gradle del nostre projecte. Tenim el fitxer gradle general i un per cadascun dels mòduls.

- presentation depèn de tota la resta de mòduls (domain, features i data), perquè és el mòdul que farà la injecció de dependències – tal com expliquem a la Sec. 3.3.

## 3.2 Components dels mòduls

A continuació, expliquem què hauria de contenir cadascun dels mòduls.

### 3.2.1 Domini

Conté els *objectes del domini*. Quins? L’arquitectura neta sovint s’aplica juntaament amb alguns patrons tàctics del *disseny guiat pel domini* (o *domain-driven design*) [1]. Un d’ells és de distingir els objectes del domini en *entitats*, *objectes de valor*, *agregats* i *serveis*. En aquesta secció, fem un repàs de què són cadascun d’ells.

**Entitats.** Tenen una identitat única representada per un identificador immutable, que permet que si una entitat veu modificat qualsevol dels seus altres atributs, pugui seguir sent reconeguda pel sistema com la mateixa identitat. Pensem en l’exemple d’un Client que canviï el seu correu, nom d’usuari o adreça postal.

⚠️ Les referències d’objecte no són identificadors d’entitat vàlids perquè canvien quan les entitats es guarden i es recarreguen de persistència (es crea un objecte nou i, per tant, una referència nova).

**Objectes de valor.** No tenen identificador, sinó que la seva identitat ve donada pel valor conjunt de tots els seus atributs. Per exemple, un Price (double amount, String currency) seria un clar objecte de valor. Dos objectes amb la mateixa quantitat d’una certa divisa són, essencialment, el mateix preu. I si canvia un dels atributs, la quantitat o la divisa, estem parlant d’un objecte

fonamentalment diferent. Per tant, els objectes de valor són immutables, no permetent canviar-los cap del valor dels seus atributs.

☞ Si tenim un objecte `Preu` i haguéssim de canviar-li la quantitat, crearíem un nou objecte. Això "embruta" el codi client (el que fa ús de l'objecte de valor). En lloc d'això, podríem delegar la creació del nou objecte a la instància de l'objecte de valor antic. Per exemple: `public Price addAmount (double amount) { return new Price(this.amount + increment, this.currency); }`.

**Agregats.** Un agregat és una agrupació conceptual d'entitats i objectes de valor que formen una unitat de consistència i integritat dins del domini. Cada agregat té una entitat arrel que actua com a punt d'entrada i és responsable d'orquestrar les operacions que afectin l'estat de les dades dels objectes agrupats en l'agregat. Cal evitar, doncs, exposar fora de l'agregat qualsevol entitat que no sigui l'entitat arrel o objectes de valor. A més, els agregats són unitats de persistència i, per tant, tampoc hem de permetre que guardin referències d'altres agregats. Si un agregat ha de tenir accés a un altre agregat, sempre podem fer ús d'una referència transitiva (passant-li-la per un mètode i descartant-la després).

☞ Els objectes de valor es poden exposar fora dels agregats, precisament, per la seva condició immutable i formar part de diversos agregats alhora.

**Serveis.** Implementen lògica de coordinació entre agregats. Els serveis poden invocar altres serveis i s'implementen com a façanes sense estat.

### 3.2.2 Features

Inclou, principalment, els *casos d'ús*, abstraccions dels casos d'ús i abstraccions de *gateway* que les capes externes implementen. A vegades també tindrem *models de petició i/o models de resposta*, que són objectes que s'utilitzen per a facilitar la comunicació amb els casos d'ús.

**Casos d'ús.** Són serveis exposats a l'usuari, els quals poden – alhora – delegar en altres casos d'ús, serveis, agregats, etc. Fem la distinció amb els serveis per seguir el paradigma de l'*Screaming Architecture*<sup>1</sup>, que vol dir que mirant el codi tindrem una idea clara i ràpida de què fa.

**Abstraccions dels casos d'ús.** Definirem abstraccions també pels casos d'ús. En aquest cas, l'abstracció no les definim per a complir amb la regla de dependència de l'arquitectura neta, ja que, qui crida els casos d'ús és un model de vista. Ho fem per a millorar la testabilitat dels casos d'ús. Hem dit que els casos d'ús poden delegar (dependre) d'altres casos d'ús i, per a testejar-los, podrem substituir les abstraccions per implementacions mock dels casos d'ús.

---

<sup>1</sup> Article sobre l'*Screaming Architecture* per Robert C. Martin: <https://blog.cleancoder.com/uncle-bob/2011/09/30/Screaming-Architecture.html>

**Model de petició o resposta.** Els models de petició són objectes de tipus contenidor per a encapsular el pas de paràmetres als casos d'ús. D'altra banda, els models de resposta serveixen per a encapsular les dades de retorn del cas d'ús. Pot valer la pena si s'han de passar o retornar tipus de dades més complexes que variables primitives, col·leccions o objectes del domini.

**Abstraccions dels gateways.** Les abstraccions que permeten el control de les implementacions dels *gateways*. Els gateways són mecanismes de sortida, per tant, repositoris o APIs externes.

### 3.2.3 Presentació

El mòdul de presentació contindrà els components de la UI (activitats o fragments<sup>2</sup>) i els d'adaptació que tenen a veure amb la UI (adaptadors de vista recicitable, models de vista i objectes de presentació). No entrarem a explicar què són cadascuna d'aquestes coses perquè ja ho heu vist en pràctiques introductòries anteriors. Potser cal parlar del que anomenem *objectes de presentació*.

**Objectes de presentació (POs).** Els objectes de presentació són classes sense comportament que utilitzem per representar informació dels objectes del domini, sense exposar-los fora de les capes del domini (*domain*) i de l'aplicació (*features*). *Quines implicacions tindria fer-ho?* La primera implicació és la possibilitat d'inconsistències en les dades. Les activitats podrien causar canvis en els objectes del domini que no farien persistir, perquè no tenen capacitat de fer-ho, podent generar discrepàncies entre els objectes en la memòria i la persistència. Una altra implicació és la vulneració de la regla de dependència. Els objectes de presentació es poden passar entre activitats o fragments, però això obliga a fer-los implementar la interfície *Parcelable*. Aquesta interfície és pròpia d'*Android SDK* i crea una dependència que no volem pels objectes del domini.

Els encarregats de transformar objectes del domini en POs, en el context de la programació Android, seran els models de vista. Un cop feta la transformació, el model de vista li podrà – si cal – guardar-se el PO i, també, fer-li arribar a l'activitat corresponent.

☞ Per convertir d'objectes del domini a objectes de presentació, podem utilitzar – com feiem per convertir DTOs a objectes del domini – un mapejador genèric *ModelMapper*. Veure la classe `presentation.pos.mappers.DomainToPOMapper` en el codi base.

### 3.2.4 Data

Contindrà les implementacions de *gateways* (p. ex. repositoris) i els *objectes de transferència de dades* (DTOs).

---

<sup>2</sup>Introduïts més endavant, a la Sec. 4.1.

**Implementacions de passarel·les (o *gateways*).** Un clar exemple de passarel·la és – com bé sabeu – un repositori. En l'exemple del repositori de productes, tindreu `ProductRepository` al mòdul `domain` i `ProductRepositoryImpl`<sup>3</sup> aquí, al mòdul `data`.

☞ En el cas de fer ús de bases de dades relacionals, els repositoris podrien delegar en DAOs. Ara bé, assumint que feu servir `Firestore`, no els necessitareu. L'accés a la `Firebase` l'implementarà la passarel·la (p. ex. `ProductFirebaseRepository`) directament, gràcies a la capacitat de l'API de `Firebase` de mapejar automàticament els objectes de Java a documents i viceversa.

**DTOs.** Els DTO són classes contenidores que s'utilitzen per carregar-hi les dades de persistència i, en cas de necessitat, transferir-les entre components de persistència relacionats. Quines implicacions tindria no utilitzar DTOs? Si carreguessim directament objectes del domini, aquests estarien condicionats per les dades i la seva estructura. L'altra és, de nou, la vulneració de la regla de dependència. Com ara, quan el *framework* utilitzat per la persistència, ens obliga a que els objectes on carreguem les dades depenguin d'ell. És el cas de `Firebase` amb el decorador `@DocumentId`, el qual utilitzem per a carregar l'identificador del document en l'atribut de l'objecte carregat. Per tant, triem que siguin els DTOs qui tinguin aquesta dependència i no els objectes del domini directament.

☞ La conversió de DTO → objectes del domini la fan els mateixos repositoris amb la classe de mapeig genèrica `data.dtos.firebaseio.mappers.DTOToDomainMapper`, explicada en detall a la pràctica introductòria de `Firebase`.

Arribats a aquest punt, seria un bon moment per a revisar el codi base i intentar trobar cadascun dels elements descrits fins ara.

### 3.3 Injecció de dependències a Android

La **injecció de dependències** és la pràctica de crear els components i passar-los les seves dependències (altres components) com a paràmetres del constructor. El que normalment s'injecta són abstraccions, permetent que els components que estan sent injectats no estiguin acoblats a implementacions concretes. Això ens serà útil per:

- Injectar repositoris als casos d'ús.
- Injectar casos d'ús a altres casos d'ús.
- Injectar casos d'ús als models de vista.

---

<sup>3</sup>Si preferiu dir-li `IProductRepository` a l'abstracció `ProductRepository` a la implementació també podeu fer-ho. Només us demanem ser consistents amb la nomenclatura que feu servir.

Aquestes injeccions hauran de tenir efecte el més aviat possible just després de l'inici de l'execució de l'aplicació i, per tant, abans que l'activitat principal iniciï el seu cicle de vida i es cridi el mètode `onCreate` de l'activitat.

Fins ara hem obviat que existís cap altre objecte amb un cicle de vida que s'iniciés abans que el de la pròpia activitat principal. Bé doncs, sí que n'hi ha un, el que representa la pròpia aplicació. Es tracta d'un objecte de la classe `Application`, el qual té també els seus mètodes de callback (un d'ells `onCreate`).

El que farem, doncs, serà estendre la classe `Application` de l'*Android Sdk* i sobrecarregar-li el mètode `onCreate` per a crear un objecte d'una classe pròpia `AppContainer` on hi inicialitzarem tots els nostres components i hi farem la injecció de dependències:

```
1 import android.app.Application;
2
3 /**
4  * To be able to perform manual dependency injection.
5  * Source:
6  * https://developer.android.com/training/dependency-injection/
7  * manual
8 */
9
10 public class MyApplication extends Application {
11
12     public AppContainer appContainer;
13
14     @Override
15     public void onCreate() {
16         super.onCreate();
17         // Initialize the AppContainer when the application starts
18         appContainer = new AppContainer();
19     }
20
21     @SuppressWarnings("unused")
22     public AppContainer getAppContainer() {
23         return appContainer;
24     }
25
26     @SuppressWarnings("unused")
27     public AppContainer.ViewModelFactory getViewModelFactory() {
28         return appContainer.viewModelFactory;
29     }
30 }
```

Fixeu-vos que l'únic que fem és crear un objecte de la classe `AppContainer` i afegir-lo com atribut de `MyApplication`. També definim un parell de *getters* que ja veurem més tard perquè ens serviran. Centrem-nos en la implementació d'`AppContainer`:

```

1 public class AppContainer {
2     /* Dependency injection */
3     // Repositories
4     private final ClientRepository clientRepository = new
5         ClientFirestoreRepository();
6     private final ProductRepository productRepository = new
7         ProductFirestoreRepository();
8     // Use cases
9     private final LogInUseCase logInUseCase = new
10        LogInUseCaseImpl(clientRepository);
11     private final SignUpUseCase signUpUseCase = new
12        SignUpUseCaseImpl(clientRepository);
13     private final GetAllProductsUseCase getAllProductsUseCase =
14         new GetAllProductsUseCaseImpl(productRepository);
15     private final GetProductsByNameUseCase
16         getProductsByNameUseCase = new
17             GetProductsByNameUseCaseImpl(productRepository);
18     private final RemoveProductUseCase removeProductUseCase = new
19         RemoveProductUseCaseImpl(productRepository);
20
21     ...
22 }

```

Veieu com creem i injectem els objectes seguint l'ordre de les dependències: si un cas d'ús depèn d'un repositori, primer creem el repositori i després el cas d'ús (passant-li el repositori pel constructor). Com hem dit abans, injectem abstraccions.

Ara faltaria **injectar els casos d'ús als models de vista**. Però com que els models de vida tenen el seu cicle de vida independent (que no controlen nosaltres), no té sentit crear-los i injectar-los ara mateix aquí. El que farem serà crear una *factory* de models de vista que permeti, de manera reflexiva, obtenir-los injectats. Aquesta és la resta de l'AppContainer on fem això:

```

1 public class AppContainer {
2     /* Dependency injection */
3     ...
4     // ViewModel factory: initialization of internal class (see
5         below)
6     public ViewModelFactory viewModelFactory = new
7         ViewModelFactory(this);
8
9     /* Internal class definitions */
10    // ViewModel factory
11    public static class ViewModelFactory implements
12        ViewModelProvider.Factory {
13        private final AppContainer appContainer;
14
15        public ViewModelFactory(AppContainer appContainer) {
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
311
312
313
313
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1
```

```

13     this.appContainer = appContainer;
14 }
15
16 @SuppressWarnings("unchecked") // This would be helpful for
17     lint warnings for casts.
17 @NonNull
18 @Override
19 public <T extends ViewModel> T create(Class<T> modelClass)
20 {
21     if (modelClass.isAssignableFrom(SignUpViewModel.class)) {
22         return (T) new SignUpViewModel(appContainer.
23             signUpUseCase);
22     } else if (modelClass.isAssignableFrom(LogInViewModel.
24         class)) {
23         return (T) new LogInViewModel(appContainer.logInUseCase
25             );
24     } else if (modelClass.isAssignableFrom(ShoppingViewModel.
25         class)) {
26         return (T) new ShoppingViewModel(
26             appContainer.getAllProductsUseCase,
27             appContainer.getProductsByNameUseCase,
28             appContainer.removeProductUseCase
29         );
30     } else if (modelClass.isAssignableFrom(
31         ViewProductDetailsViewModel.class)) {
32         return (T) new ViewProductDetailsViewModel();
33     }
33     throw new IllegalArgumentException("ViewModel Not Found")
34     ;
34 }
35 }
36 }
37 }
```

Aquesta és la *factory* que passarem als ModelViewProvider que fem anar a les activitats per inicialitzar els seus corresponents models de vista. De fet, aquesta **factory global**, substitueix la definició de *factories* individuals que definíem als models de vista (explicat a la pràctica introductòria dels models de vista). Anem a veure el cas concret de la creació del LogInViewModel a una LogInActivity:

```

1 public class LogInActivity extends Activity {
2
3     /* ViewModel */
4     private LogInViewModel logInViewModel;
5
6     ...
7
8     /**
9      * Initialize the viewmodel and its observers.
```

```

10    */
11    private void initViewModel() {
12        /* Init viewmodel */
13        logInViewModel = new ViewModelProvider(
14            this,
15            ((MyApplication) getApplication()).getViewModelFactory
16            ())
17            .get(LogInViewModel.class);
18
19        initObservers();
20    }
21
22    ...
}

```

I el constructor del `LogInViewModel` rebria el cas d'ús que injectem reflexivament des de la *factory* global:

```

1 public class LogInViewModel extends ViewModel {
2     /* Use cases */
3     private final LogInUseCase logInUseCase;
4
5     ...
6
7     /* Constructor */
8     public LogInViewModel(LogInUseCase logInUseCase) {
9         this.logInUseCase = logInUseCase;
10        ...
11    }
12
13    ...
14 }

```

D'aquesta manera, ens quedaria ja tot injectat i lligat: l'activitat `LogInActivity` tindria assignat el model de vista `LogInViewModel`, el model de vista el cas d'ús `LogInUseCase` injectat i, alhora, el cas d'ús `LogInUseCase` tindria el repositori `ClientRepository`.

## 4 Observacions addicionals

A més de l'arquitectura neta, el codi proporcionat també inclou altres novetats. Concretament, l'ús de *fragments* i del *controlador de navegació* per la UI, que seran **OPCIONALS** pel desenvolupament del vostre projecte. Per tant, no els cobrirem en profunditat a les pràctiques introductòries. Tot i això, us en fem unes pinzellades ràpides i us proporcionem recursos per investigar-ho pel vostre compte amb més profunditat.

## 4.1 Fragments

Els *fragments*, igual que les activitats en Android, tenen una disposició associada, definida mitjançant arxius XML, i una lògica de control, implementada en codi Java. Aquests fragments disposen d'un cicle de vida propi, però són sensibles al cicle de vida de l'activitat que els conté. Els fragments tenen la capacitat de transitar entre ells, és a dir, poden canviar de manera dinàmica, enviar i rebre informació, entre d'altres operacions. Per tant, poden considerar-se com a "*activitats petites*" dins de les activitats que proporcionen diversos avantatges:

- **Modularitat:** Permeten dividir activitats grans en fragments més petits i manejables.
- **Reusabilitat:** Un mateix fragment pot ser utilitzat en diverses activitats.
- **Integració amb el controlador de navegació:** Els fragments poden ser fàcilment gestionats mitjançant el sistema de navegació d'Android.

Avui en dia, és habitual trobar aplicacions d'una sola activitat que actuen com a contenidors de fragments i que inclouen una barra de navegació, com ara una BottomNavigationView, que permet als usuaris seleccionar quin fragment es mostra en el contingut.

Trobeu informació sobre fragments a la pàgina de desenvolupadors: <https://developer.android.com/guide/fragments>.

## 4.2 Controlador de navegació

El controlador de navegació ofereix diverses característiques. Entre les quals, el maneig transparent de transaccions entre fragments. Les transaccions són les transicions de canvi d'un fragment per un altre en un contingut de fragments, les quals es poden fer manualment amb *transaccions* (veure <https://developer.android.com/guide/fragments/transactions>), similar a com fan les *intencions explícites* de les activitats, o utilitzant el controlador de navegació (veure NavController a <https://developer.android.com/guide/navigation>).

El controlador de navegació és recomanable, sobretot, quan hi ha una vista de navegació: BottomNavigationView (barra inferior) o NavigationDrawer (panell lateral que es mostra i s'oculta).

Trobareu el codi i fitxers relacionats amb aquesta part al mòdul **presentation**, principalment:

- `java/edu/ub/pis2425/cleanarchitectureexample/presentation/ui/activities/MainActivity.java` i el seu layout a `res/layout/activity_main.xml`. Mireu també l'AndroidManifest.xml per veure com ha quedat de net.

- `java/edu.ub.pis2425.cleanarchitectureexample.presentation.ui.fragments/*.java` i els seus layouts a `res/layout/fragment_*.xml`.
- `res/navigation/nav_graph_main.xml`
- `res/menu/bottom_navigation_menu.xml`
- `res/values/dimen.xml`
- `res/values/strings.xml`

## 5 Exercicis

Aquesta pràctica introductòria no inclou exercicis específics. El seu objectiu és que el codi base us serveixi com a recurs i guia per a la implementació del vostre projecte.

Podeu executar l'aplicació i accedir-hi amb les credencials "admin/admin". Tanmateix, cal tenir en compte que el permís d'escriptura de la base de dades (*Firebase*), a la qual està connectat el projecte, està deshabilitat. Així doncs, la funcionalitat d'esborrar un producte de la llista no funcionarà si no teniu connectada la vostra pròpia base de dades, tal com es detalla a la pràctica introductòria sobre *Firebase*.

## References

- [1] Eric Evans. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004. [6](#)
- [2] Robert C Martin. Clean architecture, 2017. [2](#)