Macedolean Delia –Maria
Anul 4, IS, grupa 3.2

Tema 2 PSSC – Inversion of Control

Inversarea controlului (IoC) este un principiu de proiectare în care porțiunile personalizate scrise ale unui program de calculator primesc fluxul de control dintr-un cadru generic. O arhitectură software cu acest design inversează controlul în comparație cu programarea tradițională procedurală. În programarea tradițională, codul personalizat solicită ca bibliotecile reutilizabile să aibă grijă de sarcinile generice. În schimb cu inversarea controlului, cadrul este cel ce face apel la codul personalizat.

Inversarea controlului este folosită pentru a crește modularitatea programului și pentru a il face extensibil având aplicații în programarea orientată pe obiecte.

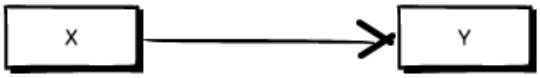
Termenul de inversare a controlului este legat de principiul inversării dependenței, dar diferit de acesta. Principiul inversării dependintei se referă la decuplarea dependențelor dintre straturile de nivel înalt și nivelurile joase prin intermediul abstractizărilor partajate. Conceptul general se referă, de asemenea, la programarea bazată pe evenimente, prin faptul că este adesea implementată utilizând IoC, astfel încât codul personalizat se referă în general numai la gestionarea evenimentelor, în timp ce bucla evenimentului și expedierea evenimentelor / mesajelor sunt tratate de cadrul sau mediul de executare

Inversarea controlului are conotația puternică în care codul reutilizabil și codul specific pentru probleme sunt dezvoltate independent chiar dacă funcționează împreună într-o aplicație. Cadrele de software, callback-urile, planificatorii, buclele de evenimente, injectarea dependenței și metoda șablonului sunt exemple de modele de design care urmează principiului inversării controlului, deși termenul este cel mai frecvent utilizat în contextul programării orientate pe obiecte.

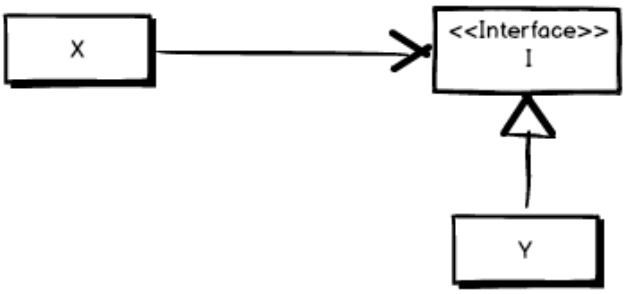
Exemple:

1. Pentru a ajunge la locul de muncă, se presupune că trebuie să conducem mașina până acolo. Asta înseamnă că noi controlam autoturismul. Principiul IoC sugerează să inversăm controlul, ceea ce înseamnă că, în loc să conducem noi mașina, închiriem un taxi unde o altă persoană va conduce mașina. Astfel are loc inversarea controlului de la noi la șoferul taxiului. Nu mai este nevoie să conducem noi mașină ci să lăsam șoferul să isi facă treaba, timp în care noi ne putem concentra asupra lucrării principale.

2. Un exemplu de dependință este atunci când clasa X folosește clasa Y



Consumatorul X, are nevoie de clasa pe care o consumă, Y, pentru a atinge un obiectiv. Dar cu toate acestea, X nu are nevoie să știe despre Y. Este de ajuns pentru X să știe că folosește ceva care are comportamentul, metodele și proprietățile lui Y, dar nu și detalii despre o implementare concretă. Prin extragerea unei definții abstracte a comportamentului lui Y, X poate consuma orice implementare a acestei definiții (una din ele fiind Y), fară a știi nimic despre Y.



Y este o implementare a interfeței I, iar X o foloseste pe aceasta din urmă. În timp ce este posibil ca X să il folosească pe Y, acesta nu o face direct, ci prin intermediul interfeței fară să știe de Y în fapt. Această implementare a interfeței poate fi Y, dar poate fi și A, B sau C, dacă acestea implementează aceeași interfață.

Beneficii ale inversării controlului

- Un prim beneficiu este că X nu depinde de Y și de aceea o schimbare în Y e foarte puțin probabil să mai necesite o schimbare și in X.
- X devine mult mai flexibil și poate opta pentru o altă implementare a interfeței, nu doar Y.
- Y ar putea fi un serviciu care trimite emailuri. Schimbând implementarea interfeței I folosită de X, noua implementare ar putea trimite tweeturi de data aceasta. X știe doar că apelează o metodă care trimite un mesaj clientului.
- Un alt beneficiu, este că putem izola cod pentru când scriem teste unitare. În scenariul de mai sus și tinând cont că testele unitare se presupune că testeaza doar o anumită parte din cod, testul nu mai trebuie să se bazeze că serviciul de email funcționează, ci doar că acea parte din cod merge, indiferent de ce se întamplă în spate. Cu IoC, X nu mai trebuie să se bazeze pe Y, ci doar pe una dintre implementările lui I, oricare ar fi aceasta. Asta înseamnă că în partea de setup a testelor, putem folosi un mock al interfeței I în loc de obiectul în sine, lucru care ne permite să testam doar codul din X, asigurandu-ne totodată că Y e folosit correct.