Cuprins

Introducere

1. Concepte generale
   1. Injectarea dependențelor
      1. Definitie
      2. Scop
      3. Prezentare generala
      4. Avantaje si dezavantaje in folosirea injectarii dependențelor
      5. Structura
   2. Arhitecturi
      1. Model-View-Controller (MVC)
      2. Model-View-Presenter (MVP)
      3. Model-View-ViewModel (MVVM)
2. Framework-uri
3. Kotlin, Android
4. Dezvoltarea aplicatiei
5. Concluzii

Introducere

1. Concepte generale
   1. Injectarea dependențelor
      1. Definitie

În ingineria software, conceptul de injectare a dependențelor (eng. *dependency injection*) este o tehnică prin care un obiect sau o metodă statică furnizează dependențele unui alt obiect. O dependență este, practic, un obiect ce poate fi folosit, iar injectarea se referă la transmiterea unei dependențe unui alt obiect care, mai departe, o poate folosi. Acea dependență devine, astfel, o parte din starea obiectul căruia îi este injecată dependența.[1] Pasarea dependenței obiectului, in locul construirii acelei dependențe pe catre obiectul care o solicită, este o cerință fundamentală a acestui concept.

Această cerință înseamnă că utilizarea valorilor construite în interiorul clasei este interzisă. Obiectele ar trebui să accepte valori din exterior, aceasta permițându-le să lase furnizarea dependențelor in sarcina unui alt obiect. Intenția din spatele acestui concept este de a decupla obiectele până în punctul în care codul niciunei clase nu va trebui sa fie schimbat pentru simplul motiv ca un obiect de care aceasta depinde trebuie sa fie schimbat la unul diferit.

Injectarea dependențelor este o forma a tehnicii mai general, cunoscut sub numele de inversiune de control (eng. *inversion of control*). Ca si alte forme a inversiunii de control, injectarea dependențelor susinte principiu inversarii dependențelor. Obiectul injectat deleagă responsabilitatea furnizării dependențelor sale unui obiect exterior. Obiectului injectat nu îi este permis să apeleze codul injectorului.[2] Codul de injectare construieste serviciile si apeleaza clientul pentru a i le injecta, ceea ce inseamna ca nu este responsabilitatea codului clientului sa stie codul de injectare. De asemenea, clientul nu are nevoie sa cunoasca modalitatea in care sunt cosntruite serviciile sau care servicii actuale le foloseste. Tot ceea ce clientul trebuie sa stie este interfetele serviciilor, deaorece acestea definesc modalitatea in care clientul poate folosi aceste servicii. Aceasta separa reponsabilitatea de folosire si construire.

Există 3 mijloace prin care un client poate accepta o injectare a dependențelor : bazată pe setter, bazată pe interfață sau bazată pe constructor. Injectarea pe setter și prin constructor diferă în principal prin momentul în care pot fi folosite. Injectarea prin interfață diferă prin faptul că dependența își poate control propria injectare. Fiecare necesită ca un cod de construcție (injector) preia responsabilitatea de a introduce un obiect și dependențele sale unul altuia.[3]

* + 1. Scop

Injectarea dependențelor rezolva probleme precum : [4]

* Cum poate fi o aplicatie independenta de modul in care obiectele acesteia sunt create
* Cum poate fi o clasa independenta de modul in care obiectele de care aceasta are nevoie sunt create
* Cum poate fi specificat modul in care obiectele sunt create in fisiere de configurare separate
* Cum poate o aplicatie sa suporte diferinte configuratii

Creare obiectelor direct in interiorul clasei care are nevoie de acestea este inflexibil, deoarece leaga clasa de niste obiecte particulare si face imposibila schimbarea instantierii acesteia mai tarziu, independent de clasa si fara sa fie necesara schimbarea clasei. Astfel, clasa nu mai este reutilizabila daca alte obiecte sunt necesare, fiind in acelasi timp greu de testat, deoarece obiectele reale nu pot fi inlocuite cu obiecte sablon (mock objects).

Tehnica injectarii dependențelor descrie cum pot fi rezolvate probleme precum :

* Definirea unui obiect injector separat care creaza si injecteaza obiectele de care o clasa are nevoie
* O clasa accepta obiectele de care are nevoie direct de la un injector, in loc sa le creeze direct

Asta face o clasă independentă de modul in care obiectele acesteia sunt create (care clase concrete sunt instantiate).

O clasă nu mai este responsabilă de crearea obiectelor de care are nevoie si nu mai trebuie să delege instanțierea unui obiect creator, precum în șablonul Abstract Factory.[5]

* + 1. Prezentare generala

Injectarea dependențelor separa crearea dependențelor unui client de coportamentul clientului, permitand programelor sa fie slab cuplate si sa urmeze principiile inversarii dependențelor si pe cel al responsabilitatii unice (single responsibility).

Injectia, unitatea de baza a injectarii dependențelor, nu este un termen nou sau un mecanism specific, el functionand in acelasi mod in care pasarea parametrilor functioneaza.

O injectie de asemenea defineste ceea ce este in controlul furnizarii (niciodata clientul) si este indepenta de cum este realizata furnizarea, fie ca este prin valoare sau prin referinta.

Injectarea dependențelor implica 4 roluri :

* Serviciile (obiectele) care sunt folosite
* Clientul care depinde de serviciile pe care le foloseste
* Interfetele care definesc modul in care clientul poate folosi serviciile
* Injectorul, care este responsabil de construirea serviciilor si furnizarea lor catre client

Orice obiect care poate fi folosit poate fi considerat ca fiind un serviciu. Orice obiect care foloseste alte obiecte poate fi considerat un client. Numele nu au de a face cu scopul pentru care sunt create, ci mai degraba se refera la rolul pe care obiectele il joaca in injectare.

Interfetele definesc tipul dependențelor care sunt asteptate de client. Acestea pot fi tipuri de interfete, implementate de servicii, dar pot fi, de asemenea, clase abstracte sau chiar servicii concrete, cu toate ca aceasta ar incalca principiul injectarii dependențelor si ar duce la o cuplare mai putin dinamica, ceea ce ar ingreuna testarea. Este doar necesar ca obiectul injectat sa nu stie care sunt acestea si astfel, nu le trateaza niciodata ca si clase concrete, prin construirea sau extinderea lor, spre exemplu.

Clientul nu trebuie sa aiba cunostinte concrete legate de implementarea dependențelor sale. Trebuie sa cunoasca doar numele interfetei si API-ul expus de aceasta. Ca si rezultat, clientul nu va trebuie sa se schimbe chiar daca ceea ce este in spatele interfetei se schimba. Cu toate acestea, daca interfata este refactorizata de la o clasa la o interfata sau invers, clientul va trebui sa fie recompilat.

Injectorul furnizeaza serviciile clientului si de multe ori construieste clientul. Un injector poate conecta un un graf de obiecte foarte complex, tratand un obiect ca si client, iar mai tarziu ca un serviciu pentru un alt client. Injectorul poate reprezenta, de fapt, mai multe obiecte care interopereaza, dar nu poate fi client.

Injectarea dependențelor poate fi aplicata ca si disciplina, una care cere ca toate obiectele sa isi separe constructia de comportament.

* + 1. Avantaje si dezavantaje in folosirea injectarii dependențelor
    2. Structura
  1. Arhitecturi
     1. Model-View-Controller (MVC)
     2. Model-View-Presenter (MVP)
     3. Model-View-ViewModel (MVVM)

Bibliografie

1. I.T., Titanium. "James Shore: Dependency Injection Demystified". www.jamesshore.com.

2. "HollywoodPrinciple". http://c2.com

3. https://martinfowler.com/articles/injection.html

4. "The Dependency Injection design pattern - Problem, Solution, and Applicability". w3sDesign.com

5.  Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley. pp. 87ff